Владимир ПАРОНДЖАНОВ

КАК УЛУЧШИТЬ РАБОТУ УМА

Алгоритмы без программистов — это очень просто!

НОВЫЕ СРЕДСТВА
ДЛЯ ОБРАЗНОГО
ПРЕДСТАВЛЕНИЯ ЗНАНИЙ,
РАЗВИТИЯ ИНТЕЛЛЕКТА
И ВЗАИМОПОНИМАНИЯ

**Академия народного хозяйства
при Правительстве Российской Федерации**

**Москва
Издательство «Дело»
2001**

Перед вами второе издание книги “Как улучшить работу ума”, вызвавшей большой интерес специалистов и читающей публики. В ней удачно сочетаются рассказы об **алгоритмах** и новейшие идеи о повышении **творческой силы ума.**

Мы живем в мире алгоритмов, но знаем о них удивительно мало. Многие люди всю жизнь пользуются алгоритмами, не догадываясь об этом. Между тем алгоритмы играют огромную роль в жизни общества. Они оказывают заметное влияние на эффективность экономики и уровень жизни. К сожалению, многие алгоритмы и программы похожи на загадочный ребус: они непонятны никому, кроме горстки их создателей. Непонимание порождает путаницу и досадные ошибки. Чтобы поправить дело, надо сделать алгоритмы “дружелюбными”. Это позволит превратить алгоритмы-головоломки в наглядные алгоритмы-картинки, обеспечивающие *быстрое и глубокое понимание*. Глубина понимания сложных проблем — как раз то, чего всем нам (от студента до министра) ой как не хватает!

“Дружелюбные” алгоритмы пишут на **эргономичных** графических языках. Они создают повышенный интеллектуальный комфорт, улучшают работу ума, повышают продуктивность труда. С их помощью вы научитесь легко и быстро, затратив минимум усилий, решать сложнейшие проблемы: проектировать сложную деятельность и бизнес-процессы, формализовать свои профессиональные знания и выполнять алгоритмизацию самостоятельно, без помощи программистов — по методу “Программирование без программистов”.

Эта книга — прекрасный подарок для студентов, преподавателей, специалистов, бизнесменов и руководителей. Но не только. Она может пригодиться всем, кто хочет улучшить работу своего ума, научиться рисовать свои мысли и планы в виде наглядных и точных блок-схем и разобраться, наконец, что же такое алгоритмы и почему они играют такую важную роль в развитии цивилизации и человеческого интеллекта.

УДК 37+681.3.06+331.015.11
ББК 32.973
П18

Рецензенты:

Ю. И. Журавлев, академик РАН, зам. директора Вычислительного центра РАН, председатель Научно-методического совета по информатике Министерства образования;

П. П. Пархоменко, член-корреспондент РАН, гл. научн. сотрудник Института проблем управления РАН им. акад. В. А. Трапезникова;

Ю. В. Трунов, д-р техн. наук, профессор, Генеральный директор — Генеральный конструктор Научно-производственного центра автоматики и приборостроения им. акад. Н. А. Пилюгина, зав. Базовой кафедрой Московского института радиотехники, электроники и автоматики;

Я. В. Безель, д-р техн. наук, профессор, Генеральный конструктор Московского НИИ приборной автоматики;

В. П. Кутепов, д-р физ.-мат. наук, профессор, зав. кафедрой прикладной математики Московского энергетического института (Технического университета)

Пароджанов В. Д.

П18 Как улучшить работу ума: Алгоритмы без программистов — это очень просто! — М.: Дело, 2001. — 360 с. — Илл.: 154.

ISBN 5–7749–0211–0

В книге излагаются новые полезные для практики идеи и достижения на стыке информатики, управления и психологии. Показано, что алгоритмы, сила ума, интеллектуальный комфорт и эффективность бизнеса тесно связаны. Дается общедоступный практический курс, помогающий увеличить силу ума, ускорить разработку алгоритмов и программ, упростить формализацию профессиональных знаний, облегчить проектирование сложной деятельности и бизнес-процессов. Курс основан на “дружелюбных” графических языках, обладающих удивительной наглядностью, “заставляющих” мозг мыслить отчетливо, глубоко и продуктивно.

Для студентов, изучающих информатику, учителей и преподавателей информатики; алгоритмистов, программистов, математиков, системщиков, постановщиков задач, специалистов по *CASE*-технологиям, работающих непосредственно на фирмах; бизнесменов и руководителей, желающих эффективно контролировать потоки информации в своих организациях; психологов, изучающих работу ума, а также для широкой публики.

УДК 37+681.3.06+331.015.11
ББК 32.973

ISBN 5–7749–0211–0

ОГЛАВЛЕНИЕ

МАЛЕНЬКАЯ УВЕРТЮРА

> Чем отличается хорошее мышление от плохого?... Как улучшить мышление? Свое мышление? Мышление вообще?...
>
> Уже больше двух тысяч лет многие лучшие умы в философии, логике, психологии, педагогике пытаются найти ответы на эти вопросы. История этих усилий, блестящих идей и огромного труда, затраченного на исследования и творческое обсуждение, представляет собой яркую, драматическую картину.
>
> *Макс Вертгеймер*

Книга предназначена для всех, кто хочет упорядочить и улучшить работу своего ума. Она адресуется к работникам умственного труда всех профессий и специальностей: конструкторам и педагогам, технологам и врачам, агрономам и математикам, биологам и экономистам, психологам и нефтяникам, физикам и программистам и т. д.

У того, кто бегло пролистал ее, может сложиться ложное впечатление, что она посвящена компьютерам и программированию. На самом деле это не так. Речь идет не о думающих машинах, а о думающих людях, о загадках и особенностях человеческого познания и интеллекта. О таинственных ловушках и подводных камнях, которые подстерегают нас в трудном плавании по безбрежным морям каждодневной умственной работы. О том, как найти спасительный маяк и прибыть к цели по кратчайшему маршруту.

В книге предлагается новое универсальное средство для облегчения и улучшения работы ума, которое можно использовать во многих, практически в любых областях умственной деятельности. Само по себе это средство не имеет никакого отношения к компьютерам. Поэтому его с успехом могут применять и те, кто не любит компьютеры, относится к ним с подозрением или опаской. Вам понадобятся карандаш, бумага и больше ничего. Короче говоря, это средство вполне пригодно для улучшения самой обычной (бескомпьютерной) умственной работы.

Впрочем, любители компьютеров выиграют еще больше, поскольку указанное средство может служить основой для создания новой мощной информационной технологии, являясь частью информационных технологий следующего поколения — *когнитивных* информационных технологий.

Возможно, книга попадет в руки читателя, который хотел бы улучшить работу своего ума, но которого пугают или раздражают такие слова, как “алгоритм”, “программа”, “формализация”. Этому горю нетрудно помочь. Сейчас мы сочиним шуточный словарик, который хотя и нарушает все каноны научной строгости, зато вполне понятен новичкам.

Легкомысленный словарик

Алгоритм — точное описание решения задачи, которое ведет к победе

Алгоритм — точно описанная последовательность человеческих действий

Алгоритм — точное и полное описание работы (деятельности), которое позволяет другим людям повторить эту деятельность фотографически точно, без малейших отклонений, и получить нужный результат

Визуальный алгоритм — алгоритм, изображенный не в виде текста, а в виде наглядной картинки

Визуализация алгоритма — преобразование алгоритма, который записан в виде плохого и непонятного текста, в хорошую и понятную картинку

Когнитивный — познавательный. Это неуклюжее словечко надо запомнить, так как оно будет попадаться на каждом шагу

Программа — последовательность действий, которые человек ленится выполнять сам и поэтому поручает компьютеру или роботу

Формальный — точный

Формальное описание — точное, однозначное и полное описание, лишенное пробелов и двусмысленностей

Формализация — превращение обычного (плохого) описания в формальное (хорошее)

Автоформализация — это когда человек выполняет формализацию сам, не обращаясь к помощи друзей, родственников и случайных прохожих

Формализация деятельности — точное описание правил, по которым выполняется деятельность. В ходе формализации необходимо разбить деятельность на отдельные действия, указать последовательность их выполнения, а также условия, при которых выполняется (или не выполняется) каждое действие. В результате формализации описание деятельности превращается в алгоритм

Алгоритмизация — то же самое, что формализация деятельности

Алгоритмизация — внесение порядка в царство анархии, устранение путаницы и разгильдяйства, наведение технологической дисциплины

Алгоритмизация — процесс создания алгоритма

Эргономика — наука о том, как превратить сложную, трудную и противную работу в простую, легкую и приятную

Когнитивная эргономика — наука о том, как облегчить и улучшить умственную работу

Эргономизация науки — облегчение и улучшение научной деятельности

Эргономизация образования — облегчение и улучшение учебной деятельности

ТРЕТИЙ ГЛАЗ ДЛЯ БИЗНЕСМЕНОВ И РУКОВОДИТЕЛЕЙ

У каждого человека имеются огромные интеллектуальные ресурсы, из которых большинство людей использует лишь незначительную часть.

Давид Лассер

Предположим, вы — крупный руководитель. Например, генеральный конструктор ракетно-космической корпорации. Или даже министр. Или, скажем, глава крупного банка, в котором несколько сотен мощных компьютеров перемалывают финансовую и иную информацию. Или, предположим, вы — главный инженер большого металлургического завода, где успешно действуют самые современные системы управления технологическими процессами. Возможно, вы возглавляете нефтяную компанию, железную дорогу или центр спутникового телевидения.

Гордость вашей организации, ее интеллектуальный костяк составляют золотые умы — квалифицированные специалисты, обладающие драгоценными профессиональными знаниями; в своей работе они используют компьютеры, объединенные в локальные и иные сети. В этой книге описан **практически полезный метод, позволяющий улучшить работу ума этих людей, чтобы увеличить их интеллектуальный вклад в процветание вашей частной фирмы или государственной организации.**

Речь идет о совершенно новой идее, которая, впрочем, уже прошла тщательную проверку в ряде частных случаев и показала хорошие результаты. Предлагаемая идея тесно связана с компьютерами и автоматизацией, но в то же время существенно отличается от них, так как объектом воздействия является не компьютер, а человеческий мозг.

Здесь уместны некоторые пояснения общего характера. Когда говорят об автоматизации, имеют в виду автоматизированные системы управления предприятиями, технологическими процессами, научными исследованиями, конструкторскими разработками, проектированием, программированием, финансовой деятельностью, войсками и множество других. В XX в. широкая волна компьютеризации и автоматизации охватила весь мир и принесла замечательные плоды. Общественное богатство увеличилось. Доля физического труда сократилась, умственного — возросла. В этот период были осознаны две важные истины.

1. Современный мир — продукт мысли и ума. В конечном счете именно человеческий ум произвел все то, что мы видим и ощущаем вокруг себя. Цивилизация — это результат усилий человеческого ума.
2. В конкурентном мире успех деятельности фирм и организаций зависит от профессиональных знаний и интеллекта специалистов, от интеллектуального потенциала фирмы, от умения увеличить силу ума работников. Улучшение работы ума специалистов превращается в важнейшую, приоритетную задачу.

Здесь, однако, возникает проблема. Многие считают, что эта задача решается автоматически, сама по себе, вместе с улучшением образования и широким распространением компьютеров. Они полагают, что автоматизация умственного труда усиливает человеческий интеллект. По их мнению, чем совершеннее компьютеры, чем лучше их характеристики, тем продуктивнее работает мозг. Чем больше сложных и интеллектуальных задач возлагается на машины, тем эффективнее решает свои задачи человеческий интеллект. Чем выше степень автоматизации умственного труда, тем лучше и эффективнее действует наш ум. Согласно этой логике, чтобы улучшить работу ума специалистов, нужно увеличить мощность компьютеров и сделать их более интеллектуальными.

Можно показать, что в этих рассуждениях скрыта тонкая и коварная ошибка. На самом деле автоматизация и улучшение работы ума — хотя и связанные, но *существенно разные* вещи. Творческая продуктивность человеческого мозга не зависит от мощности и других характеристик компьютеров; она определяется совсем другими факторами.

Таким образом, налицо противоречие. С одной стороны, очевидно, что эффективность фирм и организаций зависит от силы ума специалистов; поэтому повышение творческой силы интеллекта становится задачей первостепенной важности. С другой стороны, в настоящее время, насколько нам известно, отсутствуют эффективные методы интенсификации человеческого интеллекта, которые можно применить на практике с целью увеличения реальной отдачи фирм и организаций. Более того, задача улучшения работы ума специалистов не только не решена, но фактически даже не поставлена.

Данная книга, по-видимому, представляет собой первую в мировой литературе попытку четко сформулировать проблему улучшения работы ума, показать ее огромную значимость для повышения эффективности фирм и организаций и продемонстрировать возможность ее практического решения для некоторых важных частных случаев.

ИНТЕЛЛЕКТУАЛЬНЫЙ ТЕРРОРИЗМ: ФАНТАЗИЯ ИЛИ РЕАЛЬНОСТЬ?

(Вместо предисловия)

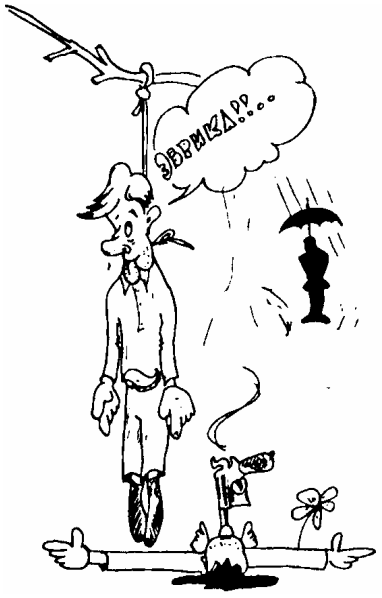
Мы просто не научились еще использовать на полную "проектную мощность" возможности нашего мозга.

Эвальд Ильенков

ПОЧЕМУ УМНЫЕ ЛЮДИ СТРАДАЮТ И ГИБНУТ?

Два ученика известного математика Давида Гильберта, изнуренные непосильной умственной работой, не получив требуемых шефом научных результатов, в отчаянии покончили с собой. Бедный старик не нашел ничего лучшего как, стоя на похоронах под проливным дождем, в течение часа произносить речь, в которой доказывал, что их диссертации могли быть исправлены.

В науке драматические ситуации, увы, не редкость. Таурин, доведенный до крайности равнодушием математиков, сжег свой труд об основах геометрии. Больяи впал в душевное расстройство. Лобачевского в одной из рецензий объявили чуть ли не сумасшедшим. Клейна постигла катастрофа — соперничая с Пуанкаре в построении теории автоморфных функций, он надорвался, тяжело заболел и вынужден был навсегда прекратить научную работу по математике. Даже великий Гаусс, несмотря на блестящие успехи и выдающиеся открытия, однажды признался: "Смерть мне милее такой жизни", причем историки предполагают, что его ипохондрия и душевный недуг — ответная реакция на неимоверно интенсивную работу и сверхчеловеческое усердие.



— Я дни и ночи бьюсь над диссертацией, а шеф опять недоволен. Лучше удавиться!

— Меня он тоже вконец замучил. Застрелюсь — и дело с концом!

РАЗВЕ ТАКАЯ ПРОБЛЕМА СУЩЕСТВУЕТ?

Анализируя подобные случаи, трудно избавиться от впечатления, что за трагедиями конкретных людей скрывается и постепенно набирает силу новое и крайне негативное социальное явление, которое иногда характеризуют как “интеллектуальный терроризм”, но которое, наверно, было бы лучше назвать интеллектуальной каторгой. В той или иной степени с ним сталкиваются все, кому приходится испытывать хроническое перенапряжение и трудиться на пределе своих возможностей. Для некоторых непосильные перегрузки начинаются уже в школе. Отчасти этому способствуют недостатки преподавания. Жан-Луи Лорьер пишет: «Существует определенный вид интеллектуального терроризма, когда некоторых учеников называют “нуль в математике”, хотя их единственная вина состоит в том, что они не понимают то, о чем... никогда не говорится» [1].

Сильнейшие умственные перегрузки испытывают многие студенты, бизнесмены, ученые и многочисленные армии интеллектуальных трудоголиков, что нередко ведет ко всевозможным расстройствам и порою — серьезным заболеваниям. Здесь есть нечто загадочное, поскольку за всеми этими внешними проявлениями скрывается неуловимая проблема-невидимка.

ИНФОРМАЦИОННЫЙ СТРЕСС — ЗЛОВЕЩИЙ СПУТНИК ИНФОРМАЦИОННОГО ОБЩЕСТВА

Будущее человечества, самое его выживание прямо зависит от роста его интеллектуальных возможностей. Однако требование всемерного развития человеческого интеллекта, максимальной интенсификации его работы во многих случаях наталкивается на жесткое препятствие, имя которому — *информационный стресс*. Именно в этой точке, как молния из искры, вспыхивает проблема интеллектуального терроризма, которой, к сожалению, часто пренебрегают, считая ее второстепенной, а то и вовсе несуществующей. Впрочем, так думают не все.

Некоторые ученые полагают, что информационный стресс возникает в ситуации информационных перегрузок, когда человек не справляется с задачей, не успевает принимать верные решения в требуемом темпе при высокой ответственности за последствия принятых решений. Анализируя тексты, решая те или другие задачи, человек перерабатывает информацию. Завершается этот процесс принятием решения. Объем перерабатываемой информации, ее сложность, необходимость часто принимать решения — все это и составляет *информационную нагрузку*. Если она превосходит возможности человека при его высокой заинтересованности в выполнении данной работы, то говорят об *информационной перегрузке*.

Стресс и вызываемые им расстройства оказывают огромное влияние на жизнь и здоровье современного человека. Стресс коварен. С одной стороны, для возникновения его вредных последствий совсем не требуется, чтобы воздействующий фактор был чрезвычайно сильным и необычным. Установлено, что обычный и заурядный фактор (такой, как дефицит времени) может оказать очень сильное стрессовое воздействие. С другой стороны, стресс может привести к общему истощению организма и даже к смерти.

КАМИКАДЗЕ УМСТВЕННОГО ТРУДА

Защита интеллектуальных работников от стресса ведется во многих направлениях: от медицинской профилактики до облегчения труда через усиление возможностей интеллекта. Вот далеко не полный перечень известных "противоядий": гигиена умственной деятельности, рациональная организация труда, повышение интеллектуальной культуры специалистов [2], стимулирование научного творчества [3], использование возможностей интуиции, совершенствование интеллектуальных способностей [4], различные теории развития интеллекта, например [5], концепция гибридного интеллекта [6] и множество частных методик, таких, как ТРИЗ (теория решения изобретательских задач) [7], динамическая техника силы ума [8] и т. д. Хотя существующие средства, теории и инструменты несомненно являются полезным и порою весьма эффективным лекарством, тем не менее, к сожалению, они не соответствуют глобальному масштабу и нарастающей значимости проблемы.

К чему это приводит? Не справляясь с неуклонным ростом сложности цивилизационных процессов, которая существенно превышает наличные интеллектуальные возможности человечества, последнее вынуждено компенсировать слабость и нехватку интеллектуальных инструментов за счет нервного перенапряжения (читай — истощения!) и увеличения длительности рабочего дня добровольных и вынужденных трудоголиков. При этом за кадром общественного внимания, телевидения и средств массовой информации остается тот факт, что интеллектуальные работники зачастую превращаются в интеллектуальных камикадзе, которых общество приносит в жертву жестокому и коварному Молоху интеллектуального прогресса.

Известный математик Герман Вейль подчеркивает: недопустимо, когда трансцендентное господствует над человеком, превращая его всего лишь в рупор интеллектуального откровения. И делает вывод: хотя наука — высокая объективная ценность, одновременно она — "ветвь человеческой деятельности, ради которой нельзя приносить в жертву самое жизнь" [9].

ЧТО ТАКОЕ ИНТЕЛЛЕКТУАЛЬНЫЙ ТЕРРОРИЗМ?

— Виновен ли профессор математики геттингенского университета Давид Гильберт в гибели своих учеников?

— Нет.

— Хотел ли он их смерти?

— Нелепый вопрос. Конечно, нет.

— В таком случае, что явилось причиной самоубийства?

Интеллектуальный терроризм — это особая социальная ситуация, когда общество, действуя возможно из лучших побуждений, формирует систему моральных норм и социальных ценностей, а также систему прямых и косвенных стимулов и с их помощью навязывает человеку такой стиль умственного труда, который почти неизбежно или с высоким риском приводит к перегрузке или другим отрицательным последствиям, наносящим ущерб физическому и душевному здоровью, снижающим качество или сокращающим продолжительность жизни. Пара-

докс в том, что интеллектуальный терроризм, даже если он влечет за собой тяжелейшие нервно-психические и иные заболевания и суицидальные попытки, в рамках существующей системы взглядов и моральных норм не рассматривается как нарушение прав человека.

По нашему мнению, интеллектуальный терроризм — это пока еще не осознанная, но вполне реальная и серьезная угроза. Источник всех этих бед и напастей состоит в том, что имеющиеся интеллектуальные средства, методы и инструменты в значительной степени устарели. Они нацелены на решение интеллектуальных задач по принципу "любой ценой", без учета реальных умственных затрат и нервно-психических последствий (когда почти полностью игнорируются тонкие когнитивно-эргономические характеристики сложной умственной деятельности), а их развитие драматическим образом отстает от насущных интеллектуальных потребностей практики. Досаднее всего, что это противоречие остается скрытым, неявным, поскольку оно пока еще не попало в сферу интересов современной науки в качестве одной из наиболее приоритетных, архиважных проблем.

ГУМАНИТАРНАЯ ПОСТАНОВКА ЗАДАЧИ

Можно ли повысить качество решений сложных и сверхсложных интеллектуальных проблем, необходимых для развития цивилизации, и одновременно защитить людей от опасных для здоровья умственных перегрузок? Как облегчить и улучшить работу человеческого ума? Увеличить продуктивность творческого мышления? Преобразовать трудные и непосильные задачи в легкие и посильные? Словом, превратить интеллектуальные муки-мученические во что-нибудь более достойное человека и даже приятное? Можно ли решить эту "сверхзадачу" хотя бы в принципе?

Анализ этих вопросов позволяет выявить проблему, которая, насколько нам известно, пока еще не обсуждалась в литературе. Суть проблемы, образно говоря, состоит в том, что современные методы интеллектуальной деятельности, пораженные вирусом интеллектуального терроризма, слишком часто превращают работников умственного труда и учащихся в пациентов, инвалидов и покойников.

Необходимо коренным образом изменить ситуацию, добиться кардинального улучшения форм и методов умственной работы, научиться решать более сложные интеллектуальные задачи с более высоким качеством за меньшее время и без ущерба для здоровья.

Интеллектуальная безопасность цивилизации — комплексное свойство глобальной интеллектуальной деятельности людей, позволяющее, во-первых, своевременно решать все более сложные интеллектуальные задачи, обеспечивающие устойчивое развитие цивилизации, во-вторых, защитить человеческий мозг от опасных и вредных для здоровья перегрузок, сводя их к минимуму или полностью исключая.

Принцип "сначала калечим, потом лечим" неэффективен ни с экономической, ни с медицинской точки зрения. Поэтому мы выдвигаем другой принцип: система "наука + образование" не должна быть вредной для здоровья.

Однако нынешняя наука не может не калечить — так уж она устроена. Почему? В частности потому, что на протяжении всей истории ее развития создатели научных теорий и методов ставили перед собой какие угодно цели и задачи, но только не задачу эффективной защиты человека от интеллектуальных перегрузок. По этой причине человеческий мозг, этот хрупкий сосуд разума, сталкиваясь с демоном науки, оказывается в крайне уязвимом положении — не выдерживая запредельной нагрузки, он получает вызванные стрессом многочисленные травмы.

Чтобы устранить вопиющее рассогласование между невообразимой сложностью науки и скромными интеллектуальными возможностями среднего человека, необходимо уяснить, что *психологическая сложность науки* не является константой — это переменная величина, которой можно управлять и уменьшать ее в желаемых (хотя и ограниченных) пределах. Для достижения цели необходимо осуществить крайне болезненную операцию — с помощью когнитивно-эргономических методов реконструировать все здание современной науки, во всех ее разделах и построениях, превратив ее из громоздкого и опасного монстра в науку с человеческим лицом — чтобы занятия наукой были эффективными, но не угрожали здоровью человека.

— Пихай-пихай! Утрамбовывай!

— А он не помрет? Слышишь, как вопит.

— Ничего. Родине нужны образованные люди.

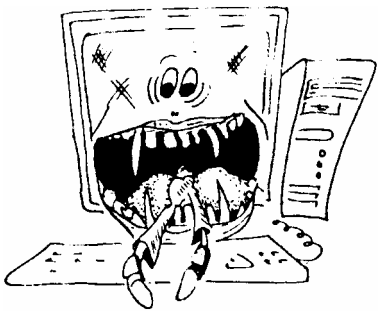
Коренная перестройка науки и образования на основе создания нового поколения интеллектуальных средств с целью ликвидации негативных проявлений интеллектуального терроризма — беспрецедентная по сложности задача. Вообще говоря, пока еще совершенно не ясно, поддается ли она решению, а если поддается, то в какой степени. Однако цель настолько важна и благородна, что стоит провести специальное исследование для более глубокого изучения проблемы.

КОМПЬЮТЕРНАЯ МИФОЛОГИЯ: ОБЛЕГЧАЮТ ЛИ КОМПЬЮТЕРЫ УМСТВЕННЫЙ ТРУД?

Чтобы избежать опасных для здоровья перегрузок, надо уменьшить интеллектуальную нагрузку на человеческий мозг. С другой стороны, развитие цивилизации приводит к усложнению интеллектуальных задач и непрерывному увеличению их количества, что предъявляет к мозгу постоянно растущие требования. Как разрешить это противоречие? Можно ли выполнить два противоположных требования — облегчить работу мозга и одновременно увеличить его умственную продуктивность?

Иногда говорят, что компьютеризация и автоматизация умственного труда снимают эту проблему. Это неверно. Использование компьютеров не приводит к уменьшению напряженности умственной деятельности, поскольку вместо одних заданий (которые удалось переложить на компьютер), человеческий мозг чаще всего получает множество новых задач, так что его суммарная нагрузка не уменьшается и даже возрастает.

Все больше исследователей приходят к выводу, что применение компьютеров во многих случаях не только не упрощает, а наоборот, резко усложняет интеллектуальные задачи, которые остаются на долю человека. Например, Эдсгер Дейкстра пишет о "неисчерпаемой" и "беспрецедентной" сложности задач, которые приходится решать программистам. Психолог М. Ярошевский отмечает: "Успехи кибернетики, все расширяющиеся перспективы передачи техническим устройствам поддающихся формализации умственных операций, которые раньше поглощали значительную часть интеллектуальных усилий ученого, резко повышают требования к формированию его способностей производить такие действия, которые не могут совершаться компьютерами" [10]. Большинство ученых признает, что информационная технология — самая сложная из всех известных технологий, а некоторые даже утверждают, что использование компьютеров приводит к усилению эксплуатации нервной энергии трудящихся и в ряде случаев "отрицательно влияет на развитие мыслительных процессов" [11].



— Он поверил, что компьютеры облегчают умственный труд!
— Ха-ха-ха!

Таким образом, массовая компьютеризация не отменяет интересующую нас проблему повышения продуктивности умственного труда, напротив — делает ее еще более актуальной.

ЧТО ТАКОЕ ИНТЕНСИФИКАЦИЯ ИНТЕЛЛЕКТА?

На наш взгляд, для решения поставленной задачи следует перейти от экстенсивной умственной деятельности к интенсивной. Поясним термины. Деятельность называется *экстенсивной*, если скорость, с которой мозг решает задачи, предполагается относительно неизменной, а выполнение сложной работы в сжатые сроки достигается за счет уплотнения рабочего времени и удлинения рабочего дня. Это означает, что человек работает на износ — по 12, 16 или 20 часов в сутки, причем перерывы для отдыха сокращаются почти до нуля ("бутерброд перехватить некогда!"). Если сотрудник, действуя в таком режиме, выполняет работу досрочно и с высоким качеством, его называют интеллектуальным героем и ставят в пример: он сделал невозможное! При этом считается хорошим тоном стыдливо умалчивать о том, насколько подобная работа приблизила нашего героя к больнице или могиле.

При *интенсивной* умственной деятельности своевременное окончание задания достигается не за счет подобных варварских методов, а за счет увеличения скорости работы мозга. ***Интенсификация интеллекта — совокупность интеллектуальных приемов и средств, изменяющих режим функционирования человеческого мозга в благоприятном направлении, чтобы использовать его возможности "на полную проектную мощность". Указанные средства специально конструируются таким образом, чтобы одновременно улучшить работу ума за счет повышения продуктивности мозга и облегчить умственный труд путем минимизации интеллектуальных затрат на единицу получаемых интеллектуальных результатов.***

Вообще говоря, эта идея не нова — на протяжении всей истории человечество безостановочно изобретало новые интеллектуальные средства, улучшающие и облегчающие работу ума. Однако делалось это в значительной степени неосознанно, отчасти вслепую и во многом стихийно. Задача состоит в том, чтобы этот процесс превратить в ясный, осознанный, целеустремленный, управляемый и, самое главное, массовый.

— Крути быстрее! Не ленись!

— Куда уж быстрей — сейчас мозги лопнут!

— Давай-давай! И запомни: повышение умственной продуктивности — задача номер один!

КРИТЕРИЙ ДЕКАРТА И ЭРГОНОМИЗАЦИЯ НАУКИ

Излагая философское учение о методе, Рене Декарт подчеркивал, что научные открытия и изобретения следует производить не путем беспорядочного блуждания наугад по дорогам науки, а с помощью метода. "Под методом же я разумею достоверные и легкие правила, строго соблюдая которые человек никогда не примет ничего ложного за истинное" и сможет добывать новое знание — все, что он способен познать — *"без излишней траты умственных сил"* [12]. Выделенные слова можно охарактеризовать как "критерий Декарта" и с современных позиций трактовать их в том смысле, что при разработке эффективных методов реализации любой умственной работы (в науке, технике, образовании и других областях) во главу угла — наряду с принципом быстрого и качественного выполнения работ — следует ставить принцип минимизации умственных усилий, т. е. минимизацию затрат нервной энергии человеческого мозга на единицу создаваемой интеллектуальной продукции[1].

Сопоставляя мысли Декарта с идеями предшественников, В. Катасонов подчеркивает: «Вдохновляясь мечтой францисканского миссионера

[1] Применительно к сфере образования эргономический критерий Декарта есть не что иное, как требование минимизации умственных усилий учащегося, затрачиваемых на единицу прочно усваиваемых знаний, умений и навыков.

XIII в. Р. Луллия о “великом искусстве”, которое могло бы автоматизировать процесс мышления, многие мыслители заняты поисками удобной знаковой системы, универсального алгоритма, позволившего бы “без излишней траты умственных сил” решить все возможные проблемы. Само создание алгебры в XVI—XVII вв. представляется даже как бы лишь побочным продуктом этой титанической “супер-идеи”» [13].

В отличие от Катасонова титанической суперидеей мы склонны считать не всеобщую математику Декарта или универсальную характеристику Лейбница, проложившие путь к созданию современной математической логики, а скорее их интеллектуальную квинтэссенцию, энергетический сгусток, мощное ядро, инициировавшее многие великие идеи и открытия, подобно тому, как во время Большого Взрыва вся наша грандиозная вселенная родилась из первичного сверхплотного ядра.

По нашему мнению, этим ядром — титанической суперидеей — является именно критерий Декарта, подлинное значение которого его последователи в должной мере так и не сумели оценить. Мы же предполагаем, что критерий Декарта — это и есть искомый архимедов рычаг, позволяющий (в сочетании с другими методами) “перевернуть” науку и образование, чтобы избавить общество от интеллектуального терроризма.

Вспомним рассуждения Декарта. Он начинает с утверждения, что в основании всех наук лежит одна и та же тождественная себе человеческая мудрость, относящаяся к разным наукам, как солнце к различным освещаемым предметам. Для познания, следовательно, было бы гораздо полезнее, чем искать “многознания” в науках, обратиться к исследованию законов самой этой мудрости. На этом пути Декарт формулирует основные положения своего метода [12].

Попытаемся перевести эти рассуждения на современный язык. Мудрость — это разум, интеллект. Изучением интеллекта, мозга и интеллектуальной деятельности занимаются когнитивная наука, психология, нейробиология, логика и как бы синтезирующая их достижения наука о человеческих факторах (эргономика). Современным эквивалентом декартовского учения о методе, исследующего и использующего законы человеческой мудрости, на наш взгляд, в какой-то мере мог бы стать научный подход к эргономизации науки и образования, опирающийся на критерий Декарта и преследующий цель максимально возможной интенсификации интеллекта. Можно предположить, что в случае успеха этого предприятия будут созданы предпосылки для проведения стратегической реформы интеллектуального труда, призванной радикально усилить мощь и могущество человеческого интеллекта.

О ЧЕМ ЭТА КНИГА?

Повторим вопрос: можно ли улучшить работу ума и одновременно облегчить деятельность мозга, чтобы интеллектуальная работа выполнялась без ущерба для здоровья? На первых порах нет необходимости решать эту сверхсложную проблему в общем виде. Вполне достаточно продемонстрировать существование эффективного решения хотя бы для некоторых значимых частных случаев. Подобный прием покажет, что проблема не безнадежна, что она в принципе поддается решению. А раз

так, появляется обоснованная надежда на то, что, действуя по аналогии, можно искать решение и для других случаев.

Таков, вкратце, общий замысел книги, который будет подробнейшим образом детализирован вплоть до самых элементарных практических случаев и многочисленных примеров, призванных наглядно показать, что создание нового поколения интеллектуальных средств, обладающих желаемыми свойствами, является хотя и трудным, но вполне осуществимым делом.

Вместе с тем мы исходим из того, что задачу интенсификации интеллекта нельзя решить традиционными средствами. Необходимо выработать и внедрить принципиально новые формы интеллектуальной деятельности в проектировании, управлении, науке, технике, образовании, медицине, экономике и других сферах. Но как это сделать на практике? Для решения задачи ниже предлагается ряд теоретических средств, опирающихся на критерий Декарта, в частности *новый когнитивный подход*. Возможность практической реализации новых идей демонстрируется на ряде примеров, среди которых подробнее всего рассмотрен язык ДРАКОН.

— Ты кто?
— ДРАКОН.
— Что это значит?
— Дружелюбный Русский Алгоритмический язык, Который Обеспечивает Наглядность.

Язык ДРАКОН — общедоступный интеллектуальный инструмент нового типа, специально сконструированный для облегчения и улучшения работы ума интеллектуальных работников и учащихся, особенно полезный при решении трудных и сверхтрудных задач систематизации и автоформализации профессиональных знаний, описания структуры человеческой деятельности и многих других задач, о которых речь впереди.

Отличие ДРАКОНА в том, что это не текстовый, а визуальный (графический) язык. Образно говоря, он прокладывает кратчайший путь к цели, взрывая логико-математические, алгоритмические и технологические скалы и препятствия динамитом наглядных картинок. Благодаря этим и другим приемам многие (хотя, разумеется, далеко не все) сложные проблемы превращаются в простые, непонятное становится понятным. В итоге достигается искомый выигрыш: производительность растет, качество улучшается, трудная работа облегчается и оказывается более приятной, умственные перегрузки резко уменьшаются, опускаясь намного ниже опасной черты.

СЕКРЕТЫ МУДРОГО ДРАКОНА: ОБЪЯСНЕНИЕ НА ПАЛЬЦАХ

Некоторые идеи, связанные с языком ДРАКОН, необычны. Их очень трудно изложить кратко, понятно и вместе с тем строго научно. Чтобы избавить читателя от утомительных длиннот и громоздких объяснений, этот параграф написан в форме забавного диалога.

Автор. Не правда ли, выполняемая вами работа очень сложна и требует больших умственных усилий? Так вот, если изобразить вашу работу на языке ДРАКОН, наблюдается следующий неожиданный эффект. Хорошо знакомая задача на глазах преображается и предстает перед вашим изумленным взором в совершенно новом свете — она резко упрощается и становится ясной, четкой и обозримой. То, что выглядело сложным и запутанным, оказывается прозрачным и очевидным. Смутное — отчетливым. Абстрактное — наглядным. А прежде скрытые ошибки видны, как на ладони.

Читатель. Но ведь чудес не бывает! За счет чего это достигается?

Автор. За счет использования более эффективных (более ДРУЖЕЛЮБНЫХ по отношению к человеку) образных средств представления профессиональных знаний, проектов и документации.

Читатель. Наверно, это очень трудно?

Автор. Как раз наоборот. Язык наглядных образов — самый легкий язык. Девиз ДРАКОНА: взглянул — и сразу стало ясно!

Читатель. Но ведь языков и так расплодилось великое множество. Зачем создавать еще один?

Автор. Пришла хозяйка в магазин: товару много, а купить нечего. В общем, языки есть, да не про нашу честь. Давайте послушаем притчу.

Притча о том, как Господь Бог языки создавал

На восьмой день Творения, когда мир уже был создан, Господь приступил к разработке формальных языков. И тут произошло нечто удивительное.

— Поскольку больше всего я люблю программистов, — заявил Всевышний, — специально для них я создал три тысячи прекрасных языков.

— А как же остальные? — удивились референты и апостолы. — Ведь им тоже нужны свои языки.

— Какие такие остальные?

— Ну все остальные, кроме программистов: физики, химики, геологи, медики, энергетики, атомщики, управленцы, экономисты, биологи, юристы всякие.

— Зачем им свои языки? Пусть пользуются языками программирования.

— Да они их не знают.

— Что значит не знают. Пускай выучат.

Наступило неловкое молчание. Наконец, апостол Павел дипломатично произнес:

— Ваше Божественное Всемогущество! Поскольку Вы сами создали все языки, для Вас выучить язык программирования — раз плюнуть. Но человек слаб.

— Это верно, он слаб, — подтвердил Господь.

— Поэтому для среднего работника умственного труда (не программиста), у которого своих забот выше крыши, разобраться в тонкостях программирования довольно трудно.

— Трудности можно преодолеть.

— Можно-то оно можно. Так ведь душа не лежит, потому как — противно, а главное — зачем? Нельзя же насильно заставлять человека учить то, что ему не нужно для работы. Для большинства людей язык программирования — это "собачий" язык, а написанные на нем программы — странная окрошка из египетских иероглифов. Они непонятны никому, кроме горстки их создателей.

— Что вы такое говорите! — возмутился Господь. — Сразу видно, что вы отстали от жизни. Академик Ершов учит, что "программирование — вторая грамотность". Нынче даже школьники программы освоили. А студенты их, как орехи, щелкают. Запомните: программирование должны знать все! Это и будет общий язык для взаимопонимания между специалистами. И никаких других языков не нужно. Все. Совещание окончено. Выполняйте!

Однако, как это часто бывает, с реализацией руководящих указаний по неизвестным причинам возникла небольшая заминка. Или, наоборот, большая. Потому что лозунг "программирование — вторая грамотность", подразумевающий чуть ли не поголовное умение программировать, воплотить в жизнь до сих пор не удалось. Практика показывает, что умеющие программировать составляют лишь около 10% от общей численности работников умственного труда. Поэтому сегодня в сообществе интеллектуальных работников образовался значительный языковый дисбаланс. Он заключается в том, что меньшинство (10% программистов) владеет огромным языковым богатством, включающим 3000 языков программирования. А подавляющее большинство (90% специалистов) кроме языка математики не имеют в своем распоряжении никакого другого широко распространенного и универсального формального средства.

Читатель. Так, может, этим специалистам и не нужны никакие языки?

Автор. Это не так. Язык — интеллектуальное оружие специалиста. Чем лучше язык, тем лучше работает мозг, тем выше производительность умственного труда.

Читатель. Как же быть?

Автор. Прежде всего следует признать, что при выборе генерального направления разработки искусственных языков допущена стратегическая ошибка. Нынешняя ситуация, когда 90% специалистов не имеют языка, пригодного для быстрого и эффективного решения своих задач, является ненормальной и неприемлемой.

Читатель. Где же выход?

Автор. Нужно устранить диспропорцию в обеспечении специалистов языковыми средствами. Поскольку меньшинство (т. е. программисты) уже располагает достаточным числом высокоэффективных языков, настало время подумать об остальных. Сегодня необходимо создать не очередной язык для меньшинства, а язык для всех, который позволит укрепить слабое звено и улучшить умственную продуктивность большинства специалистов. Для этого нужно построить формальный или частично формальный язык, который был бы не "собачьим" или "птичьим", а "человечьим" — общедоступным, удобным и понятным для каждого. Язык ДРАКОН как раз и призван хотя бы отчасти заделать эту зияющую брешь. ДРАКОН — это язык не для элиты, а для широких масс, которые категорически не приемлют "птичьи" языки программирования.

Смена терминов или изменение концепции?

Читатель. Стало быть, ДРАКОН — это не язык программирования, а что-то новенькое. Как же прикажете его величать?

Автор. Назвать можно как угодно. Например, "технологический язык", сокращенно "техноязык".

Читатель. Все-таки непонятно: зачем менять устоявшуюся терминологию, к которой все привыкли? Чем вам не нравится название "язык программирования"?

Автор. Речь не о смене терминов, а о коренном изменении концепции. Давайте начнем от печки. Мы говорим об ученых, врачах, технологах, педагогах и других работниках умственного труда. О тех, кто не программировал, не программирует и не собирается программировать. О тех, кому по характеру работы это просто не нужно. Потому что их работа заключается совсем в другом. Так вот, цель состоит в том, чтобы создать для этих людей новый язык, который помог бы им решать те задачи, которые они сегодня решают, но делать это более быстро и эффективно. Таким образом, речь идет не о программировании, а совсем о других видах деятельности. Поэтому название "язык программирования" здесь просто неуместно.

Читатель. Все равно непонятно.

Автор. Рассмотрим пример. Химик написал формулу

$$HCl + NaOH = NaCl + H_2O$$

Какой язык здесь использован? Ясно, что это не язык программирования, а язык химических формул. Последний является "родным"

языком химиков и помогает им успешно справляться со своими проблемами. Правда, этот язык не общий, а частный: он позволяет решать не все задачи, волнующие химиков, а только некоторые. А за рамками химии он вообще почти никому не интересен. В отличие от него техноязык — это универсальный язык, пригодный для широкого класса задач практически в любых областях человеческой деятельности.

Самая сложная вещь на свете

Читатель. Что значит "в любых областях деятельности"? Что общего между деятельностью врача и конструктора, финансиста и агронома, металлурга и микробиолога?

Автор. Общее то, что все они работают, т. е. занимаются деятельностью. Человеческая деятельность — самая сложная вещь на свете.

Читатель. Что в ней такого уж сложного?

Автор. Деятельность состоит из действий, а последние зависят от условий. При данном условии я выполняю одни действия, при другом — другие. Если работа сложная, приходится учитывать сотни и тысячи условий, которые образуют невообразимое число сочетаний. И для каждого сочетания порою нужно делать совершенно разные цепочки операций.

Иногда работник действует интуитивно, "наощупь", по обстоятельствам. Некоторые операции человек выполняет сам, другие поручает различным механизмам, роботам, компьютерам.

Проблема в том, что до сих пор отсутствует эффективный язык, позволяющий дать целостное и точное описание деятельности во всем ее красочном многообразии, богатстве и многосложности, выявить ее правила и структуру, учесть тончайшие отличия и особенности разных профессий (а их — тысячи), устранить путаницу и неразбериху, навести порядок, систематизировать знания о деятельности и представить их в наглядной и удобной форме. Нынешние многотомные руководства, содержащие описание деятельности, слишком трудны — мозги сломаешь, пока поймешь. К тому же они неполны — многие знания о деятельности нигде не записаны и хранятся только в головах людей. Вытащить их оттуда — сложнейшая задача. Отсутствие удобного языка для описания структуры деятельности сильно затрудняет обучение. Многие важные сведения вообще не зафиксированы в документах и передаются как эпос по принципу "из уст в уста".

Язык ДРАКОН призван ослабить или устранить эти недостатки, чтобы хоть как-то ограничить вакханалию путаницы и хаоса. Цель ДРАКОНА — внести порядок в царство анархии, установить четкие стандарты в области, где их никогда не было, положить конец цыганской вольнице и неуемному разгильдяйству, постричь всех (кто согласится) под одну гребенку, за счет этого значительно повысить производительность труда и получить ощутимый экономический эффект. Заметьте, ДРАКОН предоставляет стандартные средства описания деятельности независимо от того, кто выполняет действия: сам человек или созданные им машины, роботы, компьютеры.

Кстати, программирование — это тоже деятельность. Поэтому техноязык можно использовать как язык программирования (обратное неверно).

Читатель. Ага, так значит ДРАКОН — это все-таки язык программирования!

Автор. Послушайте, вы, по-моему, нарочно хотите поссорить меня с теми, ради кого написана эта книга. Надо же учитывать человеческую психологию! Если я скажу, что ДРАКОН — язык программирования, немалая часть потенциальных читателей тут же отшвырнет ее со словами: "Это для программистов, мне это не нужно!" Их можно понять, потому что сам термин "язык программирования" для многих уже давно превратился в красную тряпку, в ненавистное пугало.

Читатель. А я подозреваю, что вы сознательно пытаетесь обмануть людей, подсунув им старый товар в новой упаковке.

Зачем ДРАКОНУ две головы?

Автор. Никакого обмана нет и в помине. Просто язык ДРАКОН выполняет две принципиально разные функции. Для большинства работников он является новым средством повышения эффективности интеллектуального труда, причем у этого средства практически нет аналогов в мировой практике. В этом качестве ДРАКОН не имеет ни малейшего отношения к программированию. Поэтому тем глубокоуважаемым людям, которые не любят или даже ненавидят программирование, можно со всей откровенностью сказать:

> Вы правы. Язык программирования — ваш враг.
> Но ДРАКОН — не язык программирования.
> ДРАКОН — ваш друг

Вторая функция состоит в том, что для программистов ДРАКОН действительно является языком программирования. Таким образом, ДРАКОН имеет две головы, обращенные к совершенно разным аудиториям. Причем каждая голова пытается угадать сокровенные потребности своей аудитории и по возможности удовлетворить их наилучшим образом.

Читатель. Стало быть, вы хотите угодить и нашим, и вашим?

Автор. Вот именно. В этом состоит одно из ключевых преимуществ, поскольку язык ДРАКОН можно использовать как удобный "мост взаимопонимания" между непрограммирующим большинством и программирующим меньшинством, между "бескомпьютерной" и компьютерной интеллектуальной деятельностью.

СПРАВКА О СОСТОЯНИИ ДЕЛ

Язык ДРАКОН разработан совместными усилиями Российского космического агентства (НПЦ автоматики и приборостроения, г. Москва) и Российской академии наук (Институт прикладной математики им. М.В. Келдыша, г. Москва) как обобщение опыта работ по созданию космического корабля "Буран". На базе ДРАКОНА построена автоматизированная технология проектирования программных систем (*CASE*-технология) под названием "ГРАФИТ-ФЛОКС". Она успешно используется в ряде крупных космических проектов: "Морской старт", "Фрегат", "Протон-М" и др.

ДРАКОН — очень легкий язык. Настолько легкий, что разработку многих компьютерных программ для космических ракет на практике ведут не программисты, а обычные специалисты — по принципу “программирование без программистов”. Причина отказа от программистов проста. При решении практических прикладных задач специалисты досконально владеют материалом и прекрасно знают постановку задачи. В отличие от них программисты не знают “физику процесса” и становятся “лишними людьми”, без которых вполне можно обойтись. Это позволяет значительно сократить издержки, улучшить показатель “затраты—результат”, ускорить ход работ и полностью избавиться от ошибок “испорченного телефона”, вызванных взаимным непониманием между ПРОГРАММИСТАМИ и СПЕЦИАЛИСТАМИ.

ДРАКОН универсален. Он может применяться для наглядного представления и быстрой разработки алгоритмов не только в “космосе”, но и в “земных” видах человеческой деятельности. Практическая полезность ДРАКОНА получила высокую оценку. Министерство образования включило изучение языка ДРАКОН в программу курса информатики высшей школы (см.: Примерная программа дисциплины “Информатика”. Издание официальное. — М.: Госкомвуз, 1996. С. 3, 4, 15, 16).

Ведется подготовка учебных книг для средней и высшей школы. Уже издана первая из них — игровое учебное пособие для детей младшего и среднего школьного возраста:

Пароджанов В. Д. Занимательная информатика: Волшебный Дракон в гостях у Мурзика. М.: Росмэн, 1998, 2000. 160с. 200 иллюстраций.

Глава 1
НА ПОДСТУПАХ К НОВОМУ ЯЗЫКУ

> В некоторых разделах своей книги я вышел за рамки теорий, в которых я могу претендовать на какие-либо профессиональные знания. Прошу тех, в чьи заповедные угодья я вторгся, простить мне мою опрометчивость. И если отдельные трофеи, о которых я пишу, существуют только в моем воображении, то, по крайней мере, такое браконьерство не причиняет никакого ущерба законным владельцам, тогда как случайный пришелец может порой увидеть то, что является одновременно и неожиданным, и реальным.
>
> *Джордж Паджет Томсон*

ЗАЧЕМ НУЖЕН ЯЗЫК ДРАКОН?

ДРАКОН — это алгоритмический язык, обладающий необычным свойством: одновременно он является языком для описания структуры деятельности, языком понимания и взаимопонимания, языком развития интеллекта. Как язык программирования, он удовлетворяет требованиям математической строгости, позволяющим из исходного текста однозначно получать объектный код (машинный код для компьютера). Но главное не в этом. При создании ДРАКОНА основное внимание уделялось человеческому фактору, улучшению наглядности и доходчивости технических и социальных проектов и технологий, повышению эргономических характеристик алгоритмов, чтобы не на словах, а на деле превратить ДРАКОН в язык для улучшения работы ума, язык понимания и взаимопонимания.

Хотя ДРАКОН внешне очень напоминает обычные блок-схемы алгоритмов и программ, фактически он является оригинальной разработкой. Наиболее близким функциональным аналогом ДРАКОНА следует считать схемы действий *(action diagrams)* [1–4] и схемы деятельности *(activity diagrams)* [5].

Для дотошных читателей, которые любят подробности, аналогами ДРАКОНА — в той или иной степени — можно назвать и более дальних “родственников”. К их числу относятся: диаграммы Несси—Шнейдермана [6], *HOS*-схемы [1], схемы “гринпринт” [7], *SPD*-диаграммы фирмы *NEC* [8], *PAD*-схемы фирмы Хитачи [8], деревья и таблицы решений, схемы декомпозиции [4], схемы зависимости [1], язык *SDL* и его производные, систему *BLS*, созданную А. Смоляниновым из Санкт-Петербургского электротехнического университета, *R*-схемы И. Вельбицкого, π-схемы В. Прохорова и т. д.

В ЧЕМ СЕКРЕТ ДРАКОНА? — В КОГНИТИВНОМ ПОДХОДЕ

Впрочем, сравнение с аналогами в данном случае малопродуктивно, так как оно не позволяет раскрыть наиболее существенную особенность ДРАКОНА, которая называется “когнитивный подход” [9]. Термин “когнитивный” (познавательный) пока еще не получил широкого распространения среди проектировщиков, разработчиков, инженеров и программистов, однако он является тайным паролем нового могущественного научного ордена, вернее сказать, знаменем двух новых, бурно развивающихся направлений в психологии и науке об интеллекте, известных как когнитивная психология и когнитивная наука[1].

Одна из целей этих дисциплин заключается в том, чтобы выявить скрытые резервы человеческого мозга, повысить творческую продуктивность интеллектуальных работников.

Суть вопроса состоит в следующем. Разработчики технических и социальных проектов, интеллектуальные работники — это живые люди, обладающие мозгом, возможности которого, хотя и велики, но тем не менее далеко не безграничны. Таким образом, проблема проектирования — это не только техническая, но и человеческая, познавательная, т. е. когнитивная проблема.

Под *когнитивным фактором* в данной книге понимаются познавательные, интеллектуальные, мыслительные, творческие аспекты деятельности ученых, специалистов и учащихся. Чем сложнее объект технического и социального проектирования, тем важнее делать акцент на необходимости тщательного учета когнитивных характеристик деятельности людей. Академик П. Симонов подчеркивает: для разработчиков систем “чрезвычайно важно знание правил, следуя которым живой мозг воспринимает, обрабатывает, фиксирует и использует вновь полученную информацию. Сведения о таких правилах, выявленных в эксперименте, поставляет когнитивная психология” [11].

Использование названных правил позволяет получить практический результат — повысить производительность умственного труда.

ПОЧЕМУ ЛЮДИ НЕ ИНТЕРЕСУЮТСЯ СОБСТВЕННЫМ МОЗГОМ?

В последние два десятилетия в нейробиологических и психологических исследованиях были получены новые и чрезвычайно важные сведения о работе мозга. Они открывают путь к революционным преобразованиям интеллектуального труда, создавая предпосылки для кардинального повышения его знаниепорождающей творческой продуктивности. Фактически мы находимся на пороге стратегической реформы интеллектуального труда, обещающей включение в созидательную работу новых мощных резервов человеческого мозга и интеллекта.

---

[1] *Когнитивная психология* (психология познавательных процессов) уподобляет мозг компьютеру, исследует переработку информации человеком и рассматривает познание как “совокупность процессов переработки информации” [10]. *Когнитивная наука* (наука об интеллекте) — это более широкое понятие, представляющее собой сплав когнитивной психологии, психофизики, кибернетики, нейробиологии, лингвистики, математической логики и ряда других отраслей знания.

Но эти результаты в силу известных междисциплинарных барьеров пока еще не стали достоянием проектировщиков, инженеров и программистов, разрабатывающих сложные технические и социальные системы. В итоге создалось парадоксальное положение. Поясним ситуацию на примере.

Программированием занимаются люди, обладающие мозгом. Однако до сих пор языки, методы и теории программирования строились без учета конструкции мозга. Невозможно максимизировать творческую продуктивность мозга программистов, не учитывая его конструкцию. Следовательно, традиционные способы создания языков и технологий программирования, игнорирующие конструкцию мозга, являются устаревшими и неэффективными.

— Отличный мозг! Но в чем дело? Почему он так медленно работает?

Думается, этот вывод справедлив и в других случаях. Игнорирование закономерностей работы мозга, недостаточное внимание к когнитивным вопросам приводит к неприятным последствиям: взаимному непониманию между соавторами сложных проектов, серьезным заблуждениям в научном познании, крупным научно-техническим просчетам, устранение которых требует значительных материальных издержек (связанных с дорогостоящими конструкторскими доработками и трудоемкими переделками программного обеспечения), а также к ощутимому снижению результирующей производительности труда разработчиков и других участников технических и социальных проектов.

Наука о человеческих факторах называется эргономикой. Когнитивные проблемы — важная часть эргономики. Чтобы вычленить когнитивную группу среди других эргономических вопросов, иногда употребляют термины “когнитивная эргономика” и “когнитивно-эргономические проблемы”.

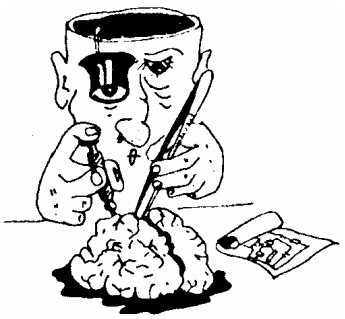
СТАНЕТ ЛИ ДРАКОН ЧЕМПИОНОМ МИРА ПО КРИТЕРИЮ “ПОНИМАЕМОСТЬ АЛГОРИТМОВ”?

Данная книга имеет сугубо практический характер. Ниже будет показано, что когнитивный подход — это рабочий метод, дающий полезные плоды: улучшение понимаемости алгоритмов и программ, проектов и технологий, повышение производительности сложного интеллектуального труда. Мы постараемся обосновать этот тезис, постепенно раскрывая особенности языка ДРАКОН.

Как и все прочие языки, ДРАКОН опирается на математику и логику. Однако сверх того, он самым тщательным образом учитывает когнитивные вопросы. Благодаря систематическому использованию когнитивно-эргономических методов ДРАКОН приобрел уникальные эргономические характеристики. Можно предположить, что в будущем ДРАКОН сможет претендовать на звание чемпиона по критерию “понимаемость алгоритмов и программ” (в классе императивных языков)[1].

ДРАКОН можно определить как общедоступный визуальный язык, предназначенный для описания структуры деятельности, для систематизации, структуризации, наглядного представления и формализации императивных знаний, а также для проектирования, программирования, моделирования и обучения. Это универсальный межотраслевой язык делового мира, служащий для описания научно-технических, медицинских, биологических, экономических, социальных, учебных и иных задач. ДРАКОН позволяет упорядочить и представить решение любой, сколь угодно сложной императивной (процедурной, деятельностной, технологической, рецептурной, алгоритмической) проблемы в виде наглядных чертежей, выполненных по принципу “взглянул — и сразу понял!”

Человечность языка ДРАКОН, стремление создать максимальный комфорт для работы человеческого мозга, всемерная забота о повышении творческой продуктивности персонала позволяет надеяться, что ДРАКОН получит самое широкое применение в народном хозяйстве, бизнесе, обороне, науке и системе образования. Используя не просто наглядные, а предельно наглядные формы представления знаний, облегчая работу мозга, ДРАКОН обеспечивает заметный рост производительности интеллектуального труда.



— Кто сказал, что мозг нельзя улучшить?
— А зачем его улучшать?
— Чтоб быстрее работал!

В основе языка ДРАКОН лежит идея *когнитивной формализации знаний*, позволяющая сочетать строгость логико-математической формализации с точным учетом когнитивных (познавательных) характеристик человека [9]. В результате удалось кардинальным образом упро-

---

[1] Императивный язык — это язык, который описывает работы и процессы, состоящие из действий, а также условия выполнения каждого действия.

стить и облегчить процедуру описания структуры деятельности, формализацию профессиональных знаний специалистов, стандартизовать ее и сделать пригодной для массового практического использования. Это в равной степени касается как компьютерной, так и "бескомпьютерной" интеллектуальной деятельности людей.

Таким образом, основная цель создания языка ДРАКОН — обеспечить качественный скачок в повышении продуктивности сложного интеллектуального труда за счет увеличения интеллектуальной производительности человеческого мозга, выявления и более полного использования резервов человеческого интеллекта, создания когнитивных предпосылок для существенного повышения эффективности информационных технологий.

НА КОГО РАССЧИТАН ЯЗЫК ДРАКОН?

Язык в равной степени рассчитан на четыре категории лиц:

- на людей, совершенно не знакомых (или слабо знакомых) с программированием и вычислительной техникой: механиков, электриков, комплексников, прибористов, испытателей, физиков, химиков, геологов, биологов, медиков, агрономов, экономистов, юристов, психологов и т. д.;
- на профессиональных программистов, математиков и разработчиков вычислительной техники, в том числе на специалистов по операционным системам, системному и прикладному программированию, а также микропрограммированию (для персональных, универсальных, управляющих и бортовых компьютеров);
- на школьников и студентов;
- на руководителей многих уровней, желающих за минимальное время уяснить суть сложных проблем.

ПЕРЕЧЕНЬ ЗАДАЧ, РЕШАЕМЫХ С ПОМОЩЬЮ ЯЗЫКА ДРАКОН

Язык ДРАКОН может быть использован при решении следующих задач:

- описание структуры человеческой деятельности;
- наглядное представление императивных знаний в любых областях народного хозяйства, науки и образования;
- описание концептуальных решений и императивных моделей;
- проектирование алгоритмов и программ;
- разработка алгоритмов и программ;
- проектирование технологических процессов;
- описание любых технологий (промышленных, сельскохозяйственных, медицинских, педагогических, управленческих и т. д.);
- описание процесса проектирования;
- описание процессов функционирования дискретных систем и приборов, в том числе интеллектуальных систем;

- описание исходных данных на разработку систем автоматизированного проектирования и систем автоматизации научных исследований;
- описание процесса решения математических задач;
- описание диалога и взаимодействия человека-оператора и машины (пульта управления);
- описание процесса проверки и поиска неисправностей;
- решение задач диагностики в любых предметных областях;
- разработка микропрограмм;
- описание процесса функционирования организаций и предприятий;
- автоформализация профессиональных знаний ученых, конструкторов, математиков, врачей, юристов, агрономов, психологов, операторов и т. д.;
- решение учебных задач: обучение навыкам алгоритмизации, программирования и автоформализации знаний в предельно сжатые сроки.

Как уже говорилось, функциональным аналогом ДРАКОНА являются схемы действий и схемы деятельности. ДРАКОН способен выполнять все функции последних (обратное неверно). Поэтому перечень можно продолжить, включив в него задачи, решаемые схемами действий. Это позволит описать некоторые функции ДРАКОНА с помощью терминов, характерных для американской литературы:

- стратегический обзор функций корпораций *(strategic overview of corporate functions)*;
- описание логических отношений между процессами *(logical relationship among processes)*;
- описание укрупненной структуры программ *(overall program structure)*;
- описание детальной логики программ *(detailed program logic)* [1];
- полная декомпозиция программ *(ultimate decomposition)*, начиная от укрупненной логики и кончая деталями кода, что в равной мере полезно при проектировании как сверху вниз *(top-down design)*, так и снизу вверх *(bottom-up design)* [4];
- проектирование программ до последнего момента может вестись независимо от языка и лишь на последнем этапе осуществляется переход к нужному языку [1];
- обучение конечных пользователей, стимулирующее их анализировать и проектировать детальную логику процессов *(detailed process logic)* [1];
- описание процедур организационного управления *(management procedures)* [4];
- описание компьютерных методологий *(computer methodologies)* [4];
- описание методологий информационной техники *(methodologies of information engineering)* [4].

Как видно из этого списка, ДРАКОН обладает свойством универсальности, оказываясь полезным при решении широкого круга разноплановых задач. Благодаря этому ДРАКОН выполняет функцию универсального языка делового общения и взаимопонимания для специалистов различных специальностей. Кроме того, ДРАКОН существенно

облегчает процесс формализации знаний, открывая новые возможности для повышения уровня автоматизации при проектировании и эксплуатации сложных объектов.

ВЫВОДЫ

1. Традиционные цели и методы создания искусственных языков, в частности языков программирования, следует признать во многом устаревшими.
2. Последние исследования в области нейробиологии, психологии, когнитивной науки и эргономики позволили получить новые и чрезвычайно ценные сведения о работе мозга, которые можно и нужно использовать при разработке нового поколения языков с целью повышения продуктивности человеческого мозга.
3. В настоящее время отсутствует продуманная стратегия, направленная на устранение междисциплинарных барьеров, преследующая цель вооружить разработчиков искусственных языков нового поколения глубокими знаниями в области наук о человеке, человеческом факторе и человеческом интеллекте. Этот недостаток нужно как можно скорее устранить.
4. Концепция искусственных языков нового поколения опирается на междисциплинарный подход и коренным образом изменяет традиционные представления о назначении искусственных языков и наборе приоритетных требований к ним. Во главу угла ставятся гуманитарные вопросы и требования, которые должны быть соответствующим образом детализированы.

Глава 2

МОЖНО ЛИ СОЗДАТЬ ЯЗЫК, УЛУЧШАЮЩИЙ ПОНИМАНИЕ И ВЗАИМОПОНИМАНИЕ?

— ...Скажите, отчего разбрелись все ученые в разные стороны и каждый говорит языком, которого другой не понимает? Отчего мы все изучили, все описали и почти ничего не знаем?

— Извините, это не мой предмет, я только собираю факты — я статистик.

Владимир Одоевский

ПОЧЕМУ СПЕЦИАЛИСТЫ НЕ ПОНИМАЮТ ДРУГ ДРУГА?

В 1880 г. баварский ксендз Иоган Шлейер, стремясь улучшить взаимопонимание между людьми, придумал язык "воляпюк" (искаж. от *world speak*, что значит "всемирный язык"). Чуть позже варшавский врач Земенгоф изобрел эсперанто. Хотя эти проекты всемирных языков не оправдали надежд, однако они сыграли положительную роль, ибо приковали внимание к назревающей проблеме — созданию искусственных языков.

Сегодня, когда число искусственных языков программирования перевалило за три тысячи, проблема взаимопонимания между людьми почти так же далека от решения, как и во времена Шлейера и Земенгофа. Да, действительно, языки Бейсик, Паскаль, Си, Си++, Ява и многие другие давно стали всемирными языками. Однако популярность языков вовсе не говорит о том, что написанные на них программы понятны всем, кому это нужно. Многие программисты жалуются, что свою собственную программу они с трудом понимают через полгода, а то и через месяц. А если речь идет о чужой программе? Тогда становится совсем тяжко. Нередко бывает легче написать свою программу, нежели разобраться в том, что делает чужая. Поэтому среди требований, предъявляемых к современным алгоритмическим языкам, на первое место все чаще выходит *понимаемость программ (comprehensibility)*, которая определяется как свойство программы минимизировать интеллектуальные усилия, необходимые для ее понимания.

Постепенно стало ясно, что улучшение понимаемости проектов, технологий, алгоритмов и программ — исключительно сложная проблема, чем-то напоминающая проблему общения и взаимопонимания ученых и специалистов. Как известно, информационный взрыв, усложнение решаемых задач и связанная с этим специализация приводят к опасной тенденции. По словам академика Н. Моисеева, "ученые начинают все хуже и хуже понимать друг друга". В особенности это касается взаимодействия работников из разных отраслей науки и техники, что создает значительные трудности для общения исследователей, занятых решением межотраслевых комплексных проблем.

ЯЗЫК ДРАКОН КАК “ЭСПЕРАНТО” ДЕЛОВОГО МИРА

А нельзя ли взмахнуть волшебной палочкой и, используя обширный опыт создания языков программирования, придумать всемирный язык принципиально нового типа — образно говоря, эсперанто делового мира, облегчающий взаимопонимание специалистов разных профессий? Трудность в том, что подавляющее большинство специалистов использует для общения не языки программирования, а совсем другие средства.

В самом деле, на каком языке разговаривают и решают свои профессиональные проблемы специалисты народного хозяйства и социальной сферы? Какой язык является для них “родным”, привычным, “свойским”? Ответ известен. Это естественный человеческий язык, включающий научные понятия и термины, математические и иные формулы, а также графики, чертежи, диаграммы, карты, схемы и т. д. Неприятность в том, что этот язык слабо формализован (допускает двусмысленности, пробелы, неточности) и к тому же не унифицирован: разные специалисты фактически используют разные профессиональные языки.

Задача формализации и унификации множества профессиональных языков с целью обеспечить эффективное взаимопонимание между специалистами любых профессий, включая программистов, является, хотя и важной, но, увы, неразрешимой. Положение в корне меняется, если ограничиться императивными профессиональными знаниями. Именно эту задачу решает язык ДРАКОН. Он построен путем формализации, неклассической структуризации и эргономизации блок-схем алгоритмов и программ, описанных в стандартах ГОСТ 19.701–90 и *ISO* 5807–85.

ЧТО ТАКОЕ ИНТЕЛЛЕКТУАЛЬНОЕ ВЗАИМОПОНИМАНИЕ?

При разработке сложных проектов (таких, как космический корабль “Буран”, атомная электростанция или химический завод), при проведении сложных исследований (например, в теоретической физике, биологии, медицине), при решении других, более простых задач (например, при программировании) нередко возникает вопрос: как добиться взаимопонимания между соисполнителями работ? Трудность в том, что каждый исследователь и разработчик, каждый участник общего дела хорошо знает лишь свой собственный, относительно небольшой (по размерам), хотя и весьма сложный (по глубине идей) участок работы и довольно смутно представляет, что творится у соседей. Отсюда взаимные недоразумения, неувязки и ошибки на стыках. Логично спросить: в чем причина неприятностей? Не слишком ли большие ресурсы (людские, материальные, финансовые и временны́е) приходится затрачивать для обеспечения эффективного взаимодействия между специалистами, участвующими в совместной работе? Почему крупные исследования и разработки нередко затягиваются на месяцы, а то и на годы?

С такими или примерно такими проблемами мы столкнулись при построении орбитального корабля “Буран”. Опыт Бурана показывает, что вопрос об *интеллектуальном взаимопонимании* специалистов зачастую

играет ключевую, основополагающую роль и во многом определяет успех дела. При создании сложнейшего комплекса бортовых и наземных программ "Бурана" приходится расплетать хитроумный клубок донельзя запутанных проблем. Поэтому в бой вступает целая армия специалистов разных профессий из множества разных организаций, которые на начальном этапе работ очень плохо понимают друг друга. Это именно тот случай, когда запредельная сложность проблемы и связанная с нею узкая специализация приводят к смешному, но, увы, реальному парадоксу, когда "специалисты по клизме" не понимают "специалистов по наконечнику".

Тем не менее создаваемые ими алгоритмы, больше напоминающие первозданный алгоритмический хаос, раздираемый молниями вопиющих неувязок, в конечном итоге должны превратиться в единый филигранный узор, управляющий "Бураном" с баснословной точностью и надежностью.

Таким образом, проблема стара, как мир: чтобы избежать печальной участи строителей вавилонской башни, участники сложного проекта должны научиться очень хорошо понимать друг друга. В противном случае многочисленные ошибки "на стыках" помешают успеху разработки.

В ЧЕМ ОСОБЕННОСТЬ ДРАКОНА?

Недостаток традиционного подхода состоит в том, что создатели языков и компьютерных систем, как подчеркивает психолог Дональд Норман, "слишком часто начинают с машины, а о человеке думают только в конце, когда уже поздно". Чтобы избежать подобных ошибок, в ходе разработки языка ДРАКОН был выбран совершенно иной подход. Была объявлена стратегическая цель: создать наиболее комфортные условия для работы человеческого интеллекта, обеспечить наилучшие возможности для повышения эффективности коллективного разума специалистов.

В соответствии с этой программной установкой была поставлена задача: создать общедоступный, предельно легкий в изучении и удобный в работе язык, позволяющий решать проблемы ценою минимальных интеллектуальных усилий. Язык, который в силу своей изначальной ориентации на человека мог бы стать подлинно "народным", т. е. "родным" для специалистов практически любого профиля (а не только программистов).

ВЫВОДЫ

При создании языка ДРАКОН были выдвинуты следующие гуманитарные требования.

1. Улучшить работу человеческого ума.
2. Предложить эффективные средства для описания структуры деятельности.
3. Предоставить человеку такие языковые средства, которые резко упрощают восприятие сложных императивных проблем и общение с кол-

легами, делают непонятное понятным и за счет этого буквально заставляют человека мыслить отчетливо, глубоко и продуктивно. В этих условиях вероятность заблуждений, просчетов и ошибок неизбежно падает, а производительность растет.

4. Радикально облегчить межотраслевое и междисциплинарное общение между представителями разных организаций, ведомств, отделов, лабораторий, научных школ и профессий.

5. Устранить или уменьшить барьеры взаимного непонимания между работниками различных специальностей (врачами и физиками, математиками и конструкторами, биологами и экономистами и т. д.), а также программистами и теми, у кого аллергия к любому программированию.

6. За счет использования когнитивно-эргономического подхода к проектированию синтаксиса и семантики добиться кардинального улучшения качества программного обеспечения по критерию “понимаемость программ”.

Глава 3

СООБРАЖЕНИЯ, ПОВЛИЯВШИЕ НА СОЗДАНИЕ ЯЗЫКА ДРАКОН

> Вся литература, посвященная компьютерам... преподносит нам эйфорические сказки о великих победах, число же проанализированных позорных пятен весьма незначительно.
>
> *Поль Страссман*

ЧТО ВАЖНЕЕ: КОМПЬЮТЕРЫ ИЛИ ЧЕЛОВЕЧЕСКИЙ МОЗГ?

Интеллектуальную основу современной цивилизации составляет коллективная интеллектуальная деятельность миллионов людей, занятых в различных организациях, создающих и потребляющих информационный продукт. Однако в эпоху массовой компьютеризации многие организации сталкиваются с новой, во многом неожиданной проблемой. По мере роста мощности и снижения удельной стоимости компьютеров все более негативное влияние на сроки и стоимость выполнения интеллектуальных работ оказывает недостаточная производительность *самих* интеллектуальных работников. Возникает парадоксальная ситуация: интеллектуальный работник превращается в самое слабое (а нередко и самое дорогое) звено автоматизированной технологии решения многих научных, практических и иных задач. Мировой опыт показывает, что большинство организаций расходует слишком много средств на информационный продукт, а получает незначительную отдачу из-за низкой творческой производительности персонала. В этих условиях требование повысить производительность именно интеллектуального работника (т. е. человека, а не машины) становится все более актуальным.

Еще одна неожиданность состоит в том, что традиционное понятие “производительность труда”, родившееся в недрах материального производства, где оно служит вполне законным и заслуживающим доверия учетным инструментом, будучи перенесенным в сферу информационного производства для оценки труда интеллектуальных работников, нередко теряет прежнюю однозначность, становится плохо определенным и дезориентирующим.

Чтобы избежать неприятностей, необходимо четко разграничить три понятия:

- интегральную производительность системы “персонал — компьютеры”;
- производительность компьютеров;
- производительность собственно персонала, т. е. человеческого мозга.

Сказанное позволяет сформулировать тезис. Интегральная производительность систем "персонал — компьютеры" зависит от двух независимых показателей: *производительности компьютеров* и *продуктивности мозга*. Первая быстро растет, вторая, наоборот, все больше отстает от растущих требований и нередко превращается в основной тормоз повышения эффективности организаций.

Почему растет производительность компьютеров? За счет роста их быстродействия и объема памяти, повышения эффективности программ и передачи им все новых функций. Однако все эти причины не оказывают ровным счетом никакого влияния на скорость работы человеческого мозга, так как последняя не зависит ни от мощности компьютеров, ни от степени автоматизации. Именно здесь коренится одна из основных причин многочисленных неудач, связанных с непродуманными попытками поднять интеллектуальную производительность персонала.

ЧТО ТАКОЕ ПРОИЗВОДИТЕЛЬНОСТЬ УМСТВЕННОГО ТРУДА?

В литературе по информатике перечисленные три понятия (интегральная производительность систем "персонал — компьютеры", производительность компьютеров и производительность персонала) обычно не расчленяются и смешиваются, что вносит путаницу и исключает возможность корректного анализа и решения задачи. В связи с этим целесообразно еще раз вернуться к постановке проблемы и обосновать ее более детально.

Уже давно признано, что узким местом стали не столько вычислительные, сколько человеческие ресурсы, поэтому задача экономии человеческих, а не машинных ресурсов стала центральной для технологии программирования. Нетрудно сообразить, что этот вывод выходит далеко за рамки программирования и имеет всеобщий характер, охватывая самые разнообразные типы сложного интеллектуального труда. Начиная с некоторого предела дальнейшее наращивание интеллектуальной мощи компьютеров, программ и сетей будет бессмысленным, если человеческий мозг окажется не в состоянии перерабатывать поступающую к нему информацию. Это означает, что уже в первые десятилетия XXI в. недостаточная продуктивность человеческого мозга станет основным фактором, ограничивающим возможность наращивания умственной мощи человеческих институтов и сдерживающим интеллектуальный прогресс цивилизации.

Учитывая вышеизложенное, необходимо устранить двусмысленность выражения "производительность умственного труда", исключить возможность ложных трактовок и дать предельно ясное и четкое определение этого важнейшего понятия. По мнению автора, наилучший путь состоит в том, чтобы ограничить неоправданно широкий объем понятия и трактовать его в узком смысле как повышение продуктивности мозга.

По определению, *эффективная деятельность* — это деятельность, которая позволяет получить нужный результат при наименьших затратах. Отсюда вытекает, что интеллектуальная деятельность является эффективной только в том случае, если она позволяет добиться качествен-

ного интеллектуального результата при наименьших интеллектуальных затратах. ***Таким образом, требование минимизации интеллектуальных усилий (минимизации затрат нервной энергии человеческого мозга, расходуемой на получение заданного интеллектуального результата) эквивалентно требованию улучшения работы ума, повышения производительности умственного труда.***

Близкую позицию занимают и другие авторы. В литературе можно встретить, например, такие выражения: повышение работоспособности мозга, совершенствование качества работы мозга [1], увеличение КПД функционирования человеческого мозга [2], "увеличение продуктивности умственного труда", связанное с "совершенствованием психических процессов человека" [3], "облегчение процесса нашего мышления" [4], "экономия мышления" и т. д.

Эти и другие факты и соображения подтверждают вывод: чтобы увеличить умственную производительность человека (который в исследовательских целях рассматривается сам по себе, так сказать, отдельно от компьютера), нужно минимизировать интеллектуальные затраты человеческого мозга на единицу созданной интеллектуальной продукции.

ЗАВИСИТ ЛИ ПРОИЗВОДИТЕЛЬНОСТЬ ПЕРСОНАЛА ОТ ПРОИЗВОДИТЕЛЬНОСТИ КОМПЬЮТЕРОВ?

От чего зависит интегральная интеллектуальная продуктивность крупной организации, активно использующей вычислительную технику? Суммарное время, которое творческий персонал затрачивает на решение сложной задачи, определяется, в частности, двумя факторами. Во-первых, временем пассивного ожидания ответа от ЭВМ; это время зависит от быстродействия компьютеров, поэтому с ростом быстродействия во многих (хотя и не во всех) случаях им можно пренебречь. Во-вторых, скоростью выполнения мыслительных операций человеческим мозгом в ходе обдумывания проблемы.

Понятно, что скорость человеческого мышления (понимаемая как скорость работы человеческого мозга при решении тех или иных производственных заданий) не зависит ни от быстродействия компьютеров, ни от объема компьютерной памяти и определяется совсем другими причинами. Какими же?

Если учесть, что при работе с компьютером свыше 99% информации человек получает с помощью зрения, то наиболее важным видом компьютерной информации (с точки зрения человеческого восприятия) является письменная, т. е. "впитываемая глазами" информация, которая определенным образом взаимодействует с мозгом и оказывает значительное влияние на его работу. В чем заключается это влияние? Зависит ли продуктивность мозга от качества поступающей в него информации? По-видимому, да. Можно предположить, что ***время решения человеческим мозгом интеллектуальных задач зависит от скорости восприятия, понимания и усвоения поступающих в мозг сообщений, а последняя — от наглядности, доходчивости, смысловой полноты и других полезных свойств информационного материала, который должен точно и наглядно отражать сущность вопроса, постановку***

задачи и ход ее решения. Если любая (в том числе самая сложная) информация будет предъявляться по принципу “взглянул — и сразу стало ясно”, если благодаря удачной форме подачи материала каждый работник быстро вникнет в суть дела и за короткое время выполнит задание, интеллектуальная производительность персонала, несомненно, будет высокой.

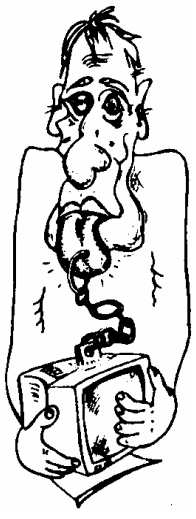
МОЖНО ЛИ УВЕЛИЧИТЬ СКОРОСТЬ РАБОТЫ ЧЕЛОВЕЧЕСКОГО МОЗГА?

Каким образом можно повысить продуктивность мозга? Ответ поясним на примерах. Замена языка римских чисел на язык арабских чисел дала возможность резко увеличить производительность труда при выполнении арифметических действий. Как отмечает Дэвид Марр, “это главная причина того, почему римская культура не смогла развить математику так, как это сделали ранние арабские культуры”.

Другой пример. Известно, что переход от программирования в машинных кодах к автокодам и ассемблерам, а затем языкам высокого уровня позволил существенно повысить производительность труда программистов. Следовательно, производительность зависит от языка: *улучшая язык, можно поднять производительность*. Исходя из этого, можно высказать предположение: при прочих равных условиях скорость решения человеческим мозгом интеллектуальных задач зависит от когнитивного качества профессионального языка, с помощью которого решаются указанные задачи.

Когнитивное качество — совокупность свойств языка, позволяющих ускорить понимание и решение задач и обеспечить эффективную верификацию (проверку). Чем выше когнитивное качество языка, тем меньше интеллектуальных усилий человек затрачивает на изучение, понимание и безошибочное решение задачи, тем выше продуктивность его мозга. Под *верификацией* здесь понимается выявление противоречий, ошибок, слабых мест, смысловой неполноты и других недостатков в ходе визуальной проверки человеком письменного решения задачи, представленного на экране или бумаге. Письменное решение (письменное представление знаний) выражается с помощью знаков, нотаций, чертежей и схем письменного языка. Причем язык здесь понимается в широком смысле: например, пользовательский интерфейс есть подмножество средств профессионального языка.

Многие авторы подчеркивают, что выбор эффективного языка может оказать благотворное влияние на продуктивность мышления. Например, Эрнст Шредер пишет, что употребление удачных знаков позволяет значительно усилить человеческое мышление. Неудачные знаки оказывают тормозящее влияние на мыш-



— За что его наказали?

— Это разработчик языков, но все его языки ужасно неудобные и трудные.

— Он признал свою ошибку?

— О, да! Теперь он говорит: чем лучше язык, тем лучше работает мозг.

ление. Знаки нужны не только для передачи другим наших мыслей, но и для формирования самих мыслей. Неудачные языки даже простую проблему способны сделать неразрешимой. И наоборот, задача, получившая удобное знаковое выражение, оказывается наполовину решенной [4].

Особенно удачными оказываются новейшие визуальные языки, которые можно охарактеризовать как эффективные методы графического представления знаний, позволяющие выявить скрытые резервы повышения продуктивности двухполушарного мозга. Джеймс Мартин и Карма Мак-Клюр отмечают: хорошие, ясные изображения играют важную роль при проектировании сложных систем и разработке программ. Наша способность мыслить зависит от языка, который мы используем для мышления. Изображения, с помощью которых мы описываем сложные процессы, являются формой языка. Подходящие изображения помогают нам визуализировать и изобретать указанные процессы. Неудачный выбор изображений может ухудшить мышление. И наоборот, применение хороших изображений может ускорить работу и улучшить качество результатов [5].

Сказанное позволяет выдвинуть следующую рабочую гипотезу: ***чтобы улучшить работу ума, повысить продуктивность человеческого мозга при решении интеллектуальных производственных заданий, необходимо улучшить когнитивные характеристики профессионального языка, используемого при выполнении интеллектуальной работы.*** Эта гипотеза положена в основу разработки языка ДРАКОН. Обоснование гипотезы можно найти в [6].

ПРОБЛЕМА ФОРМАЛИЗАЦИИ ПРОФЕССИОНАЛЬНЫХ ЗНАНИЙ

С появлением миллионов персональных компьютеров доступ к вычислительной технике помимо профессиональных программистов получили две большие группы непрограммирующих пользователей. Первую группу составляют люди относительно низкой и средней квалификации: секретари, клерки, лаборанты, кассиры и другие технические работники. Во вторую группу входят высококвалифицированные специалисты народного хозяйства и социальной сферы: ученые, конструкторы, технологи, экономисты, юристы и т. д.

С другой стороны, большая часть компьютерного парка планеты до сих пор используется для решения сравнительно простых задач. Например, в качестве пишущей машинки при оформлении документации и редактировании текстов, в режиме электронного карандаша при рисовании чертежей, диаграмм и картинок, для ввода, поиска, сортировки и пересылки информации и т. д. Хотя названные услуги несомненно полезны и экономят немало времени, они почти совсем не касаются одной из наиболее важных проблем: профессиональных знаний и умений высококвалифицированных специалистов, т. е. содержательной, творческой стороны решаемых ими сложнейших профессиональных задач.

Впрочем, этому вряд ли стоит удивляться. Ведь знания специалиста сегодня в большинстве случаев находятся в его собственной голове, а отнюдь не в компьютере. В этих условиях вполне естественно, что машина не может их обрабатывать. Чтобы выйти из положения, надо сделать, на первый взгляд, очень простую вещь: “вытащить” знания из головы специалиста и “засунуть” их в компьютер, т. е. осуществить так называемую *формализацию знаний*. При этих условиях компьютер сможет выполнять уже не поверхностную, а *глубокую* обработку знаний.

Сегодня лишь очень немногие специалисты народного хозяйства имеют опыт эффективной формализации знаний. Основная часть работников плохо представляет, о чем идет речь. Причина проста: прежние методы формализации были настолько сложны, что попросту отпугивали людей. После такой, с позволения сказать, “формализации” самые примитивные знания приобретали настолько громоздкий, противоестественный и заумный вид, что даже человек, прекрасно знающий, о чем идет речь, глядя на формализованную запись, воспринимал ее, как загадочный ребус.

Традиционное компьютерное программирование иногда рассматривают как частный случай формализации знаний. Бытует мнение, что программисты лучше других умеют формализовать свои знания. Это не совсем так. Значительная часть знаний не попадает в текст программы, оставаясь в голове программиста. Как отмечает академик А. Ершов, “язык программирования кодирует объекты предметной области задачи, а наше знание об этих объектах остается за пределами программного текста” [7]. Именно поэтому понять сложную программу в отсутствие ее автора очень трудно или даже невозможно. Приходится при-

знать, что известные методы формализации несовершенны и нуждаются в серьезном обновлении.

МОЖНО ЛИ ОБОЙТИСЬ БЕЗ КОГНИТОЛОГОВ?

Существуют две точки зрения на проблему формализации. Согласно одной из них специалист, обладающий профессиональными знаниями (обычно его называют “эксперт”), не в состоянии самостоятельно, без посторонней помощи формализовать свои знания, так как задача формализации слишком трудна. Представитель этого направления Э. Фейгенбаум подчеркивает: “По опыту нам известно, что большая часть знаний в конкретной предметной области остается личной собственностью специалиста. И это происходит не потому, что он не хочет разглашать свои секреты, а потому, что он не в состоянии сделать этого — ведь специалист знает гораздо больше, чем сам осознает”. Утрируя, можно сказать, что согласно этой позиции эксперт отчасти напоминает собаку: глаза умные, а сказать (на формальном языке) ничего не может. Отсюда делается вывод, что для решения задачи формализации необходимы особые помощники — инженеры по знаниям (когнитологи), которые, действуя по специальной методике, интервьюируют эксперта, формализуют “извлеченные” из него знания и вводят их в компьютер.

Иную позицию занимает Г. Громов, полагающий, что эксперт должен формализовать свои знания самостоятельно, без помощи инженеров по знаниям и профессиональных программистов, точнее говоря, при их “минимальной технической поддержке”. Данный метод называется “автоформализация знаний” [8] [1].

Автоформализация полезна тем, что позволяет устранить ненужных посредников и избежать ошибок типа “испорченный телефон”. Здесь однако возникает вопрос. А сможет ли эксперт формализовать свои знания? Не окажется ли задача непосильной для него?

Для таких сомнений есть веские основания, поскольку прежние методы формализации в силу своей сложности практически исключали возможность успешного достижения цели. Поэтому, если мы действительно хотим перейти к самообслуживанию при формализации знаний, первое, что нужно сделать, — это упростить технологию формализации.

---

[1] Автоформализация — это не автоматическая формализация, а самоформализация, т. е. формализация знаний, которую человек выполняет САМ.

ЧЕМ ОТЛИЧАЕТСЯ АЛГОРИТМ ОТ ТЕХНОЛОГИЧЕСКОГО ПРОЦЕССА?

Понятие технологии является весьма емким. Технологии бывают самые разные: промышленные, сельскохозяйственные, строительные, медицинские, экологические, педагогические, управленческие. Существуют и более частные понятия, например, технологии аудиторской проверки банков и предприятий и т. д. Сфера применения этого понятия постоянно расширяется. В последнее время появился термин “избирательные технологии”, т. е. технологии проведения избирательных кампаний по выборам в органы власти.

Попытаемся выяснить, есть ли сходство между понятиями “алгоритм” и “технологический процесс”? Обратимся к определениям.

Алгоритм — конечный набор предписаний, определяющий решение задачи посредством конечного количества операций [9]. Технологический процесс — совокупность приемов и способов получения, обработки или переработки сырья, материалов, полуфабрикатов или изделий, осуществляемых в промышленности, строительстве, сельском хозяйстве и других отраслях [10]. Поверхностный анализ этих определений может привести к ложному выводу, что алгоритмы и технологические процессы не имеют ничего общего. Однако в действительности это не так.

Известно, что термин “алгоритм” используется и в более широком смысле для представления человеческой деятельности в виде строгой последовательности отдельных элементарных действий или процедур [11], а технологический процесс можно определить как “последовательность направленных на создание заданного объекта действий (технологических операций), каждое из которых основано на каких-либо естественных процессах (физических, химических, биологических и др.) и человеческой деятельности” [11]. Тщательный анализ этих и многих других определений показывает, что исследуемые понятия в значительной степени совпадают, а имеющиеся различия в определенном смысле несущественны. Иными словами, технологический процесс и алгоритм — это понятия-близнецы или во всяком случае “близкие родственники”. Чтобы сделать эту мысль более убедительной, попытаемся отойти от традиционной точки зрения и предложим новые определения.

Алгоритм — последовательность информационных действий, ведущая к поставленной цели. *Технологический процесс* — последовательность информационных и физических действий, ведущая к поставленной цели. Таким образом, единственное отличие состоит в том, что в алгоритме физические действия являются запрещенными, а в техпроцессе — разрешенными. Примерами физических действий служат: транспортировка груза, нагрев детали, пуск ракеты, зашивание раны и т. д.

Для наших целей было бы удобно определить *технологию* как деятельность (последовательность действий), ведущую к поставленной цели. Согласившись с таким подходом, мы получаем возможность рассматривать алгоритм и техпроцесс как частные случаи технологии, которая приобретает статус родового понятия.

ЧТО ТАКОЕ ТЕХНОЛОГИЧЕСКИЙ ЯЗЫК?

По мнению автора, выявленное сходство понятий “алгоритм” и “техпроцесс” имеет фундаментальный характер и далеко идущие последствия. К сожалению, это сходство до сих пор не привлекало к себе должного внимания ученых, что привело к негативным результатам и в немалой степени способствовало разделению науки на “изолированные клетки”, создавая неоправданные препятствия для межотраслевых и междисциплинарных контактов. Сегодня программисты и технологи (в широком смысле слова, включая агрономов, медиков, педагогов, управленцев и т. д.) — это разные “касты”, которые получают разное образование и говорят на разных профессиональных языках. Подобные барьеры сильно затрудняют взаимопонимание между специалистами при решении проблем автоматизации и работе над междисциплинарными проектами.

Названный недостаток (трудности взаимопонимания) можно ослабить или устранить, создав единый язык, одинаково удобный для технологов, программистов и других специалистов. Для обозначения этого языка предлагается термин *технологический язык* (техноязык). Первым кандидатом на роль технологического языка является ДРАКОН.

Техноязык имеет двойное назначение. С одной стороны, он дает возможность (как и любой другой алгоритмический язык) проектировать алгоритмы, записывать программы и транслировать их в объектные коды. С другой стороны, он позволяет унифицировать запись технологических процессов любой природы в любой предметной области. Причем делать это таким образом, что унифицированная (стандартная) запись техпроцесса оказывается, во-первых, более строгой, свободной от пробелов и двусмысленностей, во-вторых, более наглядной, доходчивой и очень удобной для читателя.

Следует подчеркнуть, что цели использования технологического языка при разработке компьютерных программ и техпроцессов отличаются. В первом случае (создание программ) язык позволяет осуществить трансляцию в машинные коды. Во втором случае (описание технологий) возможны две ситуации. Если имеется автоматизированная система управления и описание технологии предназначено для компьютера, управляющего техпроцессом, описание автоматически превращается в программу компьютера, и дело сводится к предыдущему случаю. Если же автоматизированная система управления и управляющий компьютер отсутствуют или не требуются и поэтому трансляция не нужна, язык используется как средство однозначного решения задач и обеспечения взаимопонимания между людьми, что само по себе является исключительно ценным свойством языка.

Таким образом, *техноязык* — это язык нового типа, который сочетает математическую строгость алгоритмического языка с удобством языка межотраслевого и междисциплинарного общения, пригодного для наглядного описания технологий и взаимопонимания между специалистами.

Мысль о возможности и целесообразности создания универсального технологического языка опирается, в частности, на следующие предпосылки. Девяносто процентов специалистов, занятых в народном хозяйстве, не умеют программировать. Между тем эти люди успешно решают стоящие перед ними задачи. Значит, они обладают знаниями о

последовательности действий, необходимых для решения своих задач. Указанные знания можно назвать технологическими (императивными, процедурными, алгоритмическими). Таким образом, налицо любопытная ситуация: подавляющее большинство специалистов народного хозяйства обладают технологическими знаниями, но не умеют их точно выразить (алгоритмизировать), поскольку в настоящее время отсутствует легкий и удобный язык, рассчитанный на непрограммистов и предназначенный для алгоритмизации (формализации) знаний.

ТЕХНОЛОГИЧЕСКИЕ И ДЕКЛАРАТИВНЫЕ ЗНАНИЯ

Человеческие знания, выраженные с помощью любого письменного языка, можно разбить на две части: технологические[1] и декларативные.

Технологические (императивные, процедурные, алгоритмические, операторные) знания содержат сведения о последовательности информационных или физических действий, а также о выборе пути при разветвлении процессов. Примерами являются алгоритмы, компьютерные программы, а также любые технологические процессы (промышленные, сельскохозяйственные, медицинские и т. д.).

Декларативные (дескриптивные, атрибутивные, описательные) — это знания не о действиях, а об описаниях информационных и физических объектов. Примером является типичная запись в базе данных:

Фамилия	Имя	Отчество	Год рождения	Образование	Должность	Семейное положение
Иванов	Сергей	Петрович	1970	высшее	менеджер	женат

Для изложения декларативных знаний представители разных профессий нередко используют различные декларативные языки, в том числе графические (визуальные). Например, декларативные знания конструктора выражаются на языке конструкторских чертежей, электрика — на языке электрических схем, географа — на языке географических карт.

Для наших целей большой интерес представляет вопрос: можно ли создать единый универсальный язык представления профессиональных знаний, удобный для специалистов любой профессии и позволяющий улучшить взаимопонимание между ними? Для декларативных знаний ответ, очевидно, будет отрицательным. Потому что нельзя скрестить ужа с ежом и придумать разумный и полезный гибрид электрической

[1] Мы используем термин “технологические знания” и в качестве синонима “императивные знания” вместо обычно употребляемого в литературе понятия “процедурные знания”, так как последнее сильно привязано к тематике искусственного интеллекта, где оно иногда трактуется слишком узко [12]. Кроме того, термин “технологические знания” соответствует развиваемой точке зрения, согласно которой технология является общим (родовым) понятием по отношению к понятиям “алгоритм” и “техпроцесс”.

схемы и географической карты (или конструкторского чертежа). Такой путь неизбежно ведет в тупик.

Поэтому придется смириться с выводом, что специалисты разных профессий будут и впредь использовать множество самых разнообразных декларативных языков. Унификация здесь невозможна[1].

В отличие от декларативных знаний технологические знания специалистов любого профиля имеют в точности одинаковую структуру, которая нисколько не зависит от конкретной специальности и предметной области. Отсюда проистекает важный вывод: для отображения любых технологических знаний можно использовать один и тот же язык, общий для всех научных и учебных дисциплин. Это обстоятельство является существенным по трем причинам.

Во-первых, создаются благоприятные предпосылки для построения универсального технологического языка, позволяющего выражать любые технологические знания в любой предметной области в ЕДИНОЙ СТАНДАРТНОЙ ФОРМЕ.

Во-вторых, универсальный (в самом широком смысле слова) техноязык мог бы сыграть роль межотраслевого и междисциплинарного языка, содействующего решению важнейшей проблемы, — проблемы взаимопонимания между учеными и специалистами.

В-третьих, средства для описания структуры деятельности, технологические знания играют особую роль в человеческой жизни. В самом деле, человек — деятельное существо. От рождения до смерти он непрерывно действует. Деятельность выражает сущность жизни. Бездеятельность — это смерть. Поэтому знания о структуре деятельности (технологические знания) составляют важнейший компонент человеческих знаний, их основу. Можно предположить, что в системе человеческих знаний технологические знания играют фундаментальную роль — роль несущей конструкции или каркаса, который скрепляет между собой (склеивает, цементирует) отдельные фрагменты декларативных знаний. Сказанное хорошо согласуется с известным мнением, что “большинство знаний об окружающем мире можно выразить в виде процедур или последовательности действий, направленных на достижение конкретных целей” [13].

Социально-экономические успехи общества сильно зависят от разработки и внедрения новых технологий. Между тем способы описания структуры деятельности и новых технологических процессов, используемые специалистами-технологами (в широком смысле слова), недостаточно формализованы и слабо используют опыт, накопленный в алгоритмизации и программировании. С другой стороны, многие математики, алгоритмисты и программисты испытывают серьезные затруднения при необходимости наглядно и доходчиво описать сущность, структуру и содержание предлагаемых математических решений, алгоритмов и создаваемых программных комплексов и передать соответствующие знания другим людям.

В обоих случаях одна из причин этого негативного явления заключается в отсутствии подходящего формального языка, способного не

---

[1] Речь, разумеется, не идет о внутрикомпьютерных логических описаниях декларативных знаний, которые, в принципе, можно унифицировать; целесообразность подобной унификации — отдельный вопрос.

только описать проблему и ее решение, но и обеспечить высокоэффективное интеллектуальное взаимопонимание и производственное взаимодействие между людьми, что особенно важно при создании крупномасштабных проектов. Нужда в таком языке весьма велика.

Изложенные соображения позволяют сделать два вывода. Во-первых, создание технологического языка является осуществимой задачей. Во-вторых, искомый язык следует строить, в первую очередь, как язык формализации именно технологических (а не декларативных) профессиональных знаний. При этом проектирование и программирование на императивных (процедурных) языках: псевдоязык, Бейсик, Паскаль, Си, язык ассемблера и т. д. можно рассматривать как частный случай более общей проблемы — формализации технологических знаний и описания структуры деятельности.

ПОЧЕМУ НЕЛЬЗЯ ЖИТЬ ПО-СТАРОМУ?

Информационная технология — основа всех современных интенсивных наукоемких технологий. Результирующая производительность труда в наукоемких, оборонных и иных отраслях в огромной степени зависит от правильного выбора информационной технологии, от эффективного взаимодействия специалистов народного хозяйства между собой и с вычислительной техникой.

Сегодня эффективность невелика, что вызвано целой гаммой причин, в частности сложностью современных языков, их фактической непригодностью для быстрого освоения, легкого понимания и удобной формализации профессиональных знаний, в особенности при решении трудноформализуемых, так называемых слабоструктурированных задач. Именно эта сложность явилась камнем преткновения, ахиллесовой пятой информатики. Этот фундаментальный недостаток превратил формализацию профессиональных знаний и разработку сложных компьютерных программ в сверхтрудный интеллектуальный процесс, доступный сравнительно узкой социальной прослойке и недоступный всем остальным. На начальном этапе, пока компьютеров было немного, проблема не вызывала особых хлопот. Всегда можно было найти группу интеллектуалов, способных преодолеть любые сложности. Однако затем ситуация коренным образом изменилась. Вступление в эру всеобщей компьютеризации потребовало привлечения сотен тысяч и миллионов людей, интеллектуальные возможности которых существенно ниже, чем у элитных групп населения.

Драматическая недооценка чисто человеческих проблем (трудностей понимания) привела к плачевному результату. Сегодня выяснилось, что, несмотря на все усилия, значительная часть специалистов так и не научилась расшифровывать иероглифы нынешних заумных языков и, как следствие, оказалась отстраненной от плодотворного взаимодействия с программным обеспечением компьютеров, что не позволяет им автоматически и эффективно обрабатывать собственные и чужие профессиональные знания. Все это оказывает негативное влияние на интеллектуальный уровень общества, ощутимо снижает творческие возможности людей и эффективность организаций.

СОЦИАЛЬНЫЕ ТЕХНОЛОГИИ И ЭЛЕКТРОННЫЕ МЕТОДОЛОГИИ

Информационные технологии — не самоцель, а средство для улучшения промышленных и социальных технологий. *Социальные технологии* — новое и многообещающее направление [14,15], нацеленное на повышение эффективности взаимодействия людей, групп, организаций и государств (как в локальном, так и в глобальном масштабе). Неумение проектировать эффективные социальные технологии — это, пожалуй, главная причина нынешних бед цивилизации. Поэтому крайне важно научиться использовать огромный опыт человечества, накопленный при создании *технических* технологий для проектирования технологий *социальных*.

На повестке дня остро стоит вопрос о разработке нового поколения социальных технологий, способных вывести планету из нынешнего цивилизационного тупика. Это грандиозная по сложности задача. Необходимо превратить всемирную армию политиков, чиновников и специалистов (от низовых работников до руководителей государств) в профессиональных социальных технологов и социальных конструкторов, вооруженных всем богатством социально-гуманитарных и иных знаний и методов. Впрочем, это дело будущего. Сегодня делаются лишь первые шаги по совершенствованию отдельных частных аспектов социальных технологий. Одним из них является чрезвычайно интересная группа идей и подходов, направленных на качественное улучшение эффективности организаций, для обозначения которых используются выражения: “реинжиниринг бизнес-процессов”, “управление процессами”, “электронные методологии”[1]. Примеры методологий: *RAD*, *IDEF* и др. Мы ограничимся кратким знакомством с первой из них.

МЕТОДОЛОГИЯ БЫСТРОЙ РАЗРАБОТКИ СИСТЕМ *RAD*

Методология *RAD (Rapid Application Development)* служит для разработки (при активном участии пользователей) относительно небольших, но довольно сложных коммерческих информационных систем для бизнес-приложений, обеспечивающих (в этом вся суть) качественный рост эффективности организаций. Работа выполняется одним человеком или командой (до шести человек) за срок от трех до шести месяцев по строго определенной технологии. Она включает четыре этапа:

1) анализ и планирование требований,
2) проектирование,
3) конструирование,
4) внедрение,

и преследует триединую цель: обеспечить высокую скорость разработки систем при одновременном повышении качества программного продукта и снижении его стоимости [16].

---

[1] См.: *Е. Г. Ойхман, Э. В. Попов*. Реинжиниринг бизнеса: реинжиниринг организаций и информационные технологии. М.: Финансы и статистика, 1997.

Методология *RAD* является обобщением мировых достижений и ориентирована на использование мощных инструментальных средств. Она постоянно пополняется новыми изобретениями и вместе с тем содержит твердое ядро основополагающих идей.

Изложение методологии *RAD* не входит в наши планы. Руководство [16] — фундаментальный труд, насчитывающий более 800 страниц. Это энциклопедия и одновременно патетический гимн во славу *RAD*, а лучше сказать, настольная библия для разработчиков информационных систем, в которой детально расписаны указания, обеспечивающие эффективное практическое использование этого метода.

Мы ограничимся лишь самыми краткими сведениями, позволяющими ответить на вопрос: в каких отношениях находятся два понятия: “методология *RAD*” и “язык ДРАКОН”.

В 1989 г. журнал “Форчун” попытался выяснить, почему так трудно писать программы: «Программное обеспечение — это “материя чистой мысли”, бестелесная и умозрительная; поэтому проектировщики не в состоянии нарисовать ясные, точные и подробные чертежи и схемы, как это делают разработчики электронных приборов, чтобы дать четкие указания программистам, что нужно сделать. Следовательно, повседневное общение между программистами, их начальниками и обычными людьми, которые используют программы, — это вещь в себе». Однако сторонники *RAD* думают по-другому. Комментируя указанную статью, Джеймс Мартин пишет: “Важно понять, что эта народная мудрость сегодня уже неверна”. В рамках *RAD* применяются “точные и детальные чертежи и схемы (аналогичные тем, что рисуют конструкторы электронного оборудования) с помощью технологии *I-CASE*, причем из этих чертежей генерируется программный код. На уровне чертежей выполняется значительная часть проверок. Эти чертежи и схемы весьма эффективны при повседневном общении программистов, системных аналитиков, менеджеров и конечных пользователей. Попытка создавать программы без этих средств означает только одно — безответственное руководство” [16].

— Что делает этот чудак?
— Он изобретает новый визуальный язык.

Добавим, что *I-CASE (Integrated Computer-Aided Systems Engineering)* — это специальный термин, обозначающий интегрированную технологию автоматизированного создания систем, обязательный признак которой — в отличие от обычной, неинтегрированной *CASE*-технологии — наличие *автоматического* преобразования чертежей в исходный код нужного языка (и далее — в объектный код).

Чтобы спроектировать сложное бизнес-приложение, системные аналитики и пользователи должны иметь возможность рассмотреть проблему с разных сторон. Поэтому в методологии *RAD* используются различные формы чертежей (схемы "сущность—связь", схемы потоков данных, схемы действий, схемы декомпозиции процессов и т. д.), необходимые для понимания и проектирования различных аспектов создаваемого приложения.

Разные типы чертежей — это разные прожекторы, освещающие проектируемую систему с различных позиций и под разными углами; благодаря многосторонней подсветке удается хорошо понять и проработать различные детали проекта, включая самые темные углы.

Каждая из перечисленных схем представляет собой строго определенный визуальный (графический) язык. Средства *I-CASE* позволяют устанавливать точные связи между схемами, увязывая их тем самым в единую компьютерную гиперсхему, конвертировать чертежи друг в друга, хранить их значение в *репозитории* (так называется база знаний, представляющая собой общее хранилище всей информации о проекте). Репозиторий снабжен *координатором знаний (knowledge coordinator)*, который обеспечивает согласованность между различными частями знаний, хранящимися в репозитории, и проверку на правильность вновь вводимых в него данных. Информация с выхода репозитория поступает на генератор кода и, если нужно, оптимизатор. Эти и многие другие методы, средства и инструменты, предусмотренные в *RAD*, обеспечивают значительный рост производительности труда.

СХЕМЫ ДЕЙСТВИЙ И ЯЗЫК ДРАКОН

Мартин подчеркивает, что среди различных графических средств, используемых в методологии *RAD*, особенно важным является чертеж (графический язык), отображающий структуру программы. Он должен показывать оптимальную структуризацию программ, изображать циклы, вложенность конструкций, условия, структуры выбора, выходы по ошибке, обращения к базам данных, вызовы программ и другие программные конструкции.

Любопытно, продолжает Мартин, что на ранних этапах подобные чертежи отсутствовали. Возможно поэтому некоторые из ранних *CASE*-инструментов не поддерживали графических средств структуризации программ. Между тем графика дает возможность сделать структуру предельно ясной и отчетливой.

Для представления программных структур в методологии *RAD* используются специальные чертежи под названием "схемы действий" *(action diagrams)*. Эти схемы можно рисовать независимо от любого языка программирования в виде графического псевдокода, а можно настроить на какой-либо конкретный язык, например язык генератора кода. Они используются также для представления спецификаций в структурной форме.

Большинство ведущих *CASE*-технологий используют схемы действий. Мартин считает, что они "представляют собой наилучший способ для представления структурных программ и работы с ними" [16].

Автор не может согласиться с последним утверждением. По нашему мнению, хотя схемы действий обладают рядом достоинств, тем не менее *по всем наиболее важным параметрам они уступают языку ДРАКОН* и представляют собой самое слабое место *RAD*-методологии. Одна из целей создания языка ДРАКОН — улучшение характеристик методологии *RAD* за счет замены схем действий на язык ДРАКОН.

В последнее время появились новые схемы — схемы деятельности *(activity diagrams)*. Однако они также явно проигрывают по сравнению с ДРАКОНОМ.

НЕОБХОДИМОСТЬ КУЛЬТУРНЫХ ИЗМЕНЕНИЙ

Методология *RAD*, а также другие компьютерные методологии представляют собой замечательное достижение и эффектный прорыв в будущее. Вместе с тем для них характерна определенная ограниченность, так как в их задачу не входит создание подлинно “народного” межотраслевого языка (техноязыка), предназначенного для общения специалистов разных профессий, который вместе с тем обеспечил бы улучшение работы ума и мощный рост производительности в процессе автоформализации императивных профессиональных знаний.

На наш взгляд, техноязык должен повлечь за собой определенные изменения в культуре, иначе говоря — стать *новым элементом языковой культуры*. Представляется уместной следующая аналогия. Несколько столетий назад развитие машиностроения и строительного дела сдерживалось наряду с другими причинами отсутствием языка, позволяющего эффективно фиксировать необходимые знания. Общественная потребность в таком языке была чрезвычайно велика. Когда Гаспар Монж впервые предложил идеи начертательной геометрии и появились три проекции (фасад, план, профиль), это привело к формированию, закреплению в стандартах и всемирному распространению языка конструкторских и строительных чертежей. Последний стал одним из важнейших языковых средств технической цивилизации и одновременно достоянием мировой культуры. Человечество получило столь необходимый и давно ожидаемый языковый инструмент для фиксации и обогащения соответствующих знаний. В итоге развитие промышленности заметно ускорилось.

Думается, сегодня имеет место примерно такая же ситуация, ибо сформировалась общественная потребность в техноязыке, как инструменте интеллектуального взаимопонимания и взаимодействия людей, межотраслевого синтеза и автоформализации императивных знаний, эффективного описания структуры деятельности, проектирования социальных технологий, улучшения работы ума. Известно, что "многие люди с активным умом и блестящими идеями оказываются не в состоянии заставить своих коллег точно понять, что они имеют в виду", причем подобная ситуация, вызванная неадекватностью используемых языковых средств, является одной из причин недостаточной восприимчивости современной индустрии к новым идеям [17]. Эти и другие соображения свидетельствуют о том, что для улучшения взаимопонимания необходимо провести отнюдь не простые системные изменения в культуре. По-видимому, в качестве одного из первых шагов целесообразно сделать техноязык частью системы образования в школе и вузе, причем не как факультатив, а как обязательный компонент учебной программы.

ТЕХНОЯЗЫК КАК ЭЛЕМЕНТ КУЛЬТУРЫ

Изложим без доказательства пять тезисов по вопросу о возможной роли языка ДРАКОН в человеческой культуре. Автор отчетливо сознает дискуссионный характер тезисов, однако питает надежду, что, дочитав книгу до конца, вы, возможно, посчитаете их хотя бы частично обоснованными.

- ! В настоящее время ощущается острая необходимость в создании специального высокоточного средства для улучшения интеллектуального взаимодействия между людьми — техноязыка, который призван стать новым элементом языковой культуры человечества.
- ! Несмотря на то, что в современных компьютерных методологиях, в частности в методологии *RAD*, применяется значительное число — свыше десятка — различных графических языков (схемы "сущность—связь", схемы потоков данных, схемы действий и т. д.), с точки зрения человеческой культуры все они имеют лишь локальную эффективность как инструменты создания информационных систем и не удовлетворяют требованиям, предъявляемым к техноязыку. Это значит, что ни один из этих языков не может стать новым элементом языковой культуры человечества, который следует рекомендовать для массового изучения в школах и вузах.
- ! Среди указанного семейства графических языков единственным более или менее приемлемым средством для представления технологических знаний и описания структуры деятельности являются схемы действий и схемы деятельности. Однако они обладают серьезными недостатками и по своим эргономическим и дидактическим характеристикам значительно уступают языку ДРАКОН. Поэтому в качестве техноязыка и нового элемента культуры следует использовать ДРАКОН, а не схемы действий.
- ! Ставка на язык ДРАКОН является оправданной, так как она позволяет осуществить системные изменения в культуре: в системе обра-

зования, на производстве (программы и технологии), в науке (улучшение общения между учеными), в социальных технологиях и других областях.

! Обучение языку ДРАКОН целесообразно начать в средней школе, используя единую визуальную форму для записи структуры деятельности, алгоритмов, программ, технологий, а также любых императивных знаний, связанных с другими школьными предметами. Это позволит с самого начала преодолеть ныне существующий разрыв между алгоритмическими и технологическими знаниями, укрепить межпредметные связи, повысить качество школьного обучения за счет системного подхода к развитию деятельностного, алгоритмического и технологического мышления и совершенствованию визуального интеллекта учащихся.

ВЫВОДЫ

1. Необходимо различать три понятия: производительность системы “персонал — компьютеры”, производительность компьютеров и производительность персонала. Последняя не зависит от характеристик компьютеров и определяется характеристиками языка. Чтобы повысить продуктивность персонала, нужно улучшить когнитивные характеристики профессионального языка, используемого специалистами при выполнении интеллектуальной работы.
2. Разработку прикладных программ во многих случаях должен выполнять не посредник — программист, а конечный пользователь — специалист народного хозяйства. Программирование должно уступить место автоформализации профессиональных знаний, которая позволяет получить тот же конечный результат, что и программирование, т. е. отлаженный объектный код.
3. Сегодня формализация знаний — дорогостоящий и трудоемкий процесс. Задача состоит в том, чтобы увеличить производительность труда в процессе автоформализации технологических знаний, при описании структуры деятельности примерно на два порядка.
4. Языки представления декларативных знаний не поддаются унификации. С технологическими знаниями ситуация иная. Существует возможность построить единый универсальный технологический язык (техноязык).
5. Автоформализация технологических знаний, описание структуры деятельности — один из важнейших видов интеллектуального труда. Чтобы добиться резкого повышения производительности этого труда и сделать автоформализацию доступной практически для всех специалистов, нужно создать межотраслевой техноязык, обладающий высокими когнитивными характеристиками.
6. Техноязык следует использовать в современных компьютерных методологиях, что заметно повысит их эффективность.
7. Техноязык должен стать элементом языковой культуры человечества. Его изучение должно быть предусмотрено в средней и высшей

школе. Для этого техноязык нужно специально приспособить, сделать легким, доступным для школьников и студентов.

Глава 4

ПОНИМАНИЕ И ВЗАИМОПОНИМАНИЕ — КЛЮЧЕВЫЕ ПРОБЛЕМЫ ИНФОРМАТИКИ

Понимание — функция разума, главная его функция...
Наталья Автономова

ОТСУТСТВИЕ ПОНИМАНИЯ ВЕДЕТ К МИЛЛИОННЫМ УБЫТКАМ

Главным требованием к языку ДРАКОН считается облегчение и улучшение работы ума, улучшение понимаемости проектов, алгоритмов, программ и технологических знаний. Для обозначения данного требования вводится понятие "критерий сверхвысокой понимаемости". Считается, что язык удовлетворяет этому критерию, если написанные на нем проекты, алгоритмы, программы и технологии обладают наивысшим когнитивным качеством.

Вспомним общеизвестные факты не столь уж далекого прошлого. Один из руководителей фирмы IBM Джозеф Фокс сообщает, что в начале 1970-х гг. две основные авиакомпании возбудили дело против своих подрядчиков, поскольку созданная ими система обработки данных стоимостью 40 млн. долл. не только не работала, но и вообще не подавала никаких признаков жизни. Крупный европейский банк направил в суд иск о взыскании 70 млн. долл., выплаченных за программное обеспечение. Военно-воздушные силы США затратили более 300 млн. долл. на тщетную попытку автоматизировать комплексную систему перевозок и снабжения. Продолжая тему, Альгирдас Авиженис отмечает случаи, когда сложные и дорогие системы так и не смогли заработать из-за того, что в рамках установленных сроков и средств не удалось устранить ошибки в программном обеспечении. Джон Муса указывает: эксплуатационные и экономические последствия программных ошибок становятся "поистине ужасными". Причина всех этих "земных катастроф" больших программистских проектов, как правило, заключалась в том, что заказчик и исполнитель совершенно по-разному понимали задачу. Грубо говоря, заказчик надеялся получить систему "про Фому", а исполнитель сделал "про Ерему".

Перечисленные классические примеры крупномасштабных неудач, а также многие другие факты, в том числе эпидемия казавшихся непонятными провалов проектов АСУ[1] в нашей стране хотя и стали достоянием истории, тем не менее остаются поучительными до сих пор. Анализ подобных инцидентов позволяет сделать четыре вывода:

1) успех крупного проекта напрямую зависит от взаимопонимания между его участниками;

[1] АСУ — автоматизированная система управления.

2) чтобы добиться взаимопонимания, нужны недели и месяцы, а в сложных случаях — годы кропотливой совместной работы заказчика и исполнителя;
3) проблему взаимопонимания нельзя считать решенной до сих пор; достижение взаимопонимания — тяжелый и сложный интеллектуальный труд, производительность которого крайне низка;
4) было бы крайне желательно формализовать этот труд и резко увеличить его производительность.

ИЗДЕВАТЕЛЬСТВО НАД ЗДРАВЫМ СМЫСЛОМ ПОД НАЗВАНИЕМ "АБСОЛЮТНО ПРАВИЛЬНАЯ ПРОГРАММА"

Во всех перечисленных случаях создаваемые системы подвергались тщательной проверке. Возникает вопрос: почему столь мощный инструмент как тестирование не позволил выявить грубейшие ошибки в программном продукте?

Ответ понятен. В описанных примерах начало работы было на первый взгляд вполне разумным. Исполнитель заблаговременно составил и согласовал с заказчиком многотомный документ — техническое задание на разработку программного комплекса (формальные спецификации) и только после этого приступил к работе: проектированию, программированию и тестированию. Тестирование показало, что код программ был безошибочным и точно соответствовал спецификациям. Словом, программа была абсолютно правильной.

Парадокс в том, что программа называется правильной, если она соответствует техническому заданию. А если ошибки умудрились просочиться в "святая святых" — задание на разработку системы? Вот тут-то и зарыта собака! Оказывается, тестирование — отнюдь не панацея. Иными словами, тестирование программ, даже самое придирчивое и дотошное, в большинстве случаев в принципе не позволяет обнаружить ошибки в спецификациях.

СПЕЦИФИКАЦИИ ПРОГРАММ — ВОТ ГЛАВНЫЙ "ГАДЮЧНИК"!

Со временем стало ясно, что одетые в "шапки-невидимки" ошибки в спецификациях наиболее коварны и представляют собой основную опасность. Они практически неуязвимы для лобовой танковой атаки массированного тестирования. Однако наиболее сенсационным стал вывод о том, что подготовка спецификаций — один из основных источников ошибок. Выяснилось, что на устранение ошибок в требованиях на программы уходит в среднем 82% всех усилий, затраченных коллективом разработчиков на устранение ошибок проекта, тогда как на устранение ошибок кодирования — 1%.

Ошибки в спецификациях обнаруживаются обычно лишь после окончания приемосдаточных испытаний разработанной системы. Они "всплывают как мины на фарватере, уже на стадии внедрения и сопровождения готового программного продукта в организации-заказчике" (Г. Громов, 1993).

Проблема спецификаций — одна из центральных в программировании. Джеймс Мартин пишет: “То и дело встречается одна и та же печальная история: после нескольких лет напряженной разработки системы обработки данных конечные пользователи заявляют, что это вовсе не то, что они хотели... Обычная реакция разработчиков на такую скверную ситуацию — заявление, что требования к системе не были определены пользователем достаточно полно. И это несмотря на то, что требования к системе иногда представляются в виде многих томов документации”.

Что же является причиной ошибок в спецификациях? Рассмотрим вопрос на примере разработки АСУ технологическими процессами атомной электростанции (АСУ ТП АЭС). Заказчики (разработчики АЭС) прекрасно знают “физику процесса”, т. е. технологию работы реакторного и турбинного отделений и других систем АЭС, но, к сожалению, не являются профессорами по части АСУ и программирования. Исполнители (разработчики АСУ ТП АЭС), наоборот, детально знакомы с компьютерами, программированием, особенностями построения систем управления, но, увы, мало что смыслят в реакторах, спецводоочистке и прочих секретах АЭС. Таким образом, проблема состоит в том, что те и другие должны научиться понимать друг друга. Для этого нужно произвести взаимное обучение, взаимный обмен знаниями, которые обычно представлены в виде той или иной документации. Успех взаимного обучения во многом зависит от когнитивных свойств используемого профессионального языка (языка представления знаний). Очевидно, что речь идет о чрезвычайно тонких, деликатных и сложных когнитивных проблемах, в первую очередь, о проблеме понимания.

Проведенный анализ позволяет сделать два замечания:

1) ошибки в спецификациях представляют собой одну из наиболее сложных проблем теории и практики программирования;
2) нерешенность этой проблемы связана с целой гаммой причин, среди которых не последнее место занимает недооценка когнитивных проблем и отсутствие эффективных языковых средств для облегчения и улучшения работы ума и решения проблемы понимания.

СПЕЦИФИКАЦИИ ПРОГРАММ И МЕТОДОЛОГИЯ *RAD*

В методологии *RAD* используется весьма оригинальный подход к проблеме спецификаций — что-то вроде "умный в гору не пойдет, умный гору обойдет". В самом деле, зачем создавать трудоемкие бумажные спецификации, если при наличии *CASE*-инструментов гораздо проще получить работоспособный прототип системы!

Новый подход тесно связан с понятием "качество программного продукта". По мнению Джеймса Мартина, раньше большинство организаций пользовались неудачным определением этого понятия. Они понимали его как "максимально возможное соответствие письменным спецификациям". При традиционном подходе спецификации, подписанные пользователем, замораживались на весь период, пока выполнялись проектирование, кодирование и тестирование. Зачастую это приводило к тому, что внесение изменений в спецификации запрещалось на срок до восемнадцати месяцев и разрешалось лишь после запуска системы в работу. Естественно, за это время бизнес-требования к проектируемой системе успевали существенно измениться, а созданная система полностью игнорировала этот факт! Так что заказчик был вынужден начинать работу с заведомо непригодной системой, ибо она учитывала не все, а лишь часть его требований, не говоря уже об ошибках в спецификациях, вызванных просчетами пользователя, в связи с чем часть исходных требований была неточна, а многое вообще упущено из виду.

Так было. Однако методология *RAD* дает новое определение качества программ, понимая его как "максимально возможное удовлетворение истинных бизнес-требований (требований пользователя) на момент предъявления работающей системы заказчику". Чтобы этого добиться, пришлось многое изменить: резко увеличить скорость разработки с помощью *I-CASE*-инструментов и, сверх того, радикально улучшить взаимоотношения разработчика и пользователя. Если раньше пользователю "выкручивали руки", заставляя подписывать бумажные спецификации, которые он плохо понимал, то теперь ситуация изменилась. После короткой серии четко организованных начальных ознакомительных переговоров, результаты которых немедленно вводятся в компьютер, разработчик быстро создает компьютерный прототип системы и немедленно передает его пользователю. Последний, сидя за компьютером, пробует проект "на зуб", осознает свои заблуждения и направляет разработчику ответную порцию уточнений. Разработчик быстро изменяет прототип и тут же выдает пользователю усовершенствованную версию. Подобная игра в пинг-понг между разработчиком и пользователем повторяется несколько раз и приводит к тому, что прототип постепенно превращается в работающую систему.

Главная хитрость в том, что пользователи больше не должны покупать кота в мешке и подписывать кишащие ошибками бумажные спецификации (разобраться в которых — выше их сил). Они ставят подпись на проекте, полученном с помощью инструментария *I-CASE*, который вполне доступен их пониманию и который они могут профессионально оценить. Таким образом, действия пользователя, связанные с контролем проекта, имеют компьютерную точность. Участвуя в отра-

ботке проекта за экраном компьютера, пользователь гораздо глубже вникает в детали, чем при умозрительном анализе бумажных спецификаций. Изложенный метод иногда называют “раннее прототипирование при спиральном цикле разработки”, потому что прототип тестируется заказчиком на каждом витке спирали, чтобы снизить вероятность ошибки в законченной системе.

Нет сомнения, что перечисленные новшества весьма полезны. Однако нельзя забывать два обстоятельства. Во-первых, *RAD* — это не общая, а частная методология, так как она применима не к любым системам, а лишь к системам определенного класса (бизнес-приложениям). Во-вторых, наряду с достоинствами, методология *RAD* имеет и существенные изъяны. К ним относятся недооценка важности проблемы улучшения работы ума, проблемы понимания и когнитивно-эргономического подхода к проектированию языковых средств. Вследствие этого графические и иные средства *RAD* имеют слабые места и нуждаются в улучшении.

По мнению автора, неумение осознать и поставить во главу угла приоритетную роль проблемы улучшения работы ума и проблемы понимания является общим недостатком многих современных подходов к созданию языковых средств проектирования и программирования.

КОНЦЕПЦИЯ КОГНИТИВНОГО ПРОГРАММИРОВАНИЯ

При разработке нового языка программирования обычно стараются найти разумный компромисс между различными, нередко противоречивыми требованиями, которые, в частности, включают следующие:

1) легкость понимания программ;
2) небольшая трудоемкость написания программ;
3) минимизация потребной машинной памяти;
4) малое время выполнения программ;
5) небольшое время трансляции;
6) легкость автоматизированного выявления ошибок.

Перечисленные требования можно разбить на две группы. Группа *когнитивных* требований включает легкость написания программ и возможность их быстрого и глубокого понимания. *Машинные* требования охватывают все остальное: экономию машинных ресурсов, малое время выполнения и трансляции программ и т. д.

Разработка языка ДРАКОН опирается на концепцию когнитивного программирования, в основе которой лежат следующие постулаты.

ПОСТУЛАТ 1. *Когнитивные требования к языку рассматриваются как основные, машинные — как второстепенные.* Обоснование постулата состоит в том, что сегодня, когда быстродействие компьютеров и объем памяти резко возросли, а их удельная стоимость снизилась, основной проблемой является низкая производительность персонала, поэтому улучшение работы ума, повышение продуктивности человеческого мозга является задачей номер один.

ПОСТУЛАТ 2. *Легкость понимания программ — более важное требование, чем удобство их написания.* Как отмечает Я. Пайл, возможность прочитать программу и отчетливо осознать ее смысл гораздо важнее, чем возможность кратко и быстро ее написать. Причиной служит однократное выполнение работы автором программы и необходимость многократного чтения программы в течение ее жизненного цикла[1]. Известно, что высокая удобочитаемость программ облегчает их сопровождение.

ПОСТУЛАТ 3. *При создании языка выполнение когнитивных и машинных требований следует осуществлять в два этапа, используя разные средства.* На первом этапе основное внимание следует сосредоточить на реализации когнитивных требований и (в разумной степени) игнорировать вопросы машинной эффективности программ. При таком подходе использование языка приведет к созданию гарантированно понятных, но, возможно, неэффективных программ. На втором этапе (который во времени может перекрываться с первым) должна решаться проблема машинной эффективности программ, для чего следует использовать:

1) оптимизирующие трансляторы нового поколения;
2) методы автоматического улучшения (оптимизации) программ, обеспечивающие преобразование неэффективных, но понятных программ в эквивалентные, более эффективные;
3) методы интеллектуализации компьютеров;
4) улучшение характеристик компьютеров до границ, делающих массовую эксплуатацию неэффективных (или частично неэффективных) программ экономически приемлемой и даже выгодной;
5) введение в основной язык дополнительной возможности, позволяющей писать отдельные куски программ на языке ассемблера. Такая возможность используется в случае, когда неэффективность программы, написанной на основных средствах языка, выходит за пределы разумного.

Преимущество двухэтапного подхода к реализации когнитивных и машинных требований состоит в том, что он позволяет ослабить давление машинных требований на язык программирования, облегчая задачу разработчиков языка и позволяя им сконцентрировать усилия на коренном улучшении тех свойств языка, от которых зависит решение наиболее важной задачи — кардинального повышения производительности труда персонала.

---

[1] Один из апостолов компьютерного мира Чарлз Хоар говорит: “Разве не привело бы нас в восторг, если бы добрая фея предложила вам взмахом своей волшебной палочки над вашей программой убрать все ошибки с одним только условием — вы должны переписать и ввести всю вашу программу три раза!”

Таким образом, *парадигма когнитивного программирования рассматривает критерий улучшения работы ума и сверхвысокого понимания как главное требование к языку* (хотя, разумеется, в жизни всегда возможны некоторые исключения).

ВЫВОДЫ

1. Чтобы решить проблему понимания и сократить экономический ущерб, вызванный взаимным непониманием между заказчиками, разработчиками и эксплуатационниками, необходимо принять концепцию когнитивного программирования и коренным образом изменить приоритеты при создании языков нового поколения.
2. Сегодня первостепенное значение приобретает требование облегчения и улучшения работы ума, минимизации интеллектуальных затрат персонала, расходуемых на создание и сопровождение программного продукта в течение всего жизненного цикла.
3. Значительная или даже основная доля интеллектуальных усилий персонала при разработке сложных проектов затрачивается на процесс познания, на восприятие и понимание информации. Поэтому требование познаваемости проектов и алгоритмов и связанный с ним критерий сверхвысокой понимаемости становятся определяющими.

Глава 5

ПРОБЛЕМА УЛУЧШЕНИЯ РАБОТЫ УМА: НОВЫЙ КОГНИТИВНЫЙ ПОДХОД

Разум! Когда же кончится столь долгое несовершеннолетие твое!

Уильям Гэзлит

ТЕКСТ КАК ЗРИТЕЛЬНАЯ СЦЕНА

Сетчатка человеческого глаза состоит из 126,5 миллионов чувствительных элементов — рецепторов. Она способна воспринимать свет с различной длиной волны (в пределах 380...760 нанометров), отражаемый объектами, находящимися в поле зрения, и преобразовывать его в электрические импульсы, которые направляются в высшие отделы мозга по зрительному нерву. Свет воздействует на чувствительное вещество (родопсин) рецепторов, вызывает изменения в структуре белка (опсина) и запускает цепь биохимических процессов в сетчатке, инициирующих передачу импульсов по зрительному нерву [1].

Какую информацию о физических объектах внешнего мира передают эти импульсы? Глаз способен воспринять не любые характеристики физических объектов, а только те, что связаны со световой энергией, т. е. *оптические* характеристики. Для глаза внешний мир — это оптический внешний мир, а составляющие его объекты — оптические объекты. Импульсы, бегущие в мозг по зрительному нерву, несут информацию только об оптических свойствах внешнего мира. Следовательно, с точки зрения глаза, любой текст и любая компьютерная программа, находящаяся в поле зрения, — это всего-навсего оптическое явление, т. е. оптический текст и оптическая программа.

Диосцена — Двумерная Информационная Оптическая сцена, предназначенная для зрительного восприятия информации человеком, целиком лежащая в поле зрения и предъявляемая человеку на бумаге или экране компьютера. Диосцена обычно имеет форму прямоугольника, размеры которого удобно задавать с помощью понятия “формат диосцены”. Для простоты ограничимся форматами, принятыми в стандарте ЕСКД (Единая Система Конструкторской Документации): A0, A1, A2, A3, A4, A4×4. Для зрительного восприятия важен не только формат диосцены, но и ее телесный угол, зависящий от расстояния между глазом и диосценой.

Диопрограмма — это любая компьютерная программа (будь то один из чертежей технологии *I-CASE*, исходный текст, распечатка объектного кода и т. д.), рассматриваемая как диосцена.

СИМУЛЬТАННОЕ И СУКЦЕССИВНОЕ ВОСПРИЯТИЕ

За миллионы лет эволюции система “глаз — мозг” изменялась таким образом, чтобы решать две задачи: “видеть, как можно больше (одномоментно), и видеть, как можно отчетливее” [2].

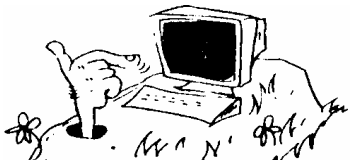
Для решения первой задачи у человека, во-первых, сформировалось поле зрения чрезвычайно больших размеров: 100...110° по вертикали и 120...130° — по горизонтали; во-вторых, образовался аппарат *периферийного* зрения. Для решения второй задачи в сетчатке глаза возникла область высокой остроты зрения (фовеа) и образовался аппарат *центрального* (фовеального) зрения [2].

Глаз и мозг способны работать в двух режимах: *симультанном* (быстрый панорамный прием обзорной информации с помощью периферийного зрения) и *сукцессивном* (медленный прием детальной информации с помощью центрального зрения). Их оптимальное сочетание позволяет получить важный приспособительный эффект. При симультанном *(simultaneous)* восприятии система “глаз — мозг” обладает способностью быстро, практически мгновенно воспринимать огромные объемы зрительной информации. Симультанно мы воспринимаем человеческие лица, картины природы, уличные сценки и многое другое. При сукцессивном *(successive)* восприятии производится тщательный последовательный анализ важной информации, первичное выделение которой произошло в ходе симультанного восприятия.

При чтении длинного словесного текста глаз и мозг работают преимущественно в сукцессивном режиме (т. е. медленно), при восприятии изображений доминирует симультанный (быстрый) режим. Если одну и ту же информацию можно представить и в текстовой, и в графической форме, последняя обеспечивает более высокую скорость понимания за счет того, что преимущественно сукцессивный режим восприятия текста заменяется на преимущественно симультанный режим анализа изображения [3].

КАК ПОВЫСИТЬ ПРОДУКТИВНОСТЬ ЧЕЛОВЕЧЕСКОГО МОЗГА?

Сетчатка человеческого глаза является двумерной поверхностью, поэтому любую информацию о внешнем мире фотокамера глаза превращает в двумерное изображение на сетчатке. Нас интересует восприятие двумерных искусственных зрительных сцен (диосцен и диопрограмм). Полезно заметить, что искусственные двумерные сцены и естественные трехмерные сцены окружающего мира имеют “общую судьбу”, ибо фотокамера глаза приводит их к одному знаменателю, преобразуя в двумерный сетчаточный зрительный образ. Учитывая этот факт и опираясь на известные результаты эволюционной теории, нейробиологии и психологии, можно сформулировать несколько постулатов, которые имеют важное значение для дальнейшего анализа.



— О чем говорит вещий голос?

— Информационные технологии, игнорирующие проблему диосцен, будут отмирать.

- Чтобы повысить продуктивность мозга при работе с текстами и чертежами, необходимо реализовать возможно более полное использование тех особенностей работы глаза и мозга, которые выработались за миллионы лет эволюции, выбрать наилучшее сочетание симультанного и сукцессивного восприятия, центрального и периферийного зрения.
- Сетчаточный образ диосцены по своим параметрам должен (в пределах разумного) уподобляться сетчаточному образу естественных зрительных сцен. Структура, размеры и телесный угол искусственных зрительных сцен (диосцен) во многих отношениях могут и должны быть приближены к соответствующим параметрам естественных сцен, что позволяет с наибольшей эффективностью использовать уже имеющиеся в человеческом мозгу механизмы, сформировавшиеся в процессе восприятия естественных сцен и, прежде всего, мощные механизмы симультанного восприятия.
- Чтобы задействовать указанные механизмы как можно лучше в целях повышения продуктивности мозга, диосцены, их оптический алфавит и оптический синтаксис следует проектировать с помощью специально разработанных научно-обоснованных когнитивных методов. Для этого когнитивно-значимые параметры диосцен нужно согласовать с конструктивными характеристиками глаза и мозга.
- Диосцена должна отражать не только сущность, но и структуру проблемной ситуации, облегчать мышление, придавать ему большую точность и силу.
- Желательно, чтобы диосцена выглядела не как набор изолированных фрагментов с разорванными линиями, а как законченный целостный зрительный образ, имеющий четкий контур, причем фрагменты этого образа также имели бы контур из замкнутых линий. *Наличие замкнутых контуров облегчает выделение смысловой фигуры из зрительного фона* как на уровне всего чертежа, так и на уровне его фрагментов [1].

Наша ближайшая цель состоит в том, чтобы подвести читателя к следующему выводу: ***замена текста эквивалентным ему чертежом (например, переход от текстового программирования к визуальному) обеспечивает более высокую продуктивность мозга за счет "симультанизации", т. е. увеличения скорости работы мозга при переходе от медленного сукцессивного восприятия текста к быстрому симультанному восприятию чертежа.***

---

[1] Известный русский физиолог И. Сеченов считал, что исходным моментом отражения предмета является его контур, т. е. отделенность по известным граням от окружающего фона. Он называл контур "раздельной гранью двух реальностей".

КОГНИТИВНЫЙ НЕДОСТАТОК ТЕКСТОВОГО ПРЕДСТАВЛЕНИЯ ЗНАНИЙ

Органический, принципиально неустранимый порок текстового представления знаний (в частности, текстового программирования) состоит в том, что оно не позволяет задействовать огромные резервы производительности человеческого мозга, связанные с его способностью к скоростной обработке больших массивов симультанно воспринимаемой информации.

Обоснование этого вывода состоит в следующем. Если встать на позиции нейробиологии и когнитивной эргономики, то — вопреки общепринятой точке зрения — изобретение текстовых книг, текстового программирования и компьютеров с текстовым пользовательским интерфейсом было в некотором смысле "противоестественным" событием. Во-первых, рабочее поле зрения ощутимо сузилось, так как телесный угол страницы текста или экрана текстового дисплея во много раз меньше физиологического поля зрения. Во-вторых, значительно уменьшилась скорость обработки информации в мозгу, ибо зрительный анализатор, созданный эволюцией прежде всего для быстрого симультанного восприятия огромных массивов информации, находящейся в широкоугольном поле зрения, мгновенного выделения из нее наиболее важных сведений и быстрого принятия решения, принудительно начал работать в искусственно замедленном и потому неэффективном сукцессивном режиме, неизбежном при чтении текста. Таким образом, ювелирная работа эволюции по формированию мощных симультанных механизмов периферийного зрения при чтении текста оказалась частично невостребованной. Возник нежелательный перекос в распределении нагрузки между двумя зрительными системами — центральной и периферийной, в результате чего роль последней оказалась ослабленной.

Между тем периферийная система важнее центральной в том смысле, что "для адекватного понимания зрительной сцены важнее способность к одномоментному крупномасштабному схватыванию отношений между предметами, чем возможность тонкого... анализа отдельных деталей" [2]. Из клинической практики известно, что поражение всех зон поля зрения, кроме центральной (фовеальной) области *практически равносильно слепоте*. Таким образом, создав текстовые книги, текстовое программирование и компьютеры с текстовым пользовательским интерфейсом, т. е. искусственно отключив значительную часть периферийного зрения, авторы названных изобретений обрекли читателей деловой литературы, программистов и пользователей на частичную "слепоту", образно говоря, выключили из работы значительную часть их мозга, вследствие чего громадные резервы коллективного интеллекта этих людей не используются.

Во всех случаях, когда дальнейший рост продуктивности мозга работников становится неотложной и приоритетной задачей, можно прогнозировать, что текстовое представление знаний и текстовое программирование будет уступать место визуальному (графическому).

Когнитивную сущность изложенных соображений можно охарактеризовать как *принцип симультанизации*: если текст и чертеж эквивалентны (содержат одну и ту же информацию), замена текста удачным

чертежом увеличивает продуктивность мозга работников за счет более активного включения в работу симультанных механизмов восприятия и мышления. Слово “мышление” добавлено не случайно, ибо, как показал В. Глезер, можно “отождествить зрительное восприятие с конкретным предметным мышлением” [4].

КАКИМ ДОЛЖЕН БЫТЬ ФОРМАТ ДИОСЦЕНЫ?

При традиционном подходе вопрос о выборе формата диопрограммы (т. е. ее габаритов — высоты и ширины) перед программистом почти никогда не встает. В самом деле, если компьютер и принтер уже куплены, то размеры экрана дисплея и формат страницы листинга жестко заданы. Вместе с тем следует иметь в виду, что выбор оптимального формата может оказать сильное влияние на производительность труда.

Итак, каким должен быть формат диосцены? Какой формат диопрограммы обеспечивает максимальную продуктивность человеческого мозга? Ответ будет разным для текстового и визуального программирования.

Для текстового случая обычно используемый формат экрана и принтера А4, видимо, близок к оптимальному. Однако для визуального программирования дело обстоит иначе.

Принцип зависимости эффективности восприятия от используемой доли поля зрения. Скорость симультанного восприятия визуальной диосцены зависит от ее габаритов и телесного угла, т. е. от фактически используемой доли поля зрения. Если эта доля невелика, скорость будет незначительной, если велика — большой. Если доля слишком мала, следует говорить о недоиспользовании возможностей симультанного восприятия, т. е. об искусственно вызванной частичной “слепоте”, которая отрицательно влияет на продуктивность мозга.

Другой недостаток состоит в том, что необоснованное уменьшение формата визуальной диосцены может привести к *когнитивной перегрузке мозга*. Предположим, имеется чертеж визуальной программы, занимающий всю площадь большого листа бумаги формата А1. Предположим далее, что мы в исследовательских целях разрезали его на восемь страниц формата А4 и стали поочередно предъявлять их человеку, который должен прочитать чертеж программы и детально в ней разобраться.

Чтобы выявить суть дела, прибегнем к аналогии. Допустим, нужно опознать человека по фотографии. Если перед нами нормальная фотография, задача решается легко. Но что будет, если фотография разорвана на восемь частей, на которые запрещается смотреть одновременно и которые предъявляются только по очереди? Очевидно, при таких условиях решение задачи опознания становится чрезвычайно трудным, а с увеличением числа фрагментов — невозможным.

Возвращаясь к примеру с чертежом программы, можно сказать, что ***расчленение целостного зрительного образа визуальной программы на несколько фрагментов есть искусственно вызванное усложнение задачи, приводящее к неоправданной перегрузке мозга.*** В условиях подобного расчленения 95% интеллектуальных усилий тратится на надуманную работу по воссозданию целостного зрительного образа

и лишь 5% — на решение основной задачи (понимание программы). В самом деле, чтобы понять смысл ансамбля из восьми диосцен, читатель должен, постоянно листая страницы программы взад и вперед, поочередно прочитать, понять и запомнить все восемь фрагментов чертежа программы, найти и мысленно соединить все отрезки, обозначающие переход линий с листа на лист; после этого его мозг должен “склеить” фрагменты в единый взаимоувязанный образ. Все эти трудозатраты оказываются ненужными, когда мы смотрим на диопрограмму формата А1.

Для условий описанного примера справедливо заключение: если заменить восемь “рваных кусков” формата А4 на единую стройную картину формата А1, можно значительно увеличить скорость мышления и продуктивность мозга программиста. Этот вывод подтверждается мировой практикой создания машиностроительных чертежей и электрических схем, где широко используются большие форматы А1 и А0, что нашло закрепление в соответствующих международных и национальных стандартах.

Изложенные соображения позволяют сформулировать *принцип приоритета целостного образа*. Если имеются два эквивалентных графических представления одной и той же программы: 1) в виде одного большого чертежа (например, формата А4×4, А1 или А0), который хорошо отражает структуру проблемной ситуации в форме целостного и стройного визуального образа, и 2) в виде набора из нескольких маленьких чертежей (например, четырех, восьми или шестнадцати форматок А4), то замена набора мелких чертежей на один большой увеличивает продуктивность мозга за счет увеличения активно используемой доли физиологического поля зрения, замены образов памяти на образы восприятия, устранения паразитной когнитивной нагрузки и более эффективного использования симультанных механизмов.

КОГНИТИВНЫЕ РЕКОМЕНДАЦИИ

Говорят, один рисунок стоит тысячи слов, и это действительно так при условии, что рисунок хороший. Последнее условие является существенным, так как неумелое использование чертежей и рисунков может принести только вред. Возникает вопрос: при каких условиях замена текста изображением дает максимальный когнитивный выигрыш с точки зрения интенсификации работы мозга?

Говоря упрощенно, таких условий всего два: визуальная диосцена должна иметь, во-первых, хорошие размеры, во-вторых, хорошую структуру[1].

Что значит хорошие размеры? Размеры зависят от когнитивной сложности проблемы. Для более простых случаев можно использовать форматы А4 и А3, для более сложных — форматы А4×4, А1, для особо сложных А0. Напомним еще раз, что все эти форматы прошли массовую проверку в мировой инженерной практике. Они начинают применяться и в практике программирования. Например, при использовании *CASE*-инструмента *ProKit WORKBENCH* фирмы *McDonnel Douglas Information Systems* используются программные чертежи размером 3 × 4 фута (91 × 122 см) — что-то среднее между форматами А1 и А0.

Что значит хорошая структура? Ниже дается примерный ответ, но не для общего случая, а только для блок-схем (потому что язык ДРАКОН строится на основе блок-схем; большинство программных чертежей методологии *RAD* — тоже блок-схемы)[2]. По нашему мнению, блок-схемы обладают хорошей структурой, если при их создании учитываются (возможно, с исключениями) следующие правила.

- ! Структура диосцены должна быть не хаотичной, а регулярной и предсказуемой.
- ! Диосцену желательно разбить на зоны, имеющие зрительно-смысловое значение[3] (зона обычно содержит несколько блоков).
- ! Назначение зон желательно разъяснить с помощью надписей, расположение которых в поле чертежа подчиняется визуальной логике картины и облегчает ее понимание.
- ! Границы зон (выделяемые пробелами или линиями) должны иметь простую прямоугольную форму.
- ! Структурные зоны, блоки и их связи желательно упорядочить по двум декартовым осям. Предварительно нужно четко определить критерии ориентации содержания диосцены по осям x и y, специально оговорив критерии и смысл движения взгляда по указанным осям как в прямом, так и в обратном направлении.

---

[1] Требование хорошей структуры имеет некоторое сходство с законом “хорошей формы” гештальтпсихологии (законом прегнантности) [2].

[2] Блок-схема по определению содержит два основных элемента: блоки (фигуры с замкнутым контуром, внутри которых помещается текст) и соединяющие их линии. Блок-схема — широкое понятие, которое охватывает схемы алгоритмов и программ, схемы декомпозиции, схемы зависимости, схемы “сущность—связь”, схемы потоков данных и т. д.

[3] Это связано с тем, что в нашем мозгу имеются специальные нейронные механизмы “для сегментации поля зрения, т. е. для разбиения его на участки, имеющие зрительно-смысловое значение” [4].

- Соединительные линии между блоками должны быть вертикальными и горизонтальными. Наклонные линии не рекомендуются.
- Желательно, чтобы входы и выходы блоков имели однозначную ориентацию. Например, если определено, что входная линия присоединяется к блоку сверху, то иное присоединение (справа, слева и снизу) следует считать не очень хорошим.
- Число пересечений, обрывов и изломов на соединительных линиях нужно минимизировать.
- Следует избегать визуальных помех, т. е. избыточных обозначений, без которых можно обойтись и которые отвлекают внимание от главного.
- Замкнутые контуры предпочтительнее, чем разорванные линии.
- Следует использовать простые и интуитивно ясные средства, позволяющие отделить смысловую фигуру (простую или составную) от фона.
- Линии контура блоков должны быть жирнее, чем соединительные линии.
- Следует использовать также некоторые правила, рекомендуемые в инженерной психологии для проектирования средств отображения информации, например правило метра и ритма [5].

Подчеркнем еще раз, что перечисленные рекомендации не претендуют на роль армейских приказов, которые следует неукоснительно выполнять во всех пунктах. Выражаясь портновским языком, это скорее стандартные выкройки, которые для каждого клиента следует подогнать по фигуре. Иначе говоря, это общие положения, однако на их основе — после учета особенностей проектируемого класса блок-схем — могут быть сформулированы конкретные правила. В последующих главах будут изложены, тщательно обоснованы и снабжены многочисленными примерами детально разработанные наборы эргономических правил, использованные при создании языка ДРАКОН.

ЗАЧЕМ НУЖНЫ ПСИХОЛОГИЧЕСКИЕ ЭКСПЕРИМЕНТЫ?

Современный подход к когнитивному проектированию языковых средств программирования состоит из шести частично перекрывающихся этапов:

- разработка теоретических положений когнитивного программирования;
- трансформация их в конкретные эргономические правила проектирования языковых средств;
- разработка желаемого языка программирования с нужными когнитивными и иными характеристиками с учетом указанных правил;
- проверка когнитивной эффективности конструкций нового языка с помощью управляемого психологического эксперимента, построенного в соответствии со строгими критериями, принятыми в экспериментальной психологии [6];
- оценка когнитивного качества нового языка методом экспертных оценок, полученных на основе использования языка в нескольких программных проектах;
- доработка языка с целью устранения когнитивных недостатков.

Цель данного параграфа — подчеркнуть особую значимость управляемого психологического эксперимента как мощного средства контроля, необходимого для улучшения когнитивного качества проектируемых языков.

Термин “психология программирования” ввел Том Лав, психолог фирмы “Дженерал Электрик”, занимавшийся методами усовершенствования производства программ [6]. Психология программирования призвана решить ряд задач, в частности, исключить случайные и субъективные факторы, обусловленные сиюминутными проектными и коммерческими соображениями и приводящие к приблизительным, качественным суждениям о том, “что людям должно нравиться” или “что проще в использовании”. Для экспериментального обоснования своих рекомендаций психология программирования предпочитает использовать точные количественные методы исследования человеческой деятельности [7].

Если вчера многие разработчики языков программирования зачастую обосновывали свою позицию с помощью аргументов типа “это удобно”, “это наглядно и понятно каждому”, “это предельно ясная и читабельная языковая конструкция”, “так проще”, “так быстрее”, “так доходчивее”, то сегодня все яснее становится крайняя неопределенность, слабость и субъективность подобных утверждений. Несостоятельность традиционных методов оценки наглядности и удобства особенно ярко проявляется в тех ситуациях, когда в ходе полемики на тему “чья нотация лучше?”, каждый из авторов, отстаивающих диаметрально противоположные точки зрения, объявляет свою систему нотаций “более наглядной” и “более удобной”.

Чтобы устранить споры, необходим корректно поставленный психологический эксперимент. В частности, чтобы выяснить, какая из двух языковых конструкций (или форм представления знаний) является более понятной, можно провести эксперимент на двух представительных выборках (группах людей, например, студентов), из которых первая группа решает задачу с помощью первой языковой конструкции или нотации, а вторая — пользуется конкурирующими структурами и обозначениями. Сравнивая объективные количественные показатели, такие, как среднее время решения задачи или среднее по группе число правильных ответов на контрольные вопросы, можно получить объективную оценку, говорящую в пользу той или иной языковой структуры, нотации, формы визуального чертежа и т. д. Использование хорошо отработанной технологии управляемых психологических экспериментов позволяет устранить субъективизм, преодолеть разногласия между разными разработчиками — соавторами языка, устранить когнитивные погрешности и за счет этого существенно улучшить когнитивное качество проектируемых языков.

К сожалению, в нашей стране психология программирования развита слабо и практически не имеет связей с кафедрами информатики, вычислительной математики и программирования вузов. По мнению автора, в качестве первого шага следует ввести курс психологии программирования или когнитивных основ программирования для студентов факультета вычислительной математики и кибернетики МГУ, а также на соответствующих кафедрах других ведущих ВУЗов, готовящих программистов.

ОШИБКА ДЖЕЙМСА МАРТИНА

Джеймс Мартин — специалист мирового класса, признанный авторитет в области компьютерных наук, автор многих книг, в том числе фундаментального руководства по методологии *RAD*. Он — один из тех, к кому прислушиваются, чье мнение во многом определяет стиль работы и практические действия многих организаций и специалистов во всем мире.

Вот почему имеет смысл обратить внимание читателя на одно спорное, чтобы не сказать ошибочное положение, которое Мартин систематически отстаивает в своих трудах на протяжении многих лет. Речь идет о рекомендации Мартина использовать для графического представления программ чертежи “нормального размера”, которые можно напечатать на обычном принтере. Некорректность связанных с этим аргументов иллюстрирует табл. 1, в левой графе которой приводятся рекомендации Мартина, а в правой — опровергающие их соображения.

“ЭТО ЧУДАКАМ-ИНЖЕНЕРАМ НУЖНЫ БОЛЬШИЕ ЧЕРТЕЖИ, А МЫ, ХИТРЕЦЫ-ПРОГРАММИСТЫ, ОБОЙДЕМСЯ МАЛЕНЬКИМИ”

По мнению автора, позиция Мартина отражает общее заблуждение, глубоко укоренившееся в умах многих специалистов по информатике и препятствующее прогрессу в деле повышения продуктивности мозга программистов.

Проследим ход рассуждений Мартина. Они весьма просты. Персональные компьютеры и дешевые принтеры получили массовое распространение и доступны всем. В отличие от них плоттеры встречаются

Таблица 1

Утверждение Мартина	**Возражение**
Чертежи программ должны быть компактными [8]	Компактность — не самоцель. К ней следует стремиться в тех случаях, когда она увеличивает производительность. Если же компактность чертежа затрудняет понимание вопроса и снижает продуктивность, разумно от нее отказаться
Чертежи программ следует выполнять на бумаге формата А4 [8]	Это верно, если проблема простая и иллюстрирующий ее рисунок целиком размещается на листе формата А4. Если же проблема сложная и требует, например, 400 форматок А4, выгодней перейти к большим форматам. В этом случае понадобится всего 50 листов формата А1 или 25 листов формата А0. Большие чертежи, нарисованные в соответствии с когнитивно-эргономическими правилами с помощью технологии *I-CASE*, снижают когнитивную нагрузку на мозг читателя и повышают умственную производительность в процессе понимания и анализа сложных программных проектов
Большие чертежи неудобны тем, что их трудно посылать другим людям или брать домой [9]	Это неверно. Во всем мире инженеры пользуются большими чертежами: их постоянно пересылают из одной организации в другую, берут для работы домой и т. д. Автор сам это делал много раз и может поручиться, что при этом не возникает никаких проблем. Чертежи форматов А1 и А0 известным способом несколько раз перегибаются и укладываются в папку для бумаг формата А4
Некоторые аналитики, программисты и администраторы данных любят рисовать цветные настенные схемы шириной шесть футов (2 м 10 см). Это можно сделать с помощью плоттера *CALCOMP*, однако такие плоттеры дороги и недоступны большинству аналитиков и программистов. Нужно отказаться от подобных громадных чертежей и ограничиться форматом, который можно напечатать с помощью обычного персонального компьютера и принтера [9]	Рассуждение некорректно, ибо Мартин рассматривает две крайние позиции: либо роскошные двухметровые цветные чертежи, сделанные на сверхдорогом плоттере, либо маленькие форматки обычного персонального компьютера. В действительности существует промежуточный вариант: малогабаритный щелевой плоттер с рапидографом, позволяющий рисовать черно-белые чертежи формата А1. Такие плоттеры относительно недороги и представляют собой разумный компромисс. Существуют и другие приемлемые варианты плоттеров

Окончание табл. 1

Утверждение Мартина	**Возражение**

Иногда чертеж требует больших размеров бумаги. В этом случае следует увеличивать не горизонтальный, а вертикальный размер чертежа, чтобы напечатать его на фальцованной (сложенной на сгибах) бумаге на обычном принтере [8]	Этот совет, как и предыдущие, не учитывает когнитивные вопросы и нарушает известную рекомендацию инженерной психологии: при создании средств отображения информации “следует учитывать особенности биомеханики глаза, в частности, то, что горизонтальные движения глаз совершаются наиболее легко и быстро. Менее быстры вертикальные движения” [5][1].

[1] В данном случае можно достичь компромисса, если с помощью программных средств повернуть компьютерный образ чертежа на 90° и вывести его на обычный принтер в графическом режиме. Это позволит превратить неудобную вертикальную “кишку” в приятный для глаза чертеж, у которого длинный размер наращивается по горизонтали. Однако такой прием неприменим к схемам действий, которые специально (и крайне неудачно) сконструированы именно в форме вертикальной кишки, что хорошо согласуется с характеристиками принтера, но плохо согласуется с характеристиками человека. Переделать их в горизонтальную форму, более удобную с точки зрения биомеханики глаза, принципиально невозможно. Таким образом, схемы действий построены исходя из технократических, а не когнитивных соображений.

сравнительно редко. Поэтому при выборе формата визуальных чертежей следует ориентироваться на имеющуюся технику. Исходя из этого, Мартин, жертвуя наглядностью ради уменьшения формата, предлагает тщательно продуманную систему мер, позволяющих сократить размеры чертежей [9], чтобы использовать “дешевые принтеры” и сэкономить на плоттерах. Хотя в каких-то частных ситуациях такой подход может оказаться разумным, однако в масштабе национальной или мировой экономики идея “дешевых принтеров” не улучшает, а наоборот, ухудшает экономические показатели информационной отрасли, так как, выигрывая по стоимости плоттеров, мы несоизмеримо проигрываем на производительности труда. Это значит, что в стратегическом плане Мартин пожертвовал не пешку за ферзя, а наоборот, ферзя за пешку.

Слабость позиции Мартина состоит в том, что он упускает из виду исключительно важный факт: *между размерами программных чертежей и производительностью умственного труда существует связь*. Эта связь хорошо понятна: при увеличении формата графической диосцены включаются в работу мощные резервы симультанного восприятия и увеличивается скорость работы мозга за счет ликвидации эффекта “частичной слепоты”.

Против Мартина работает еще один аргумент, связанный с уроками истории. Всемирная практика развития машиностроения и электротехники привела инженеров к двум выводам:

1) замена текстового описания механических и электрических устройств чертежами повышает производительность;
2) форматы чертежей в зависимости от обстоятельств целесообразно увеличить до нужных размеров.

Сегодня программисты, можно сказать, уже согласились с первым выводом и решили, что им, как и инженерам, тоже нужны чертежи. Однако второй вывод по-прежнему отвергается, поскольку миф об исключительности программистов оказывается необыкновенно живучим (см. заголовок этого параграфа). Ясно, однако, что зрительный анализатор и мозг программиста по своей конструкции не отличаются от мозга инженера. Механизмы зрительного восприятия чертежей определяются нейробиологическими и психологическими закономерностями, которые едины для программистов и инженеров. Это значит, что сказав А, нужно сказать и Б. Логично предположить, что рано или поздно в программировании будет узаконен тот же набор форматов чертежей, что и в инженерном деле. Чем скорее это будет сделано, тем лучше.

Чтобы реализовать идею на практике, нужно выполнить три условия. Во-первых, оборудование рабочих мест системных аналитиков и программистов должно содержать не только компьютер и принтер, но и плоттер (коллективного или — по мере удешевления — индивидуального пользования). Речь идет не о единичных случаях, о массовой доукомплектации рабочих мест, чтобы плоттеры стали доступны всем работникам, для которых их использование оправдано с точки зрения производительности, экономических и когнитивных факторов (за вычетом неизбежного периода больших начальных затрат). Во-вторых, инструментальные средства создания программ должны быть снабжены дополнительными интерфейсами с выходом на драйверы плоттеров (которые сегодня отсутствуют). В-третьих, нужна хорошо продуманная система переобучения.

Практическое внедрение идеи целесообразно начать с больших организаций, где разрабатываются и аппаратура, и программы. В таких организациях уже имеются плоттеры, однако сегодня их активно используют инженеры, но не программисты. Последние в большинстве случав даже не догадываются, что могут получить значительный выигрыш от энергичной работы с плоттерами. В связи с этим необходимо создать атмосферу “общественного беспокойства”, стимулирующую осознание полезности больших форматов и плоттеров для роста производительности.

Реализация намеченного плана неизбежно столкнется с немалыми трудностями. Однако они преодолимы и будут уменьшаться по мере приобретения навыков, удешевления техники и увеличения размеров экранов компьютеров. Ясно одно: господствующую сегодня точку зрения, согласно которой плоттеры нужны инженерам и не нужны программистам, следует признать порочной и неприемлемой, поскольку эффективность программирования играет существенную роль в национальной экономике. Даже незначительное в процентном отношении повышение производительности труда в этой сфере деятельности может привести к серьезной экономии.

ВОЗМОЖНА ЛИ СТРАТЕГИЧЕСКАЯ РЕФОРМА МИРОВОЙ ПРАКТИКИ ПРОГРАММИРОВАНИЯ

Согласно исследованиям американского специалиста Джона Мусы за двадцать лет между 1965 и 1985 гг. потребность в программном обеспечении увеличилась в сто раз, однако производительность программистов выросла лишь в два раза. А. Питер и Т. Рэймонд характеризуют этот факт, как "кризис производительности" [10].

Барри Боэм пишет: "Основной причиной того, что повышение производительности разработки ПО (программного обеспечения) стало острой проблемой, является увеличивающийся спрос на новое ПО, не согласующийся с имеющимися возможностями традиционных подходов" [11]. Налицо драматический разрыв между достигнутым уровнем производительности и лавинообразным ростом потребностей, который отражает объективную тенденцию, связанную с непрерывным возрастанием роли программных средств в экономике любой развитой страны. Главное противоречие современности — между мощными процессорами и громоздкими и полными ошибок программами. Именно отсюда, со стороны программного обеспечения может прийти новая революция, и тогда монотонное и пресное течение компьютерных дел прервется новыми неординарными событиями. Но вряд ли это случится в ближайшее время.

По нашему мнению, последнюю оценку следует уточнить: новая революция в производительности труда произойдет тогда, когда при проектировании следующего поколения компьютерных методологий и средств создания программ будет *повсеместно осознана и поставлена во главу угла приоритетная роль когнитивного фактора*. На практике это означает, что в битве за производительность необходимо перевести из стратегического резерва в действующую армию все три вида "когнитивного оружия":

1) переход от текстового представления знаний и текстового программирования к визуальному;
2) увеличение формата диосцен и диопрограмм до оптимальных размеров, позволяющих устранить эффект "частичной слепоты";
3) научно обоснованное увеличение наглядности визуальных форм представления знаний и визуальных программ и проектирование структуры диосцен на основе признанной теории и набора когнитивных правил, уточненных в ходе управляемого психологического эксперимента.

Эти три пункта можно охарактеризовать как предварительный план стратегической реформы мировой практики программирования, который предлагается для обсуждения и критики. Остальную часть книги можно рассматривать как попытку обосновать этот план.

ВЫВОДЫ

1. С точки зрения нейробиологии и психологии, информационные сообщения, записанные с помощью нотаций письменного языка и предъ-

являемые человеку на бумаге или экране, поступают в высшие отделы мозга через зрительный анализатор. Следовательно, указанные сообщения есть не что иное как оптический код, представленный в виде одной или нескольких диосцен. Этот код имеет оптический синтаксис и оптическую семантику.

2. Между оптическими характеристиками диосцен и продуктивностью мозга есть связь: изменяя конструкцию и характеристики диосцен (передающих заданное смысловое сообщение), можно увеличить продуктивность мозга.

3. Чтобы улучшить работу ума, оптический синтаксис и семантику диосцен следует проектировать так, чтобы характеристики диосцен и характеристики мозга были согласованы между собой.

4. Новый когнитивный подход — это попытка подготовить теоретическую платформу для разработки некоторых методов повышения интеллектуальной продуктивности человеческого мозга и построения языков следующего поколения, удовлетворяющих критерию улучшения работы ума и сверхвысокого понимания. Предполагается, что это создаст предпосылки для стратегической реформы мировой практики программирования.

Глава 6
ИЗЮМИНКИ ЯЗЫКА ДРАКОН

Графический язык является главным средством достижения наглядности

Константин Гомоюнов

КРИТИКА БЛОК-СХЕМ

Эффективным средством для улучшения понимаемости алгоритмов является визуализация программирования [1], а несколько раньше для этой цели использовались блок-схемы [2]. Однако в последнее время блок-схемы подвергаются критике. Противники блок-схем утверждают, что они непригодны для структурного программирования, не поддаются формализации, поэтому их "нельзя использовать как программу для непосредственного ввода в машину" [3]. Они занимают много страниц, причем "в клеточки блок-схем можно вписывать весьма ограниченные сведения". Блок-схемы "затрудняют обучение и снижают производительность при понимании" [4]. Кроме того, они удобны не для всех — работу с блок-схемами предпочитают только "индивидуумы с правым ведущим полушарием, ориентированные на визуальную информацию, интуитивные, распознающие образы", однако их избегают "индивидуумы с левым ведущим полушарием, ориентированные на словесную информацию, склонные к дедуктивным рассуждениям" [4] и т. д.

Если до 1980 г. блок-схемы были наиболее широко применяемым средством, то сегодня они "больше не считаются необходимыми и их популярность падает". Хотя имеются отдельные попытки приспособить блок-схемы к современным нуждам (язык *SDL* и др.), однако в целом блок-схемы явно оказались на обочине бурно развивающегося процесса визуализации программирования, а их громадные потенциальные возможности фактически не используются. Язык ДРАКОН позволяет устранить или существенно ослабить отмеченные недостатки блок-схем.

Для обозначения блок-схем, построенных по правилам языка ДРАКОН, используется термин "дракон-схемы".

ПРЕИМУЩЕСТВА ДРАКОН-СХЕМ

Чем же отличаются дракон-схемы от блок-схем? *Блок-схемы* не обеспечивают автоматическое преобразование алгоритма в машинный код. *Дракон-схемы*, напротив, пригодны для формализованной записи, автоматического получения кода и исполнения его на компьютере. Однако более важным является второе (когнитивное) отличие. Хотя блок-схемы порою действительно улучшают понимаемость программ, однако это происходит не всегда, причем степень улучшения невелика. Кроме того, есть немало случаев, когда неудачно выполненные блок-схемы запутывают дело и затрудняют понимание. В отличие

от них дракон-схемы удовлетворяют критерию сверхвысокой понимаемости.

Благодаря использованию специальных формальных и неформальных когнитивных приемов дракон-схемы дают возможность изобразить решение любой, сколь угодно сложной технологической проблемы в предельно ясной, наглядной и доходчивой форме, которая позволяет значительно сократить интеллектуальные усилия персонала, необходимые для зрительного восприятия, понимания, верификации и безошибочного решения проблем.

ИКОНЫ И МАКРОИКОНЫ

Графоэлементы (графические буквы) языка ДРАКОН называются *иконами* (рис. 1). Подобно тому, как буквы объединяются в слова, иконы объединяются в составные иконы — *макроиконы* (рис. 2).

Соединяя иконы и макроиконы по определенным правилам, можно строить разнообразные алгоритмы, примеры которых показаны на рис. 3, 4, 6, 8—11.

Шампур-блок — часть дракон-схемы, имеющая один вход сверху и один выход снизу, расположенные на одной вертикали. Примерами шампур-блоков являются иконы И3 — И10, И12 — И16, И18, И20, И21 (рис. 1) и макроиконы 2—20 (рис. 2).

ЗАЧЕМ НУЖНА ВЕТКА?

Когда принцесса Анна развелась с маркизом Ле-Шателье, возник спор о разделе имущества. Судья потребовал указать, какие покупки принцесса сделала до замужества, а какие — после.

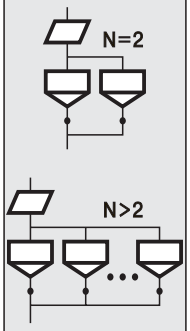
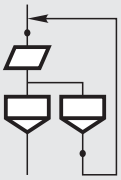
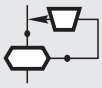
А теперь забудем об этой семейной драме и сравним между собой рис. 3*а* и 3*б*. Легко видеть, что первый не позволяет ответить на вопрос судьи. Что касается второго, то он, наоборот, содержит нужную информацию. Более того, алгоритм на рис. 3*б* нарочно нарисован так, что покупки, сделанные до и после замужества, четко делятся на два списка. Эти списки зрительно и пространственно разнесены, поэтому деление алгоритма на две части независимо от воли читателя буквально бросается в глаза. Такой прием называется разбиением алгоритма на смысловые части по принципу “взглянул — и сразу стало ясно!” А сами смысловые блоки именуются *ветками*.

Слово “ветка” имеет два значения. С одной стороны, это смысловой “кусок” алгоритма. Например, алгоритм на рис. 3*б* имеет две ветки

	Икона	Название иконы
И1		Заголовок
И2		Конец
И3		Действие
И4		Вопрос
И5		Выбор
И6		Вариант
И7		Имя ветки
И8		Адрес
И9		Вставка
И10		Полка
И11		Формальные параметры
И12		Начало цикла ДЛЯ
И13		Конец цикла ДЛЯ

	Икона	Название иконы
И14		Вывод
И15		Ввод
И16		Пауза
И17		Период
И18		Пуск таймера
И19		Синхронизатор (по таймеру)
И20		Параллельный процесс
И21		Комментарий
И22		Правый комментарий
И23		Левый комментарий
И24		Петля цикла
И25		Петля силуэта

Рис. 1. **Иконы языка ДРАКОН**

	Макроикона	Название макроиконы
1		Заголовок с параметрами
2		Развилка
3		Переключатель (число вариантов N⩾2)
4		Обычный цикл
5		Переключающий цикл
6		Цикл ДЛЯ
7		Цикл ЖДАТЬ
8		Действие по таймеру
9		Полка по таймеру

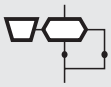
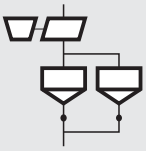
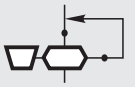
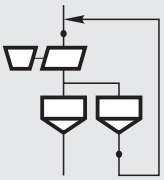
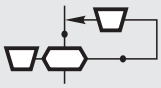
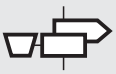
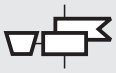
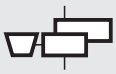
	Макроикона	Название макроиконы
10		Развилка по таймеру
11		Переключатель по таймеру
12		Обычный цикл по таймеру
13		Переключающий цикл по таймеру
14		Цикл ДЛЯ по таймеру
15		Цикл ЖДАТЬ по таймеру
16		Вставка по таймеру
17		Вывод по таймеру
18		Ввод по таймеру
19		Пуск таймера по таймеру
20		Параллельный процесс по таймеру

Рис. 2. **Макроиконы языка ДРАКОН**

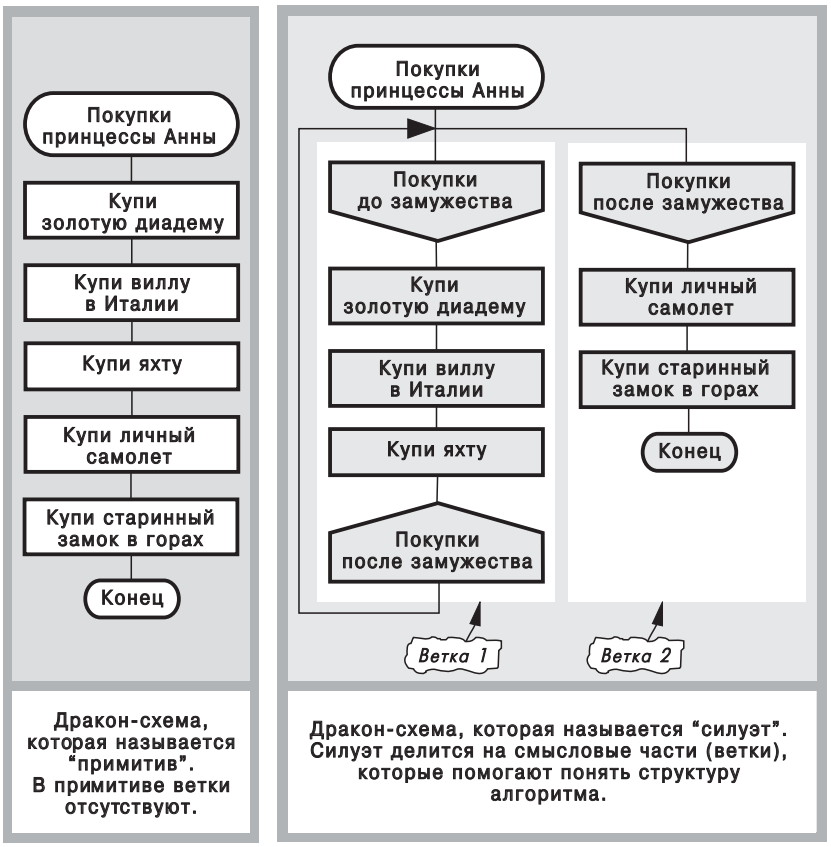


Рис. 3. **Чем отличаются примитив и силуэт?**

("Покупки до замужества" и "Покупки после замужества"). На рис. 4 — четыре ветки ("Подготовка к ловле", "Ожидание клева", "Рыбацкая работа", "Обратная дорога"). С другой стороны, ветка — составной оператор языка ДРАКОН, который не имеет аналогов в известных языках. Оператор "ветка" состоит из трех частей: *начала ветки* (икона "имя ветки"), *тела ветки* (которое может содержать большое число икон) и *конца ветки* (который содержит одну или несколько икон "адрес" либо икону "конец").

Итак, зачем нужна ветка? Чтобы помочь работнику умственного труда, программисту и разработчику технологии формализовать смысловое разбиение проблемы, программы или техпроцесса на части и дать частям удобные смысловые названия. При этом разделение проблемы на N смысловых частей реализуется путем разбиения алгоритма на N веток.

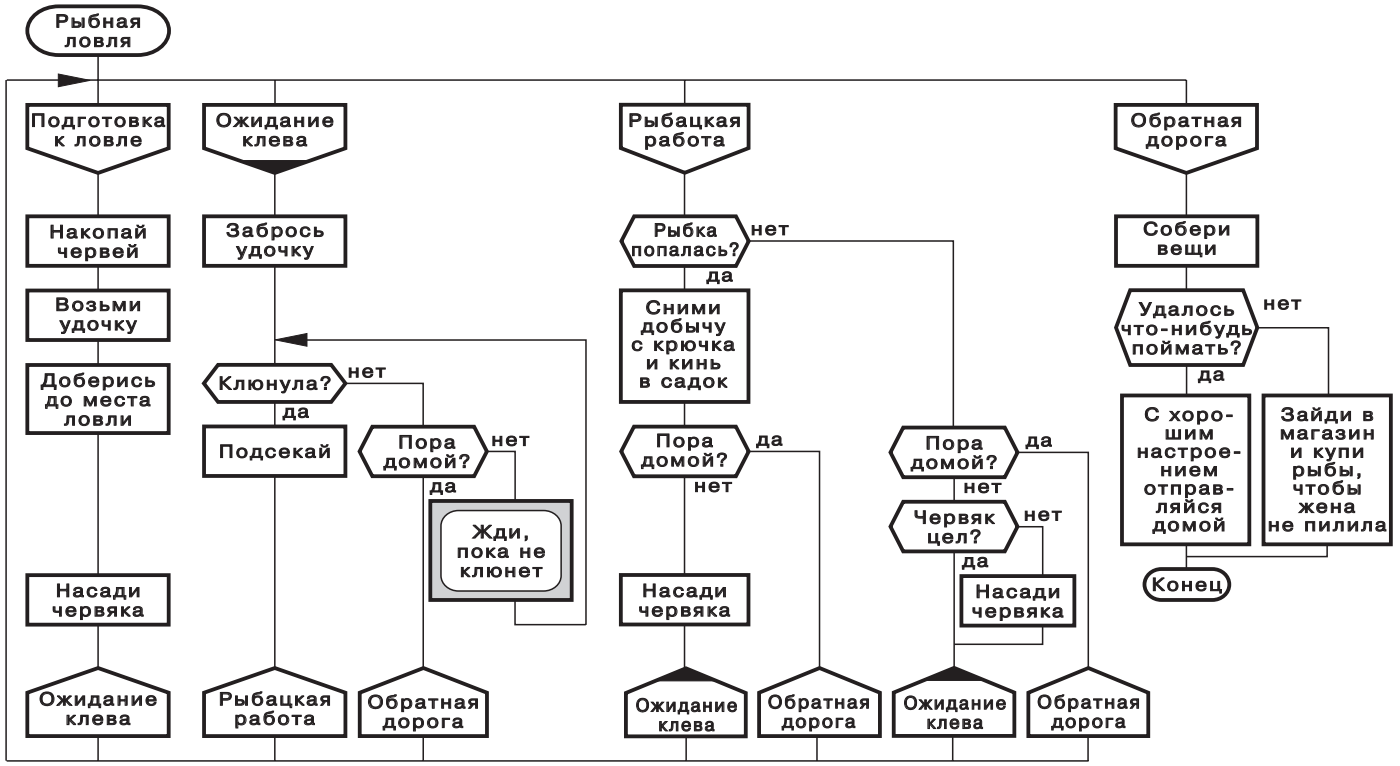


Рис. 4. **Алгоритм “Рыбная ловля”**

КАК РАБОТАЕТ ВЕТКА?

Ветка имеет один вход и один или несколько выходов. *Входом* служит икона “имя ветки”, содержащая идентификатор ветки. Визуальный оператор “имя ветки” не выполняет никаких действий, это всего лишь метка, объявляющая название смысловой части программы. Исполнение дракон-алгоритма всегда начинается с крайней левой ветки (рис. 3, 4).

Выходом из ветки служит икона “адрес”, в которой записывается имя следующей по порядку исполнения ветки. Икона “адрес” — это замаскированный оператор перехода *(goto)*, однако он передает управление не куда угодно, а только на начало выбранной ветки. Вход в ветку возможен только через ее начало. Выход из последней ветки осуществляется через икону “конец”.

КАК СЛЕДУЕТ РАСПОЛАГАТЬ ВЕТКИ В ПОЛЕ ЧЕРТЕЖА?

Ветки упорядочены двояко: логически и пространственно. *Логическая* последовательность исполнения веток определяется метками, записанными в иконах “адрес”. Однако логический порядок — это еще не все. На рис. 5 показаны три разных способа *пространственного* расположения веток, которые имеют один и тот же логический порядок. Чтобы устранить пространственную неоднозначность и облегчить понимание смысла дракон-схемы, вводится правило “чем правее — тем позже”. Оно означает: ветка, нарисованная правее, работает позже всех веток, находящихся левее.

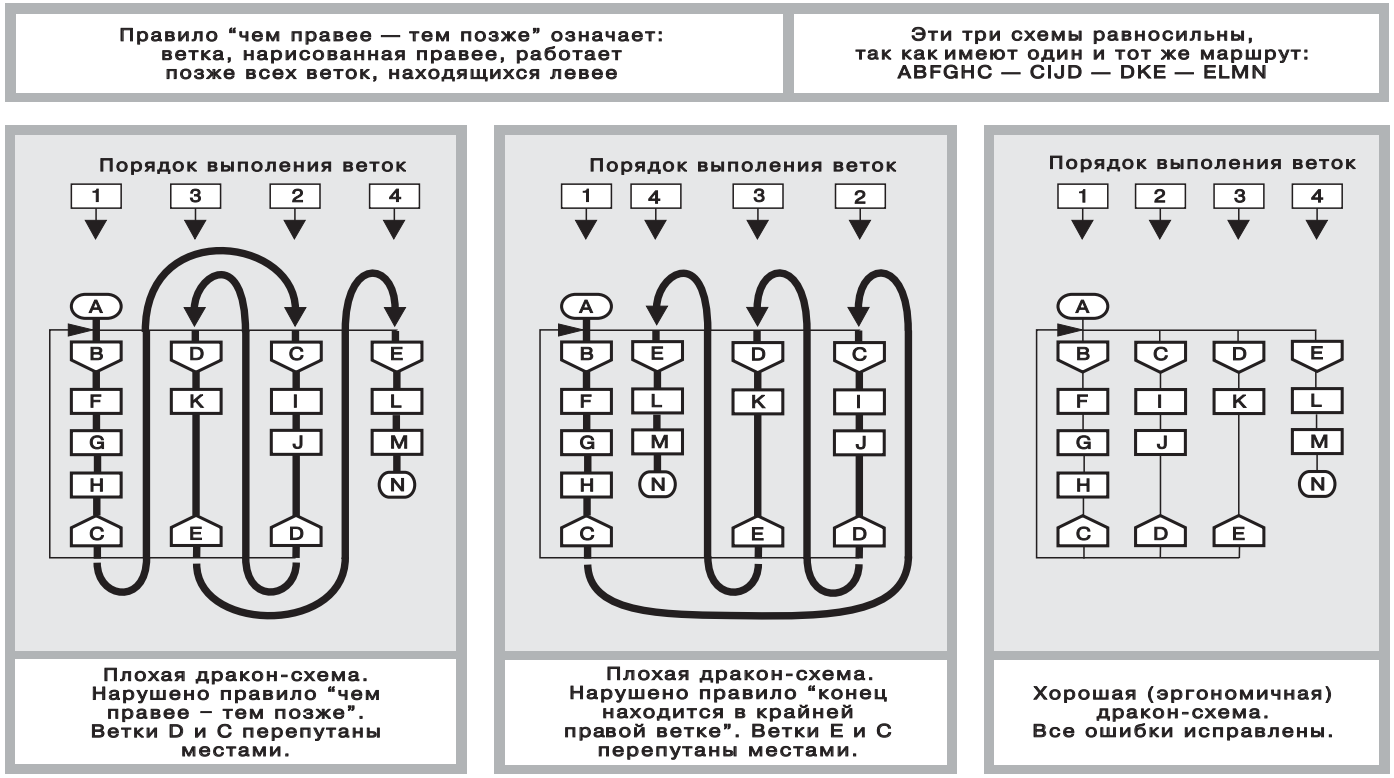
Алгоритм, нарисованный согласно правилу “чем правее — тем позже”, считается хорошим, эргономичным (рис. 5*в*). Схемы, где это правило нарушается, объявляются плохими (рис. 5*а*,*б*), их использование запрещено.

В разрешенных (эргономичных) алгоритмах имеет место следующий порядок работы (рис. 3, 4, 5*в*, 6*а*):

- первой работает крайняя левая ветка, последней — крайняя правая;
- остальные ветки передают управление друг другу слева направо (при этом может случиться так, что некоторые ветки будут пропущены);
- иногда образуется так называемый “веточный цикл”. Это происходит, когда в иконе “адрес” указано имя собственной или одной из левых веток. На рис. 4 и 6*а* веточный цикл помечен черными треугольниками.

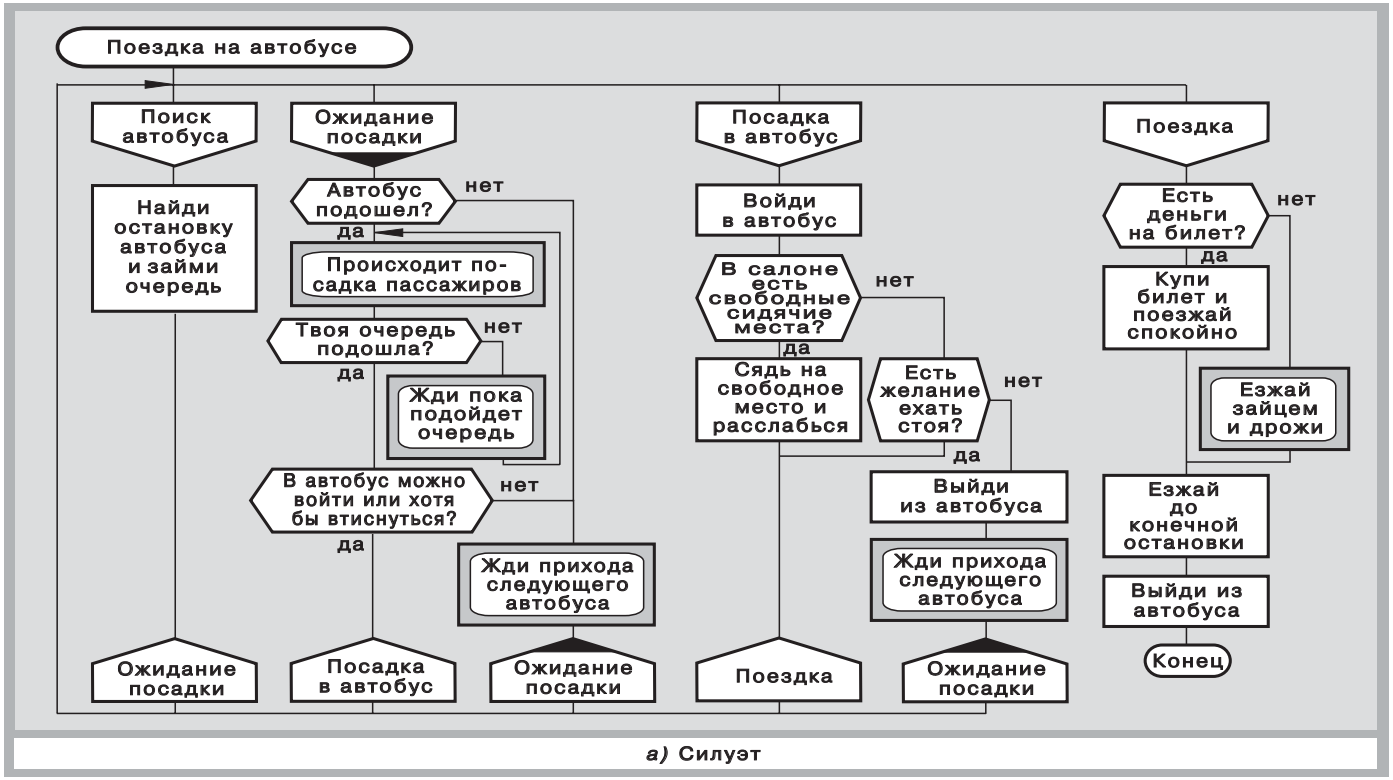
ЧТО ТАКОЕ ШАПКА?

С точки зрения читателя, любой незнакомый или забытый нетривиальный алгоритм — чрезвычайно сложная проблема, которую он отчаянно пытается понять, преодолевая мощное “сопротивление материала”. Чтобы упростить дело и облегчить задачу понимания, нужно, чтобы



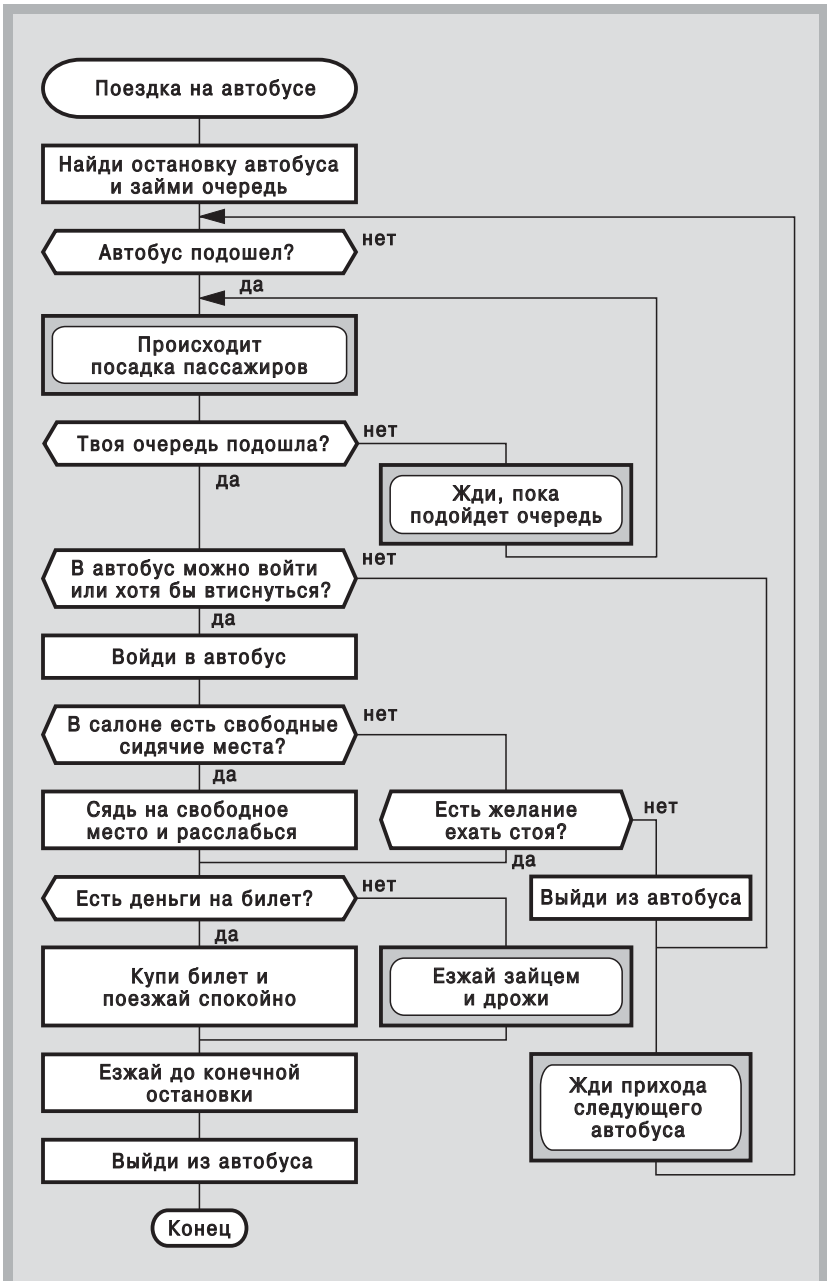
а) *б)* *в)*

Рис. 5. **Правило “чем правее — тем позже”**



а) Силуэт

Рис. 6. **Алгоритм поездки на автобусе**



б) Примитив

Рис. 6 (окончание)

читатель “прозрел” и, расчленив проблему на части, увидел ее смысловую структуру в терминах предметной области. Причем увидел не в фигуральном смысле слова, не с помощью воображения, не духовным оком, а своими двумя глазами — на бумаге или экране.

Но как это сделать? Трудность в том, что ни один из существующих языков не предоставляет читателю, изучающему сложную программу или технологию, эффективной помощи, позволяющей моментально (за несколько секунд) уяснить ее структуру, т. е. деление на смысловые блоки. В языке ДРАКОН имеются специальные средства, обеспечивающие решение задачи.

Шапкой называется верхняя часть дракон-схемы (рис. 4), которая включает заголовок алгоритма и комплект икон “имя ветки”. Назначение шапки — помочь читателю мгновенно (не более чем за несколько секунд) сориентироваться в проблеме и получить мощную подсказку — ответ на три наиболее важных вопроса:

1) как называется проблема?
2) из скольких частей она состоит?
3) как называется каждая часть?

Ведь именно с этих вопросов начинается наше знакомство с любой задачей при рациональном подходе к делу.

Вот ответы для рис. 4.

! *Как называется проблема?* Рыбная ловля.
! *Из скольких частей состоит проблема?* Из четырех.
! *Как называется каждая часть?* 1. Подготовка к ловле. 2. Ожидание клева. 3. Рыбацкая работа. 4. Обратная дорога.

Дополнительные удобства связаны с тем, что шапка занимает “парадное” место в поле чертежа, а названия смысловых частей помещаются внутри особых рамок уникальной формы и благодаря этому моментально приковывают к себе внимание читателя без всяких усилий с его стороны.

Таким образом, ДРАКОН предоставляет читателю эффективный трехэтапный метод познания незнакомой или забытой проблемы. На первом этапе, анализируя шапку, читатель узнает назначение алгоритма и его деление на смысловые части (ветки). На втором — осуществляется углубленный анализ каждой ветки. На третьем производится разбор взаимодействия веток.

ЧТО ЛУЧШЕ: ПРИМИТИВ ИЛИ СИЛУЭТ?

Дракон-схема с ветками называется *силуэтом*, без веток — *примитивом*. Силуэт, представленный на рис. 6*а*, можно изобразить в виде примитива (рис. 6*б*). Примитив есть последовательное соединение иконы “заголовок” шампур-блоков и иконы “конец”. У примитива иконы “заголовок” и “конец” обязательно лежат на одной вертикали, которая называется *шампуром*. На этой же линии лежат главные вертикали шампур-блоков. Образно говоря, шампур пронизывает иконы примитива (возможно, не все) подобно тому, как настоящий шампур пронизывает кусочки шашлыка.

Примитив рекомендуется использовать, если дракон-схема очень простая (примитивная) и содержит не более 5...15 икон. В противном

случае, чтобы улучшить читаемость программы, выгоднее использовать силуэт. Нарушение этого правила обычно чревато неприятностями, ибо мешает читателю выявить сущность решаемой проблемы и, следовательно, затрудняет и замедляет понимание смысла программы.

Например, алгоритм на рис. 6*б* выглядит громоздким и неоправданно сложным для восприятия. Это вызвано тем, что он содержит 19 икон, однако изображен в виде примитива. Криминал в том, что схема на рис. 6*б* не позволяет читателю мгновенно (за несколько секунд) распознать визуально-смысловую структуру алгоритма. В самом деле, из скольких частей состоит решаемая проблема? Глядя на рис. 6*б*, ответить на этот вопрос довольно трудно, а быстро ответить — невозможно. Положение в корне меняется, когда мы смотрим на рис. 6*а*, где тот же самый алгоритм изображен в виде силуэта. Тут, как говорится, и ежу ясно: алгоритм состоит из четырех частей: “поиск автобуса”, “ожидание посадки”, “посадка в автобус”, “поездка”. Однако это не все: не менее важно и то обстоятельство, что запутанность зрительного рисунка исчезла и схема приобрела новое эстетическое (эргономическое) качество: элегантность, ясность и прозрачность.

Таким образом, в сложных случаях силуэт позволяет существенно уменьшить интеллектуальные усилия, затрачиваемые на понимание алгоритма. В силуэте крупные структурные части программы (ветки) четко выделены, они пространственно разнесены в поле чертежа, образуя вместе с тем легко узнаваемый, стабильный, предсказуемый и целостный зрительный образ. А в примитиве структурные части не выделены и перемешаны (“всё в одной куче”), что затрудняет чтение и анализ сложных алгоритмов. Однако для простых случаев (менее 5...15 икон) примитив, как правило, оказывается более предпочтительным.

КАК ОПИСАТЬ СИЛУЭТ С ПОМОЩЬЮ ТЕКСТОВОГО ЯЗЫКА?

Из рис. 7 видно, что для описания веток в текстовый язык пришлось внести ряд изменений. В частности, появились два новых текстовых оператора, отсутствующие в традиционных языках:

ВЕТКА < идентификатор ветки >
АДРЕС < идентификатор ветки >

Оператор текстового языка ВЕТКА объявляет название ветки (записываемое на визуальном языке внутри иконы “имя ветки”). Оператор АДРЕС безусловно передает управление на текстовый оператор ВЕТКА, имя которой записано справа от оператора АДРЕС.

```
АЛГОРИТМ Поездка на автобусе
ВЕТКА    Поиск автобуса
         ВЫПОЛНИТЬ Найди остановку автобуса и займи очередь
         АДРЕС Ожидание посадки
КОНЕЦ ВЕТКИ
ВЕТКА Ожидание посадки
         ЕСЛИ Автобус подошел ? = ДА
                  ЦИКЛ ЖДАТЬ
                           КОММЕНТАРИЙ Происходит посадка пассажиров
                           ЕСЛИ  Твоя очередь подошла ? = НЕТ
                                    КОММЕНТАРИЙ Жди, пока подойдет очередь
                  КОНЕЦ ЦИКЛА
                  ЕСЛИ  В автобус можно войти ? = ДА
                           АДРЕС Посадка в автобус
         M1:      ИНАЧЕ КОММЕНТАРИЙ Жди прихода следующего автобуса
                           АДРЕС Ожидание посадки
                  КОНЕЦ ЕСЛИ
         ИНАЧЕ ПЕРЕХОД НА  M1
         КОНЕЦ ЕСЛИ
КОНЕЦ ВЕТКИ
ВЕТКА Посадка в автобус
         ВЫПОЛНИТЬ  Войди в автобус

         и так далее
```

Рис. 7. **Текстовый язык, соответствующий визуальному языку на рис. 6*а* (описаны только две ветки из четырех)**

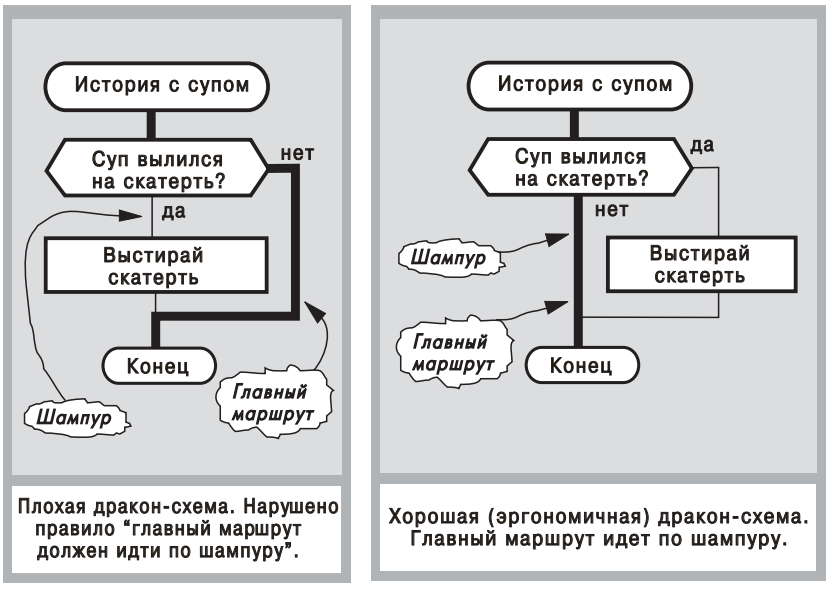


Рис. 8. **Как пустить главный маршрут по шампуру? Поменяйте местами плечи у развилки**

Сравнивая два языка: визуальный и текстовый, можно заметить, что соответствующие алгоритмы (рис. 6*а* и 7) эквивалентны [1]. Однако визуальный язык несомненно более нагляден и доходчив. Второе преимущество состоит в том, что графика позволяет полностью исключить избыточные (паразитные) элементы, каковыми в текстовом языке оказываются почти все ключевые слова: АЛГОРИТМ, ВЕТКА, АДРЕС, КОНЕЦ ВЕТКИ, ЕСЛИ, ТО, ИНАЧЕ, КОНЕЦ ЕСЛИ, ЦИКЛ ЖДАТЬ, КОНЕЦ ЦИКЛА, КОММЕНТАРИЙ, ПЕРЕХОД НА, а также метки.

ЕСТЬ ЛИ В АЛГОРИТМЕ "ЦАРСКАЯ ДОРОГА"?

Рассмотрим задачу. В запутанном лабиринте, соединяющем начало и конец сложного алгоритма, нужно выделить один-единственный маршрут — "путеводную нить", с которой можно зрительно сравнивать все прочие маршруты, чтобы легко сориентироваться в проблеме и не заблудиться в путанице развилок. Эта путеводная нить (назовем ее "главный маршрут") должна быть визуально легко различимой. Иными словами, бросив беглый взгляд на дракон-схему, мы должны обнаружить четкие ориентиры, благодаря которым можно сразу и безошибочно увидеть "царский" маршрут и упорядоченные относительно него остальные маршруты.

Для этого вводится правило: *"главный маршрут примитива должен идти по шампуру"*. Меняя местами слова "да" и "нет" в развилках и варианты в переключателях (а также присоединенные к ним гирлянды икон), следует добиться, чтобы на царском пути оказался тот выход развилки или переключателя, который ведет к наибольшему успеху (рис. 8). А побочные маршруты нужно расположить по правилу: *"чем правее — тем хуже"* (рис. 9). Если эти правила нарушены, дракон-схема считается плохой (рис. 10*а*). Однако ее всегда можно превратить в хорошую (рис. 10*б*).

В тех случаях, когда признак "лучше—хуже" не работает, вместо него следует выбрать какой-либо другой разумный критерий, чтобы смещение вправо от главного маршрута всегда было не произвольным и хаотичным, а продуманным и упорядоченным. Например, при решении математических задач выходы развилки и варианты переключателя можно расположить слева направо в порядке увеличения или уменьшения математической величины (характеристики), соответствующей этим выходам (рис. 11, 12).

---

[1] Два алгоритма называются *эквивалентными*, если они дают одинаковые результаты для одних и тех же исходных данных.

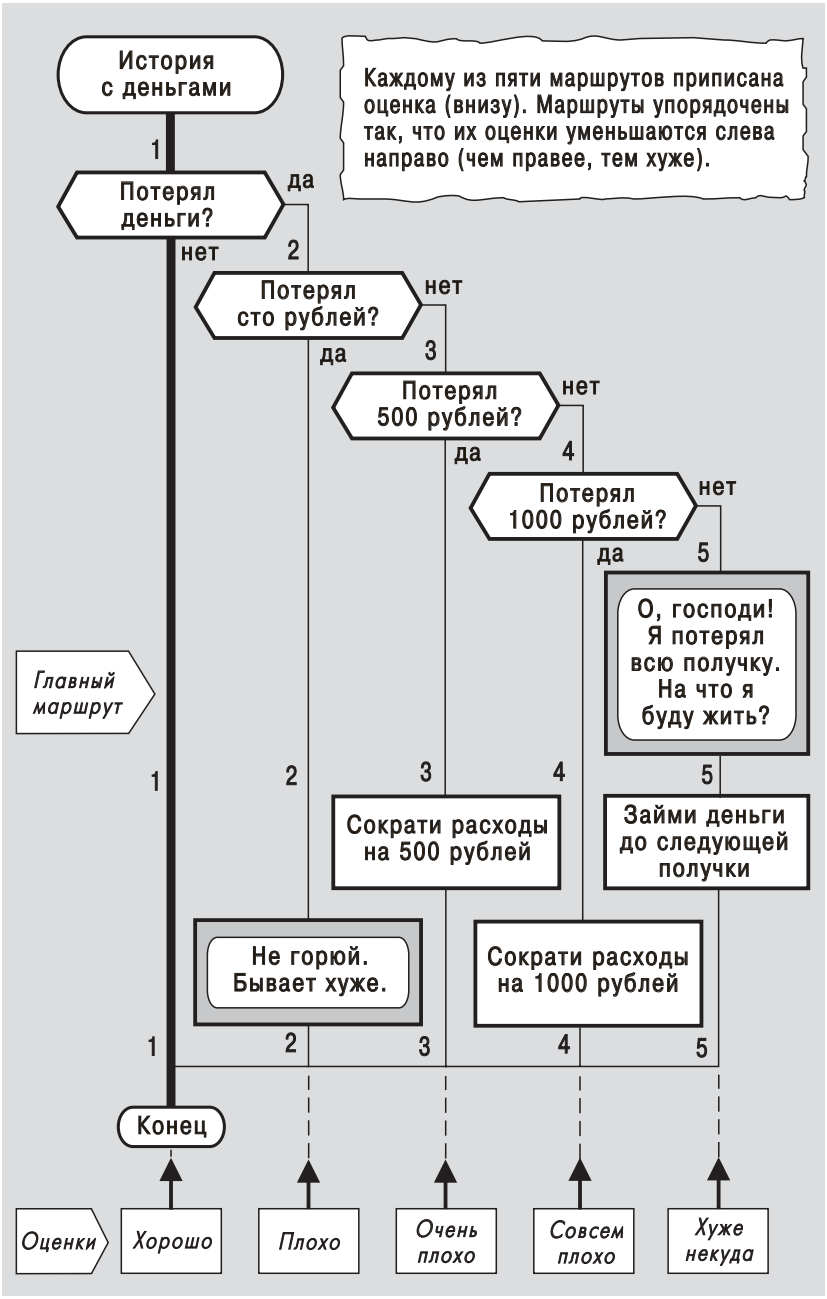


Рис. 9. **Побочные маршруты расположены по правилу "чем правее — тем хуже"**

ГЛАВНЫЙ МАРШРУТ СИЛУЭТА

В предыдущем параграфе мы узнали, как упорядочить маршруты примитива. Теперь настала очередь силуэта.

Шампуром ветки называется вертикаль, соединяющая икону “имя ветки” с иконой “адрес”, а если у ветки несколько выходов — с левым из них. Для ветки сохраняют силу оба “царских” правила:

! главный маршрут ветки должен идти по шампуру;
! побочные маршруты ветки следует упорядочить слева направо по какому-либо критерию.

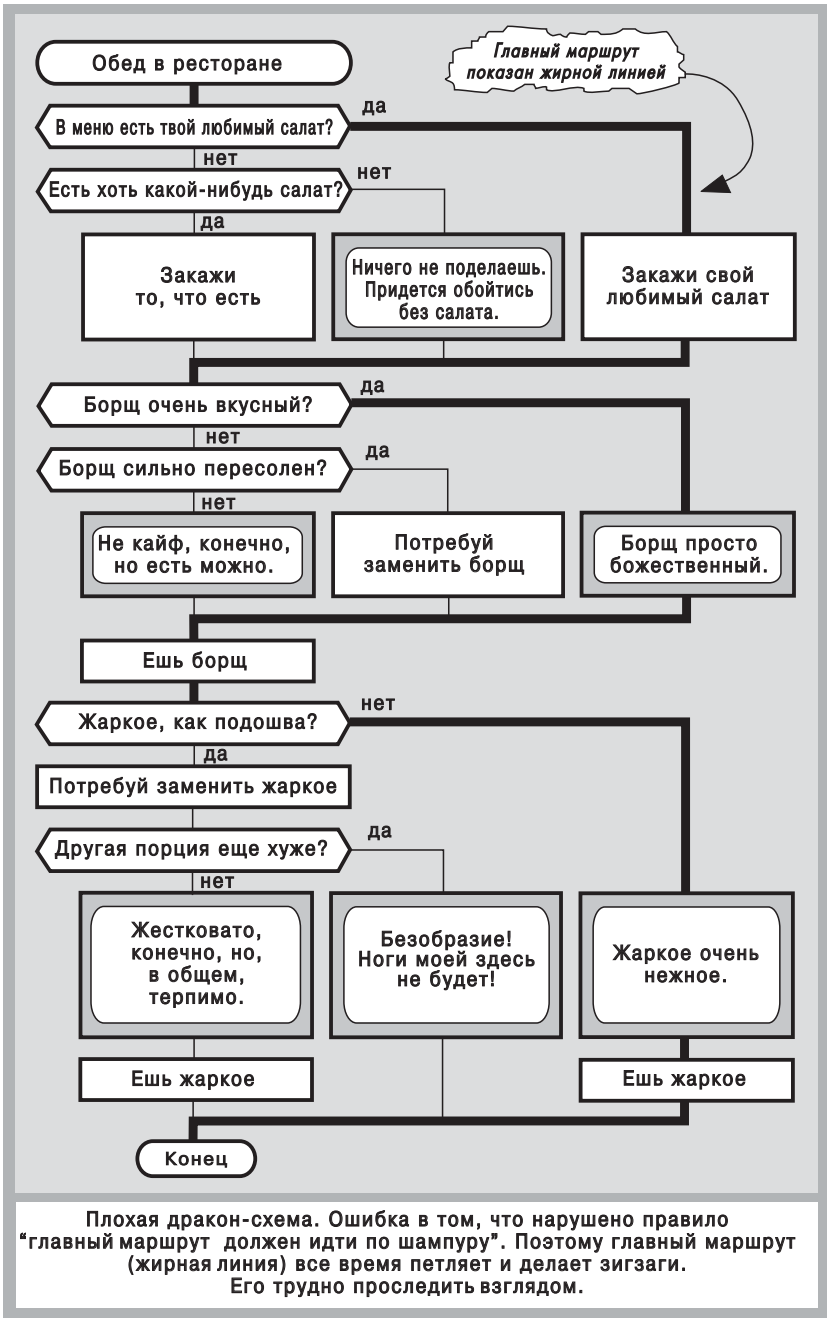
Предположим, в качестве критерия выбран принцип “чем правее — тем хуже”. В этом случае каждая ветка силуэта должна быть построена по единому правилу: чем правее (чем дальше от шампура данной ветки) расположена очередная вертикаль, тем менее успешные действия она выполняет.

Например, на рис. 6*а* ветка “посадка в автобус” имеет три вертикали. Левая вертикаль (главный маршрут) описывает наибольший успех, так как вы будете ехать в автобусе сидя. Правая вертикаль означает наименьший успех, поскольку вы вышли из автобуса и поездка откладывается. Средняя вертикаль (расположенная выше иконы “Есть желание ехать стоя?”) занимает промежуточное положение, потому что — в зависимости от ответа — может иметь место либо частичный успех (вы будете ехать, но не сидя, а стоя), либо неудача, поскольку вы выходите из автобуса несолоно хлебавши.

Главный маршрут силуэта — последовательное соединение главных маршрутов поочередно работающих веток. Таким образом, ДРАКОН позволяет читателю моментально увидеть главный маршрут любого, сколь угодно сложного и разветвленного алгоритма и, сверх того, делает смещение всех побочных маршрутов относительно “царского” не случайным, а осмысленным и предсказуемым, т. е. легким для восприятия.

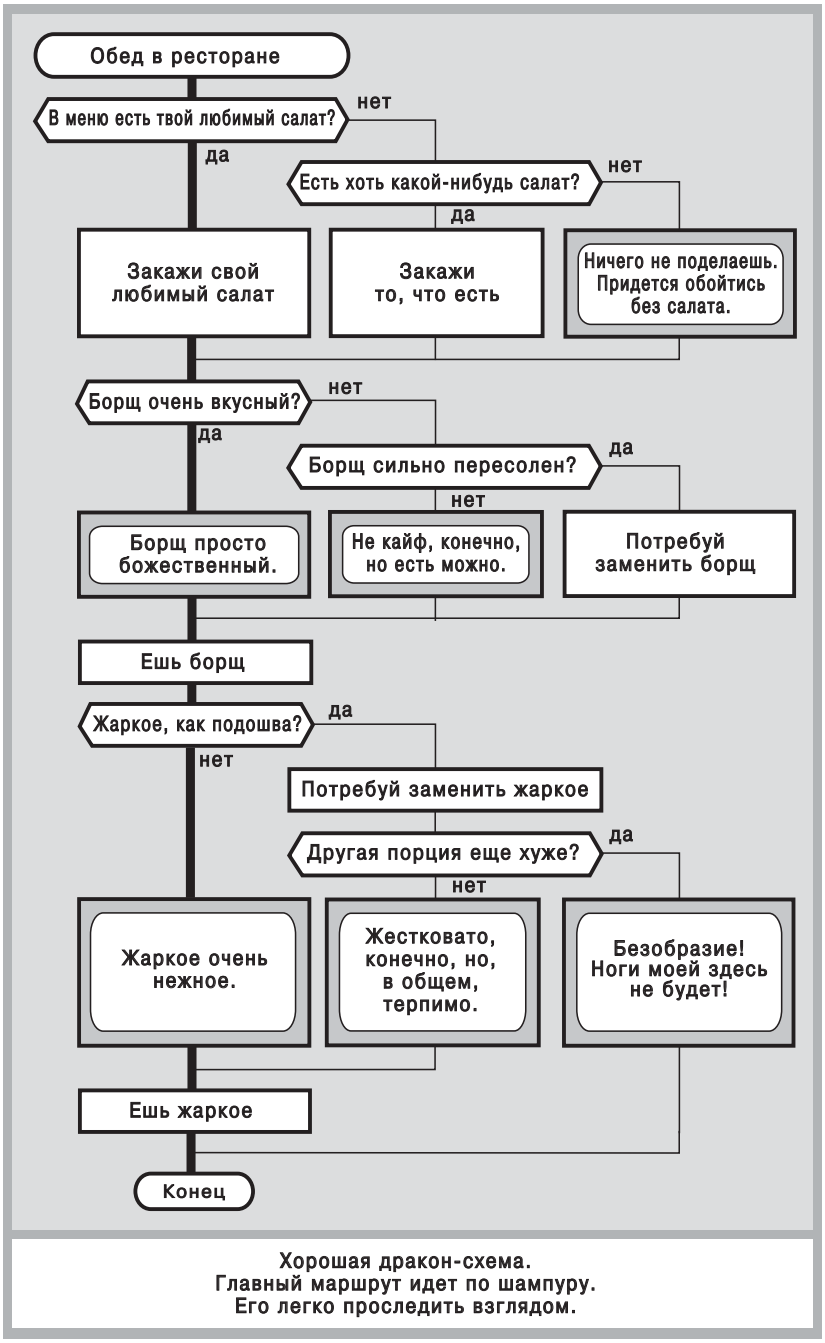
ПЕРЕСЕЧЕНИЯ ЛИНИЙ? — БОЖЕ УПАСИ!

Некоторые специалисты, склонные к резким выражениям, называют традиционные блок-схемы алгоритмов [2] “помоечными блок-схемами”, потому что изображенные на них хитросплетения блоков, соединенные хаосом куда угодно гуляющих рваных линий больше напоминают кучу мусора, нежели регулярную структуру. ДРАКОН выгодно отличается тем, что его графический узор имеет строгое математическое и когнитивно-эргономическое обоснование и подчиняется жестким и тщательно продуманным правилам. Среди них особое место занимает правило: *“пересечения и обрывы соединительных линий запрещены”*.



а)

Рис. 10. **Плохую дракон-схему** ***(а)*** **всегда можно преобразовать в хорошую** ***(б)***



б)

Рис. 10 (окончание)

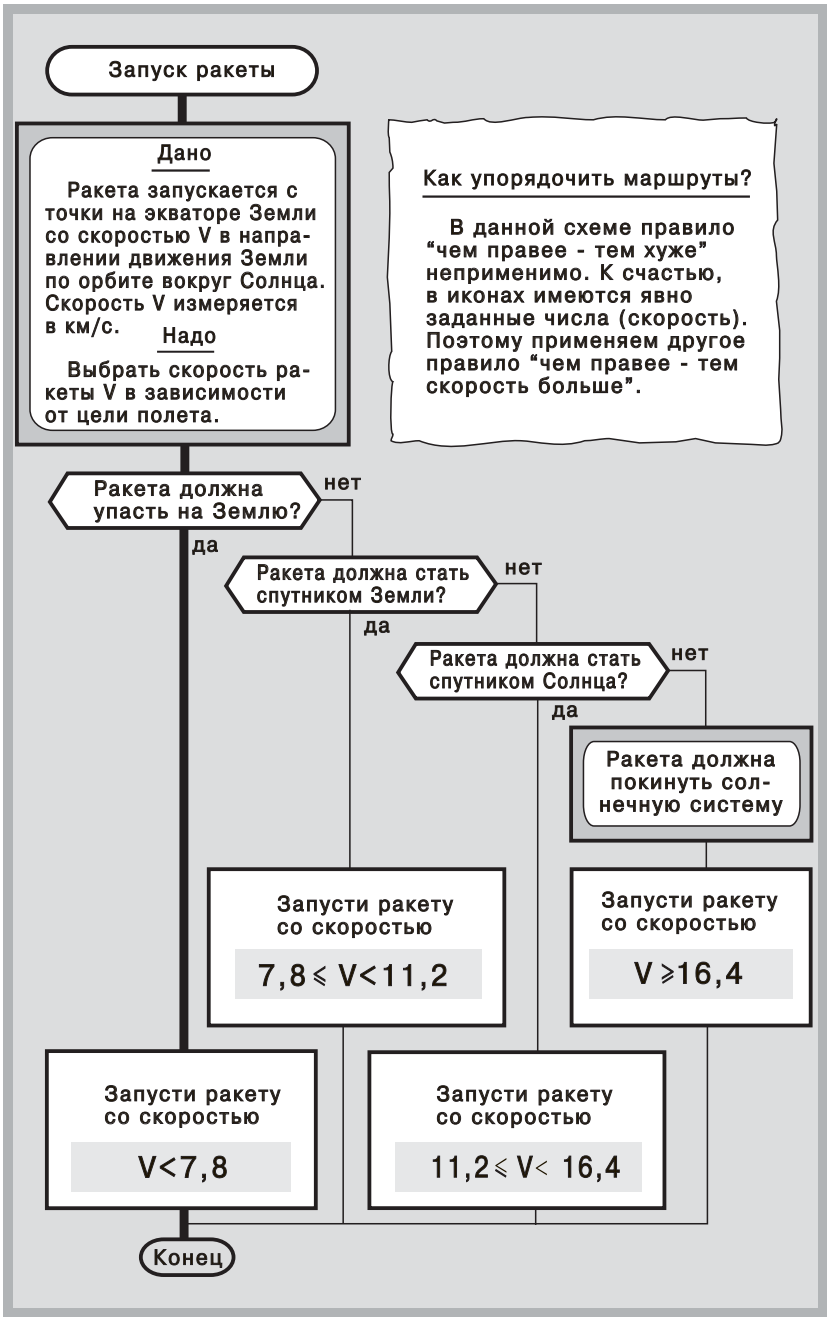


Рис. 11. **Маршруты упорядочены слева направо согласно правилу “чем правее — тем скорость больше”**

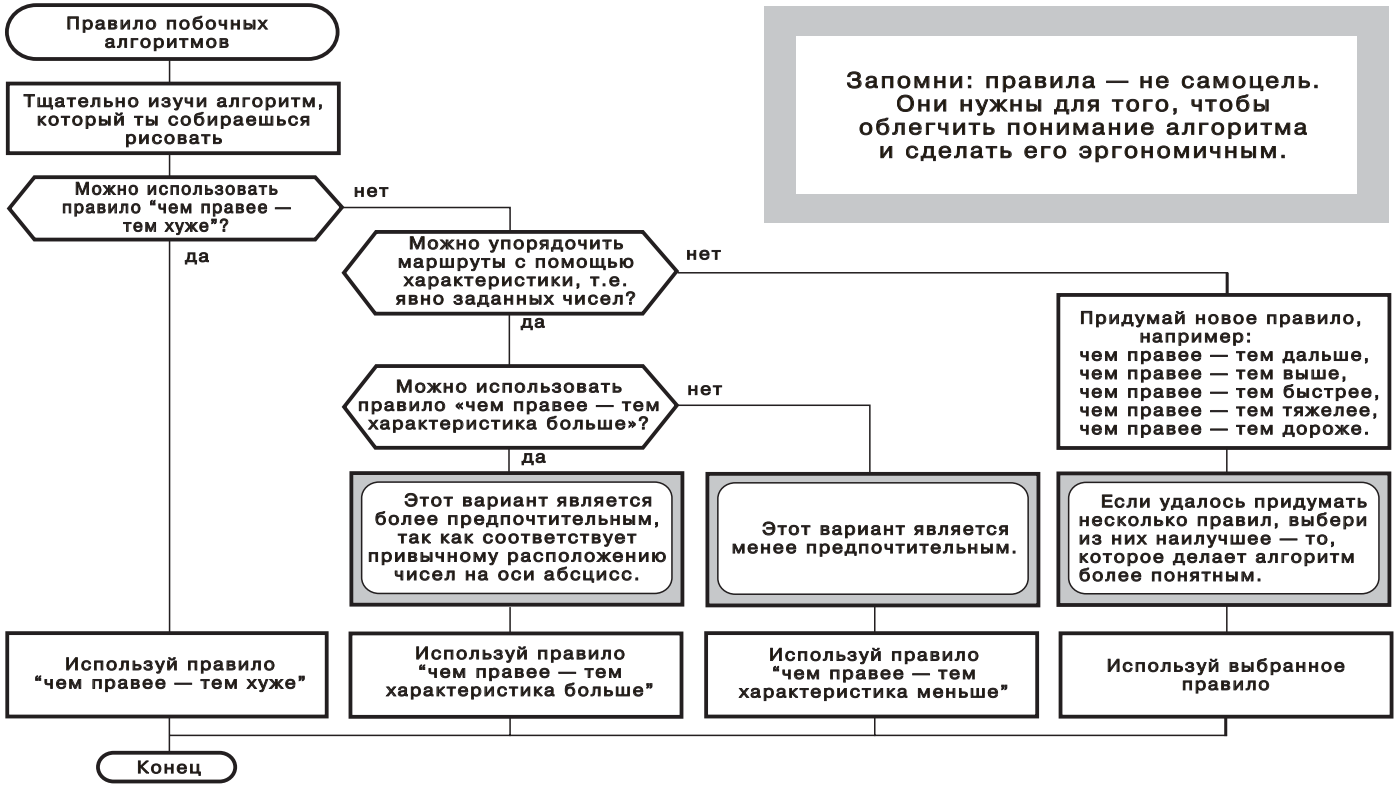


Рис. 12. **Правило побочных маршрутов, записанное в виде алгоритма**

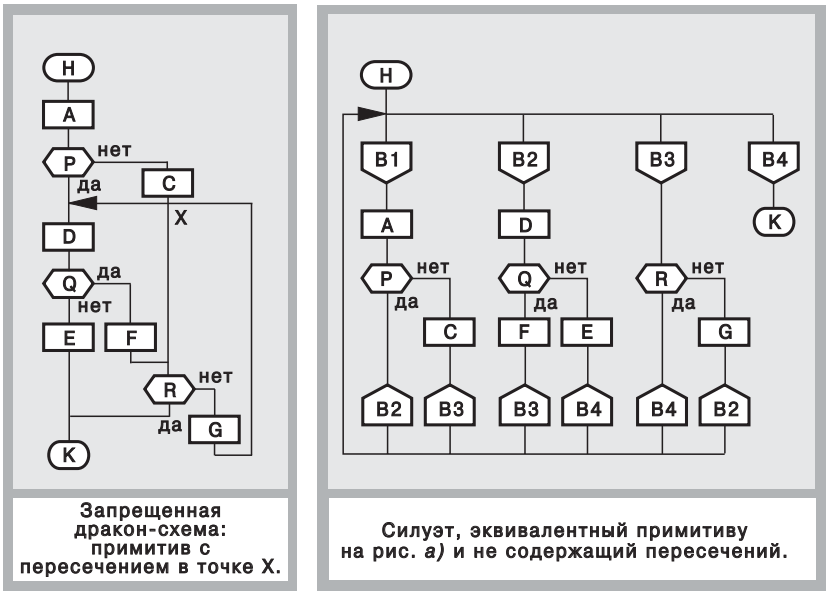


Рис. 13. **Замена примитива на эквивалентный ему силуэт позволяет устранить любые пересечения линий**

При вычерчивании обычных блок-схем допускаются два типа пересечения линий: явное, изображенное крестом линий, и замаскированное, выполняемое с помощью так называемых соединителей. Как известно, соединитель “используется для обрыва линии и продолжения ее в другом месте... для избежания излишних пересечений” [2].

В языке ДРАКОН все перечисленные ухищрения (пересечения, обрывы, соединители) по эргономическим соображениям считаются вредными и категорически запрещены, так как они засоряют поле чертежа ненужными деталями, создают визуальные помехи для глаз и отвлекают внимание от главного.

Поскольку запрет пересечений является серьезным топологическим ограничением, возникает вопрос: можно ли произвольный алгоритм изобразить в виде дракон-схемы?

Теорема 1. Любая структурная программа может быть изображена на языке ДРАКОН двумя способами: в виде примитива и в виде силуэта.

Теорема 2. Произвольная (неструктурная) программа в ряде случаев не может быть изображена в виде примитива; однако с помощью эквивалентных преобразований, допускающих введение дополнительных переменных (идентификаторов ветки), она всегда может быть изображена в виде силуэта.

Чтобы прояснить вопрос, обратимся к примерам. На рис. 13*а* приведена запрещенная дракон-схема: примитив, в котором имеется неустранимое (без введения дополнительных переменных) пересечение. На рис. 13*б* изображен силуэт, который, как нетрудно убедиться, эквивалентен примитиву на рис. 13*а* и вместе с тем не содержит ни одного

пересечения. Таким образом, пример на рис. 13 подтверждает справедливость теоремы 2[1].

Подведем итоги. Язык ДРАКОН обладает важным достоинством: он позволяет изобразить *любой* алгоритм, полностью отказавшись от таких эргономически неудачных приемов, как пересечения, обрывы, соединители. Отсутствие "паразитных элементов" создает дополнительные удобства для читателя, делает дракон-схему прозрачной, облегчает понимание.

ВИЗУАЛЬНЫЙ И ТЕКСТОВЫЙ СИНТАКСИС ДРАКОНА

ДРАКОН — визуальный язык, в котором используются два типа элементов: графические фигуры (*графоэлементы*) и текстовые надписи, расположенные внутри или снаружи графических фигур (*текстоэлементы*). Следовательно, синтаксис ДРАКОНА распадается на две части. *Визуальный синтаксис* охватывает алфавит графоэлементов, правила их размещения в поле чертежа и правила связи графоэлементов с помощью соединительных линий. *Текстовый синтаксис* задает алфавит символов, правила их комбинирования и привязку к графоэлементам (привязка необходима потому, что внутри разных графических фигур используются разные типы выражений). *Оператором языка ДРАКОН* является графоэлемент или комбинация графоэлементов, взятые вместе с текстовыми надписями.

Одновременное использование графики и текста говорит о том, что ДРАКОН адресуется не только к словесно-логическому мышлению автора и читателя программы, но сверх того активизирует интуитивное, образное, правополушарное мышление, стимулируя его не написанной, а именно нарисованной программой, т. е. программой-картинкой.

СЕМЕЙСТВО ДРАКОН-ЯЗЫКОВ

ДРАКОН — не один язык, а целое семейство, все языки которого имеют одинаковый визуальный синтаксис (что зрительно делает языки семейства почти близнецами) и отличаются текстовым синтаксисом.

ДРАКОН-1 — визуальный псевдоязык, визуальный аналог обычного текстового псевдокода. Он служит для описания структуры деятельности, создания технологий, алгоритмов и проектов программ, используется в методе пошаговой детализации, а также при формализации профессиональных знаний.

ДРАКОН-2 — язык визуального программирования реального времени. Он является элементом *CASE*-технологии для разработки программного обеспечения систем управления ракет и космических объектов, а также атомных электростанций, нефтехимических и металлургических заводов, биотехнологических производств и т. д.

---

[1] Доказательство теорем 1 и 2 предоставляем читателю. *Указание:* необходимо опереться на теорему о структурировании и метод Ашкрофта—Манны [5, 6].

Кроме того, семейство включает гибридные визуальные языки программирования: ДРАКОН-БЕЙСИК, ДРАКОН-ПАСКАЛЬ, ДРАКОН-СИ и т. д. Чтобы получить гибридный язык, например, ДРАКОН-СИ, необходимо взять визуальный синтаксис ДРАКОНА и присоединить к нему по определенным правилам текстовый синтаксис языка СИ.

Строгое разграничение визуального и текстового синтаксиса позволяет в максимальной степени расширить сферу применения языка, обеспечивая его гибкость и универсальность. При этом *единообразие* правил визуального синтаксиса семейства ДРАКОН-языков обеспечивает их концептуальное единство, а *разнообразие* текстовых правил (т. е. возможность выбора любого текстового синтаксиса) определяет гибкость языка и легкую настройку на различные проблемные и предметные области.

В настоящей книге основное внимание уделяется визуальному псевдоязыку ДРАКОН-1. Что касается остальных языков ДРАКОН-семейства, даются лишь краткие пояснения.

ВЫВОДЫ

Приведем сводку эргономических правил, позволяющих улучшить когнитивное качество дракон-схем и сделать алгоритмы, программы и технологии более понятными.

1. Сложный алгоритм следует рисовать в виде силуэта, простой — в виде примитива.
2. В иконе “заголовок” запрещается писать слово “начало”; вместо этого следует указать понятное и точное название алгоритма.
3. Разбейте сложный алгоритм на части, каждую часть изобразите в виде ветки. Дайте частям доходчивые и четкие названия и запишите их в иконах “имя ветки”.
4. Вход в ветку возможен только через ее начало.
5. В иконе “адрес” разрешается писать имя одной из веток, другие надписи запрещены.
6. Ветки следует располагать в пространстве согласно правилу: чем правее, тем позже. Наличие веточного цикла модифицирует это правило.
7. Примитив обязательно имеет шампур. Это значит, что у примитива иконы “заголовок” и “конец” всегда лежат на одной вертикали, которая и называется “шампур”.
8. Каждая ветка обязательно имеет шампур. У правой ветки шампур — это вертикаль, соединяющая иконы “имя ветки” и “конец”. У остальных веток шампуром служит вертикальная линия, соединяющая иконы “имя ветки” и “адрес”, а если адресов несколько — с левым из них.
9. Алгоритм всегда имеет главный маршрут, который должен идти по шампуру.

10. Побочные маршруты должны быть упорядочены слева направо согласно одному из выбранных критериев, например: чем правее — тем хуже.
11. В иконе “конец” следует писать слово “конец”.
12. Соединительные линии могут идти либо горизонтально, либо вертикально. Наклонные линии не допускаются.
13. Пересечения линий запрещены.
14. Обрывы линий запрещены.
15. Использование соединителей запрещено.

Глава 7

ЭРГОНОМИЧНЫЕ АЛГОРИТМЫ

На ошибках мы горим! —
Мне сказал Алеха.
Непонятный алгоритм —
Это очень плохо.

Мы ошибки победим!
Надо сделать вот чего —
Надо сделать алгоритм
Ясным и доходчивым!

Юрий Примашев

ВИЗУАЛЬНАЯ ПРОВЕРКА АЛГОРИТМОВ

При решении сложных задач, таких как предсказание погоды, управление войсками, управление большой электростанцией или нефтехимическим заводом, приходится создавать крупномасштабные алгоритмы, насчитывающие сотни тысяч и даже миллионы команд. Практика показывает: чем крупнее алгоритмы, тем больше в них ошибок, тем сложнее их найти. Исправлять ошибки в огромных и сложных алгоритмах — трудное и дорогостоящее занятие, причем цена ошибки тем выше, чем позже она обнаружена.

Наименьший ущерб приносят ошибки, которые удается обнаружить сразу, до генерации кода и исполнения программы на компьютере — в ходе мозговой или, что одно и то же, визуальной проверки. Проверка называется *мозговой*, если главная работа по поиску ошибок выполняется благодаря интеллектуальным усилиям человека, а компьютер играет хотя и важную, но вспомогательную роль. При такой проверке человек тщательно изучает компьютерные чертежи спецификаций и алгоритмов, представленные на бумаге или экране.

Мозговую проверку, как и любую другую деятельность, нужно грамотно проектировать. Критерием ее эффективности служит минимизация средних интеллектуальных усилий мозга, затрачиваемых на выявление одной ошибки. Нейронная конструкция мозга такова, что он может эффективно проводить визуальную проверку отнюдь не при любых условиях, а только в том случае, если проверяемые чертежи обладают высоким когнитивным качеством. Чтобы минимизировать удельные интеллектуальные усилия, необходимо, чтобы когнитивно-значимые характеристики чертежей были хорошо согласованы с конструктивными характеристиками мозга. Это значит, что спецификации и алгоритмы должны быть специально приспособлены для быстрой и надежной проверки, для легкого и вместе с тем глубокого понимания.

ЧТО ТАКОЕ ЭРГОНОМИЧНЫЙ АЛГОРИТМ?

В связи с этим полезно ввести понятие *эргономичный алгоритм*, т. е. алгоритм, удовлетворяющий критерию сверхвысокой понимаемости. Преимущество эргономичных алгоритмов в том, что они намного понятнее, яснее, нагляднее и доходчивее, чем обычные. Если алгоритм непонятный, в нем трудно или даже невозможно заметить затаившуюся ошибку. И наоборот, чем понятнее алгоритм, тем легче найти дефект. Поэтому более понятный, эргономичный алгоритм намного лучше обычного. Лучше в том смысле, что он облегчает выявление ошибок, а это очень важно. Ведь чем больше ошибок удастся обнаружить при визуальной проверке, тем больше вероятность, что вновь созданный алгоритм окажется правильным, безошибочным, надежным. Кроме того, эргономичные алгоритмы удобнее для изучения, их проще объяснить другому человеку.

В предыдущей главе мы рассмотрели несколько способов, позволяющих улучшить эргономические характеристики алгоритмов. В этой главе, продолжая тему, попытаемся ответить на вопрос: можно ли повысить эргономичность алгоритмов, используя формальный *метод эквивалентных преобразований?*

Для этого понадобится особый понятийный аппарат, заметно отличающийся от того, которым обычно пользуются программисты, так как традиционные понятия плохо приспособлены для решения проблемы понимания и совершенно не учитывают специфику визуальных образов.

ЧЕМ ОТЛИЧАЕТСЯ ИКОНА “ВОПРОС” ОТ РАЗВИЛКИ?

На рис. 1 (позиция И4) изображена икона “вопрос”. Она называется так, потому что внутри нее пишут “*да-нетный*” *вопрос*, т. е. вопрос, на который можно ответить либо “да”, либо “нет”. Все другие ответы запрещены. Вот примеры да-нетных вопросов: утюг сломался? Тетя приехала? Вася купил хлеб? Преступника арестовали? Эта лужа больше, чем та? Температура выше нуля?

Икона “вопрос” имеет один вход сверху и два выхода: вниз и вправо. Выход влево запрещен и никогда не используется.

На рис. 2 (позиция 2) показана макроикона “развилка”. Она содержит икону “вопрос”, точку слияния и два плеча: левое и правое (рис. 14*б*). *Левое плечо* есть путь от нижнего выхода иконы-вопроса до *точки слияния*. *Правое плечо* начинается у правого выхода иконы-вопроса и заканчивается в точке слияния (рис. 15). Таким образом, плечо имеет в своем составе надпись “да” или “нет”, соединительные линии, точку слияния, а также иконы. Одно из двух плеч может быть *пустым* (не содержать икон).

Развилки бывают простые и сложные. *Простая развилка* содержит только одну икону-вопрос. Примеры простых развилок показаны на рис. 16. Развилка называется *сложной*, если в ее плечах имеется по

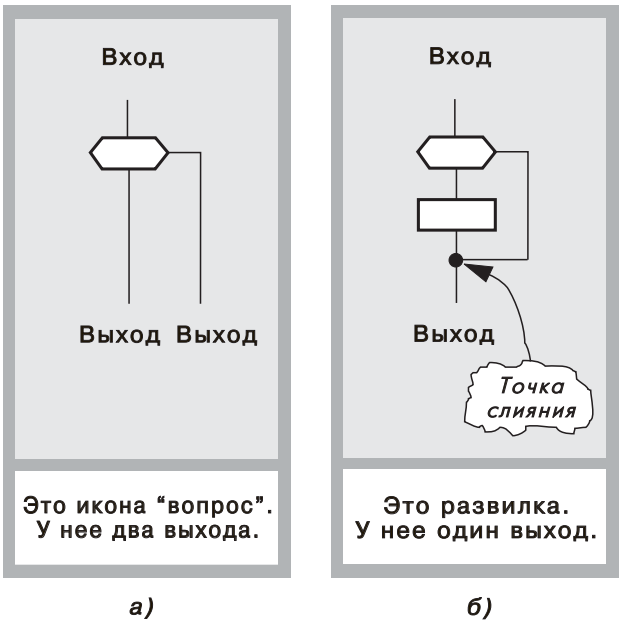


Рис. 14. **Чем отличается развилка от иконы “вопрос”?**

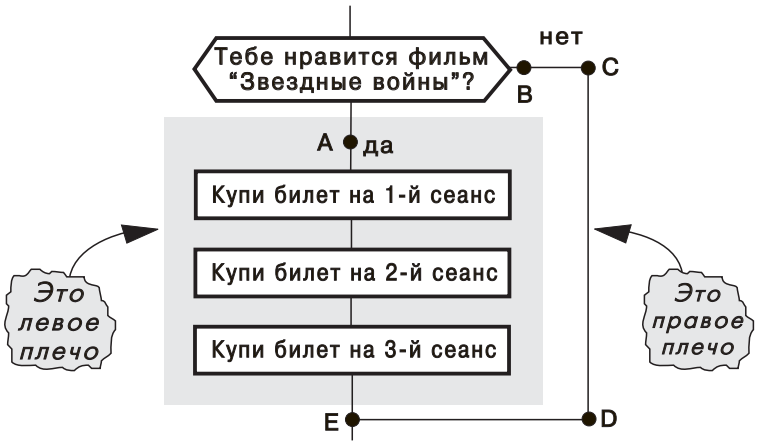


Рис. 15. **У развилки два плеча: левое и правое**

крайней мере одна простая развилка. На рис. 10*а* показаны три сложные развилки. Например, развилка “Борщ очень вкусный?” сложная, так как ее левое плечо содержит простую развилку “Борщ сильно пересолен?”. Другие примеры сложных развилок показаны на рис. 17.

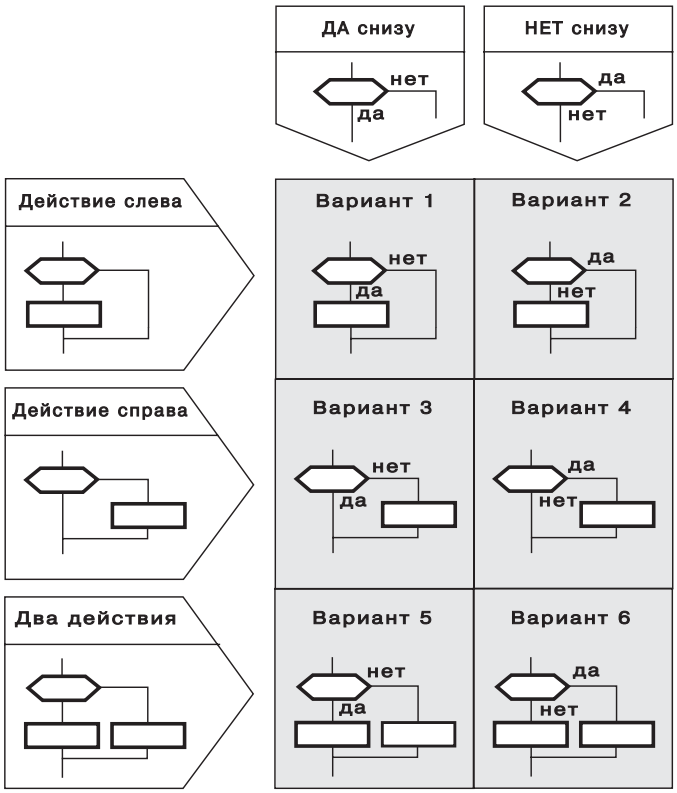
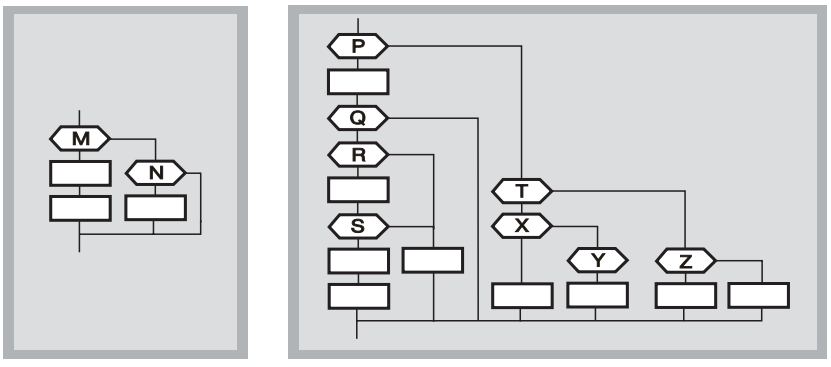


Рис. 16. **Шесть вариантов изображения развилки**



а) *б)*

Рис. 17. **Примеры сложных развилок**

МАРШРУТЫ И ФОРМУЛЫ МАРШРУТОВ

На рис. 18*а* представлена дракон-схема “Охота на мамонта”. Заменим текст внутри икон буквами. Вместо “Охота на мамонта” запишем букву А, вместо “Поймай мамонта” — букву Б и т. д. В результате получим литеральную (буквенную) дракон-схему на рис. 18*б*. Литеральные схемы удобно использовать для описания маршрутов.

Маршрут — это графический путь от начала до конца алгоритма, проходящий через иконы и соединительные линии. Маршрут можно описать с помощью *формулы*, которая представляет собой последовательность букв, обозначающих иконы. Все иконы, включая одинаковые, обозначаются разными буквами.

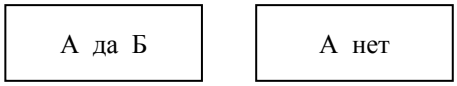
Линейный (неразветвленный) алгоритм имеет только один маршрут и одну формулу. Например, схема на рис. 18*б* описывается формулой

Разветвленный алгоритм имеет несколько (два или более) маршрутов, причем у каждого маршрута своя, отличная от других формула (рис. 19, 20). В формулах разветвленных алгоритмов наряду с буквами, обозначающими иконы, используются слова “да” или “нет” (отделяемые пробелами).

ЧТО ТАКОЕ РОКИРОВКА?

Рокировка — это преобразование алгоритма, при котором левое и правое плечо развилки меняются местами. Простейшие примеры рокировки показаны на рис. 8 и 21.

Два алгоритма называются *равносильными*, если для каждого маршрута первого алгоритма можно найти парный маршрут второго алгоритма, причем для каждой пары маршрутов их формулы совпадают. Обратимся к рис. 22. Легко убедиться, что схемы на рис. 22*а* и *б* имеют одинаковый набор маршрутов:



Следовательно, указанные дракон-схемы равносильны.

Формальное преобразование алгоритма А1 в алгоритм А2 называется равносильным, если алгоритмы А1 и А2 равносильны. Сказанное означает, что рокировка является равносильным преобразованием алгоритмов. При рокировке слова “да” и “нет” обязательно меняются местами.

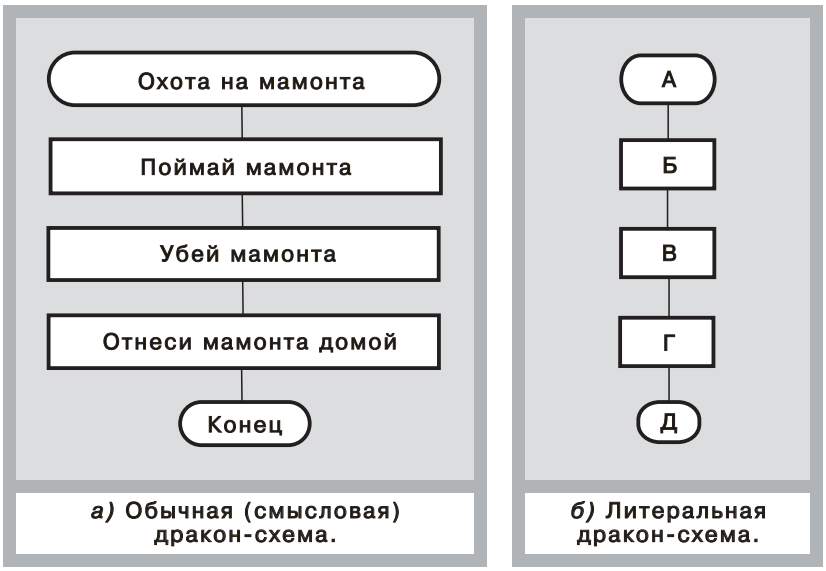


Рис. 18. **Как преобразовать обычную дракон-схему в литеральную?**

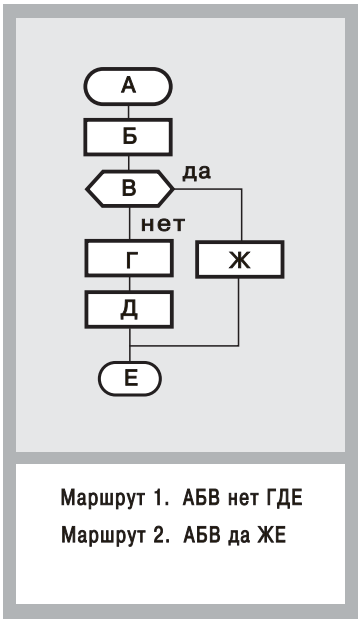


Рис. 19. **Алгоритм с двумя маршрутами**

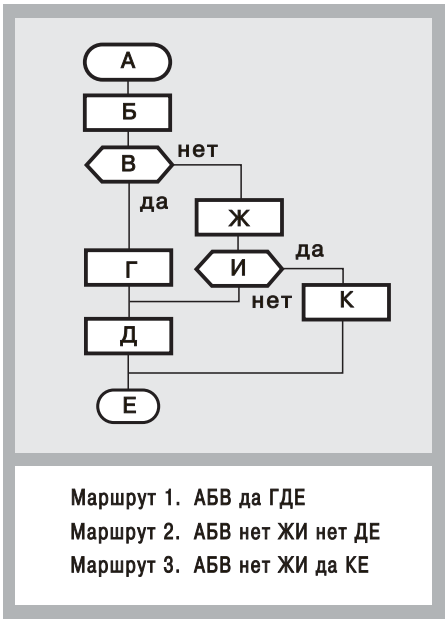
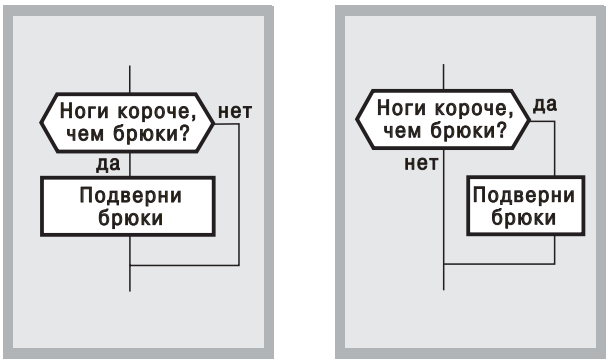
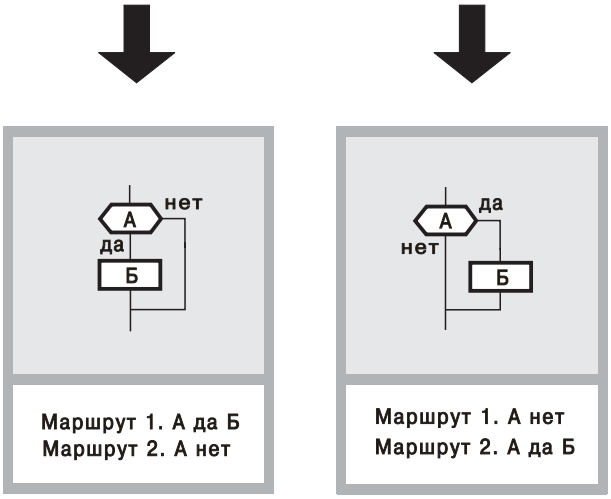


Рис. 20. **Алгоритм с тремя маршрутами**



а) *б)*

Рис. 21. **Пример равносильных алгоритмов**



а) *б)*

Рис. 22. **Литеральные дракон-схемы, полученные из смысловых дракон-схем на рис. 21**

ИСПОЛЬЗОВАНИЕ РОКИРОВКИ ДЛЯ УЛУЧШЕНИЯ ЭРГОНОМИЧНОСТИ

Мы убедились, что рокировка алгоритма на рис. 8*а* позволила получить алгоритм на рис. 8*б*, который имеет более высокие эргономические характеристики, так как на рис. 8*б* соблюдается правило “главный маршрут идет по шампуру”. Это означает, что равносильное преобразование “рокировка” в примере на рис. 8 действительно улучшает эргономичность алгоритма.

Попытаемся обосновать аналогичный вывод для рис. 21, рассмотрев последовательно все промежуточные шаги рассуждений.

На первом шаге выдвигаем гипотезу: для сравнения двух маршрутов на рис. 21*а* можно использовать признак “лучше—хуже”.

На втором шаге проверяем гипотезу с помощью рассуждений: если брюки впору — это хорошо, если их приходится подворачивать — это плохо. Следовательно, использование в данном алгоритме признака “лучше—хуже” правомерно.

На третьем шаге определяем главный маршрут, который по соглашению соответствует признаку “хорошо”. Следовательно, главный маршрут на рис. 21*а* идет через правое плечо развилки, что соответствует хорошей ситуации “брюки впору — их не надо подворачивать”.

На четвертом шаге констатируем, что главный маршрут на рис. 21*а* не идет по шампуру. Делаем вывод: алгоритм на рис. 21*а* является плохим, неэргономичным, однако его можно исправить с помощью рокировки.

На пятом шаге выполняем рокировку и получаем более эргономичный алгоритм на рис. 21*б*. На этом процедура заканчивается [1].

Рассмотрим теперь более сложный алгоритм на рис. 10*а*. Этот алгоритм тоже плохой, потому что содержит три плохие (неэргономичные) развилки: 1) “В меню есть твой любимый салат?”, 2) “Борщ очень вкусный?”, 3) “Жаркое, как подошва?”. Последовательно применяя к развилкам три операции “рокировка”, получаем эргономичный алгоритм на рис. 10*б*.

Таким образом, на основании анализа примеров мы убедились: равносильное преобразование “рокировка” позволяет улучшить эргономичность алгоритмов. Однако этот вывод относится только к смысловым алгоритмам (где можно указать главный маршрут) и неприменим к литеральным алгоритмам (где понятие главного маршрута теряет силу). Отсюда вытекает, что применение рокировки к литеральной схеме на рис. 22 бессмысленно, так как в данном случае рокировка не влияет на эргономичность.

---

[1] Правило “главный маршрут идет по шампуру” — это необходимое, но отнюдь не достаточное условие эргономичности алгоритма. Другое условие — эргономизация текста. Вопрос “Ноги короче, чем брюки?” звучит вычурно, противоестественно и сбивает с толку читателя. Вместо него следует написать: “Брюки слишком длинные?” В итоге получим действительно понятный и эргономичный алгоритм.

ВЕРТИКАЛЬНОЕ И ГОРИЗОНТАЛЬНОЕ ОБЪЕДИНЕНИЕ

Бывает, что в дракон-схемах одинаковые иконы повторяются несколько раз. Например, на рис. 23*а* икона “Отдай мотоцикл в ремонт и впредь будь умнее” встречается три раза. Это плохо: навязчивые повторения раздражают читателя, которому приходится тратить лишнее время и несколько раз читать одно и то же. К счастью, некоторые повторы можно устранить. Такая возможность появляется, если одинаковые иконы соседствуют друг с другом, причем их выходы соединены между собой (рис. 23*а*). В этом случае действует правило “повторы запрещены”. Устранение повторов производится с помощью вертикальной линии (рис. 23*б*) и называется *вертикальным объединением*. Нетрудно доказать, что операция “вертикальное объединение” осуществляет равносильное преобразование алгоритмов.

Название “объединение” объясняется тем, что несколько одинаковых икон объединяются в одну. Последовательность действий такова: сначала входы этих икон объединяются вертикальной линией, после чего удаляются все одинаковые иконы, кроме крайней левой.

Сравнивая алгоритмы на рис. 23*а* и *б*, легко заметить, что схема на рис. 23*б* обладает более высоким эргономическим качеством: она стала более компактной, ясной и наглядной, поскольку освободилась от бессмысленного повторения икон. Отсюда проистекает вывод: равносильное преобразование “вертикальное объединение” позволяет улучшить эргономичность алгоритмов.

Иногда для исключения повторов используется не вертикальная, а горизонтальная линия. Соответствующая операция называется *горизонтальным объединением*. Она также является равносильным преобразованием.

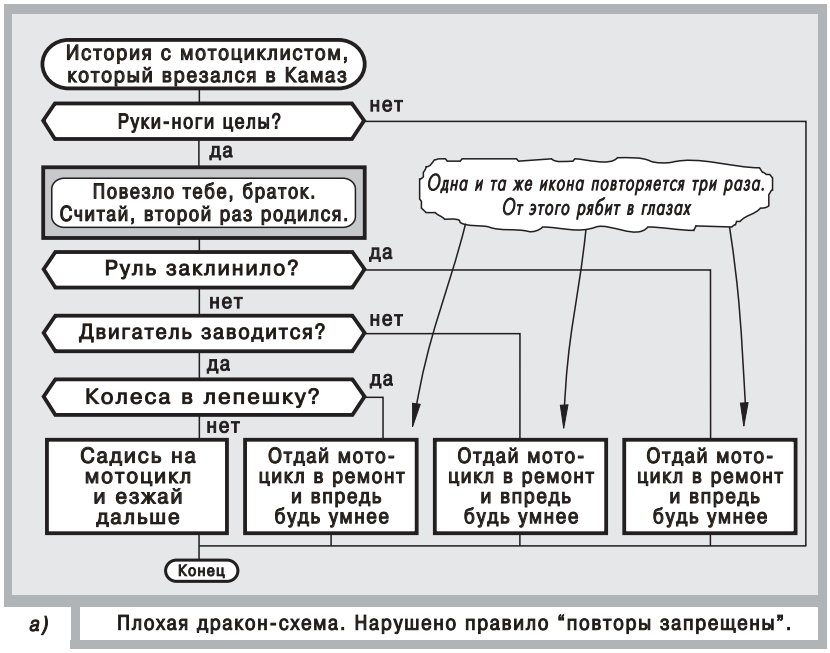
Рассмотрим пример на рис. 24*а*. Сначала с помощью горизонтального объединения устраним повторение иконы “Съешь кашу” и получим схему на рис. 24*б*. Затем применим вертикальное объединение и исключим повторение развилки “Есть можно?”. Окончательный результат показан на рис. 24*в*. Полученная схема более удобна и занимает меньше места, чем исходная схема на рис. 24*а*.

Разобранный пример показывает, что равносильное преобразование “горизонтальное объединение” также улучшает эргономичность алгоритмов.

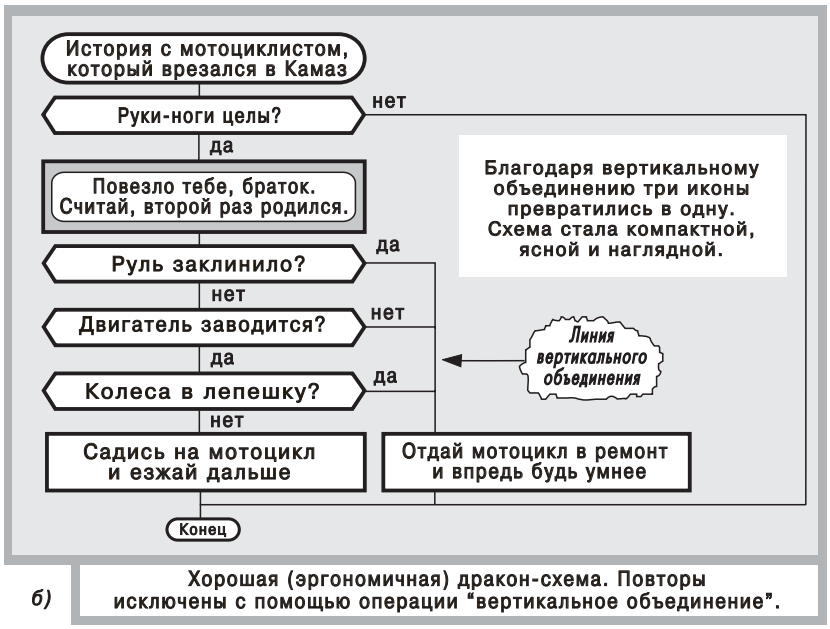
П р е д о с т е р е ж е н и е: выполняя вертикальное и горизонтальное объединение, нужно следить, чтобы не появились пересечения соединительных линий (рис. 25).

ЭРГОНОМИЧНОСТЬ ЛИТЕРАЛЬНЫХ АЛГОРИТМОВ

Можно ли улучшить эргономичность литеральных дракон-схем с помощью равносильных преобразований? Мы уже знаем, что рокировка в этом случае бесполезна. Однако вертикальное и горизонтальное объединение позволяют заметно повысить эргономичность литеральных схем.

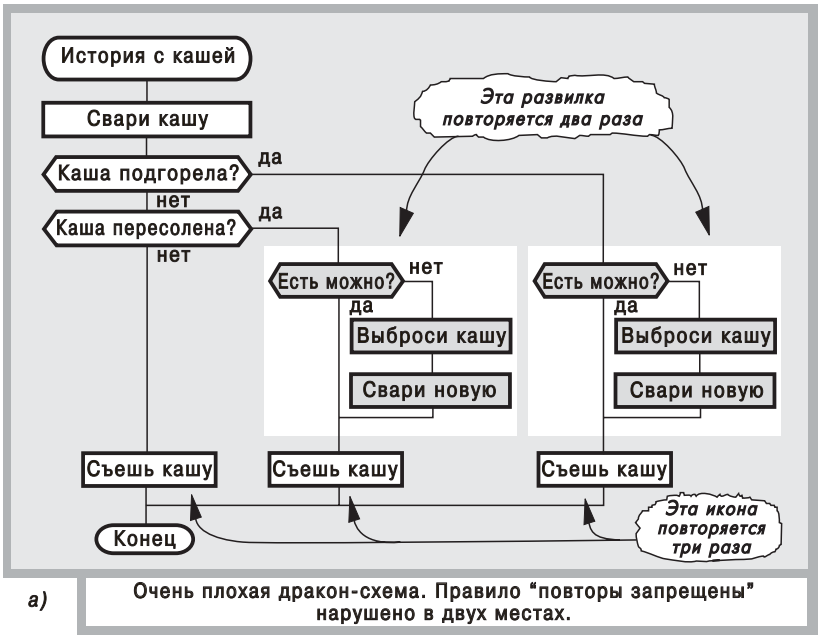


а) Плохая дракон-схема. Нарушено правило "повторы запрещены".

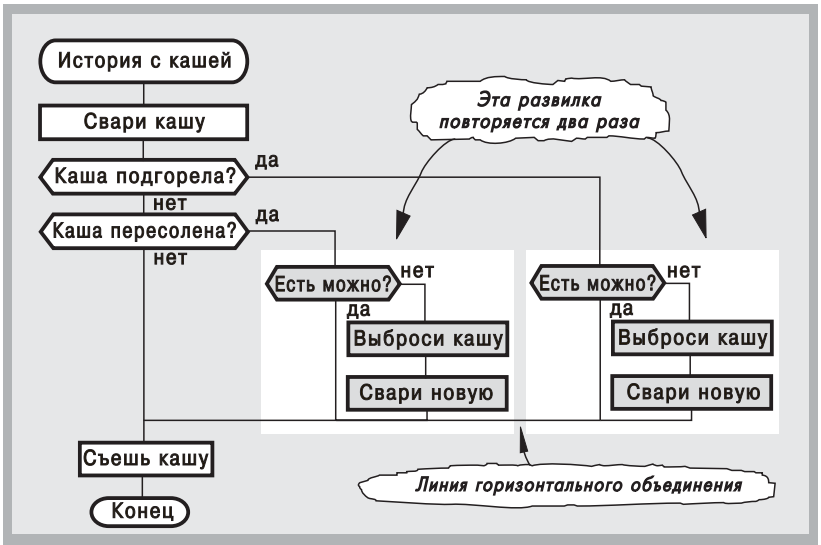


б) Хорошая (эргономичная) дракон-схема. Повторы исключены с помощью операции "вертикальное объединение".

Рис. 23. **Операция "вертикальное объединение" позволяет устранить ненужные повторы и превратить плохую (неэргономичную) дракон-схему в хорошую (эргономичную)**

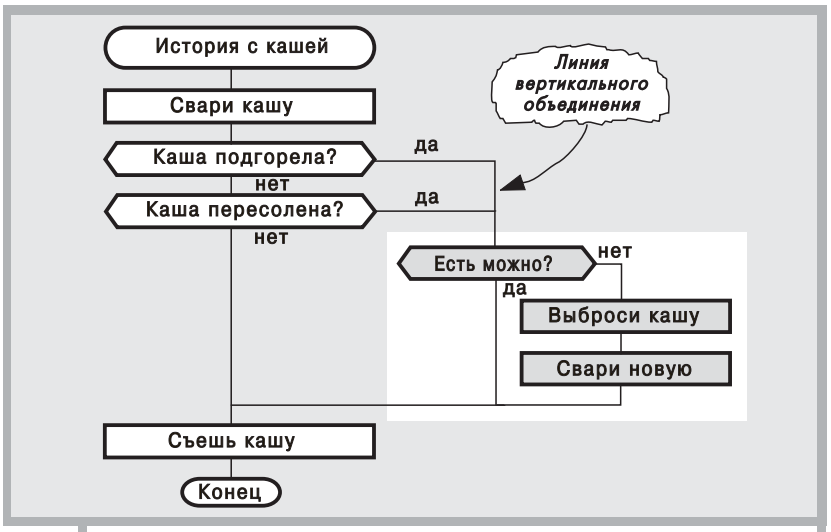


а) Очень плохая дракон-схема. Правило "повторы запрещены" нарушено в двух местах.



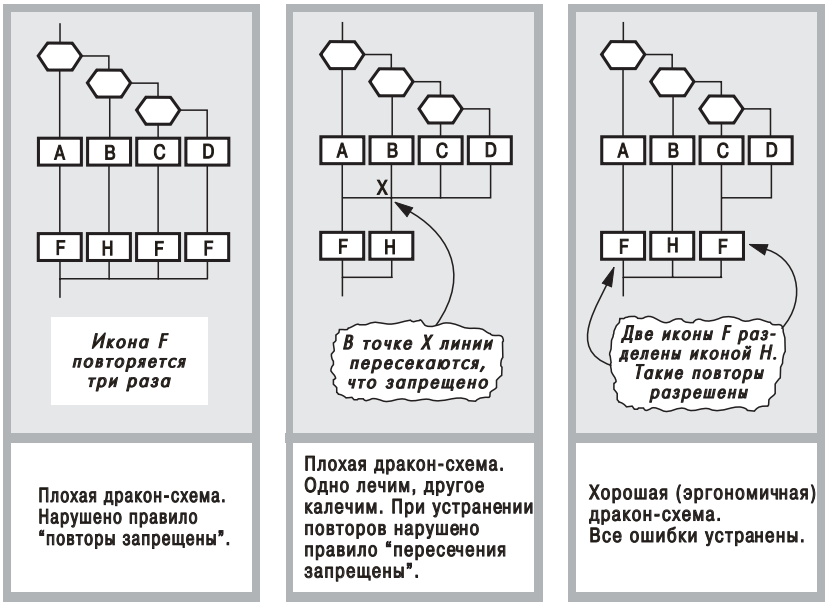
б) Плохая дракон-схема. Нарушено правило "повторы запрещены". Данная схема получена из схемы *а)* в результате горизонтального объединения трех икон "Съешь кашу".

Рис. 24. **Последовательное применение операций "горизонтальное объединение" и "вертикальное объединение" позволяет устранить ненужные повторы и превратить очень плохую (неэргономичную) дракон-схему в хорошую (эргономичную)**



в) Хорошая (эргономичная) дракон-схема. Данная схема получена из схемы *б)* в результате вертикального объединения двух развилок "Есть можно?".

Рис. 24 (окончание)



а) *б)* *в)*

Рис. 25. **Устраняя повторы, следи, чтобы не появились пересечения**

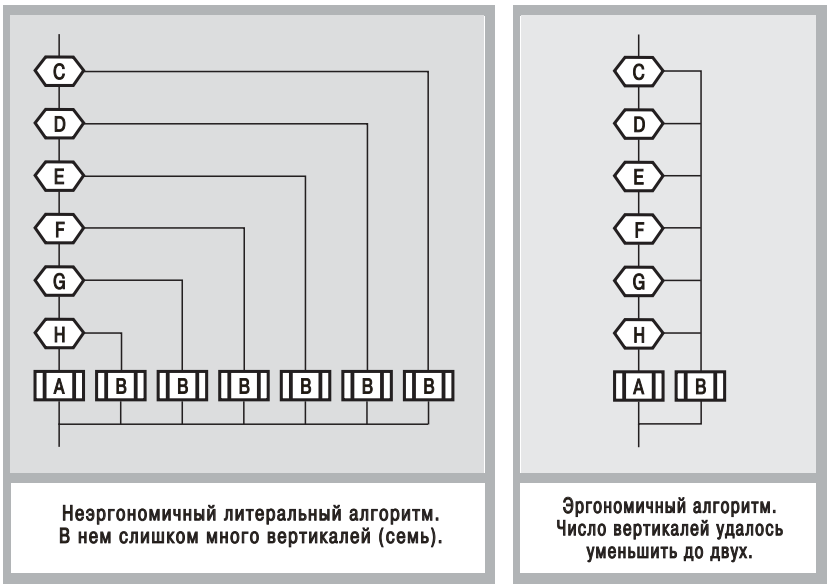
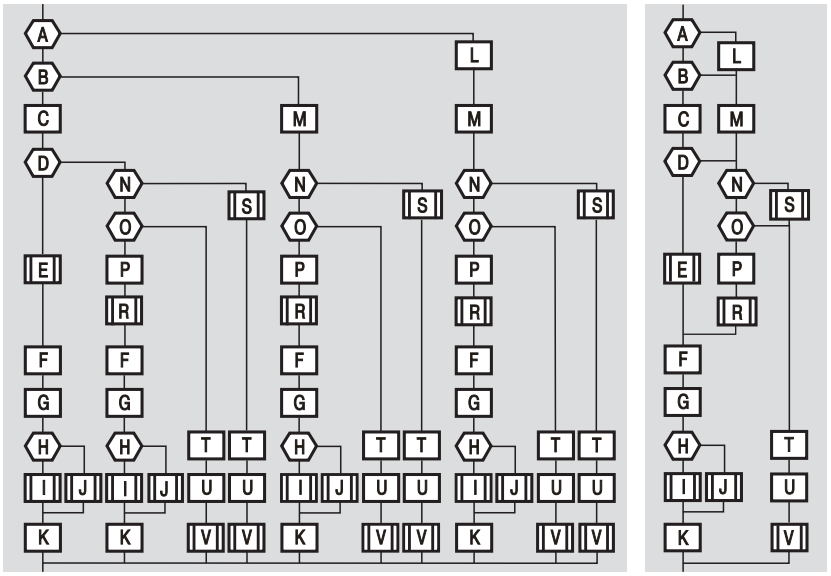


Рис. 26. **Равносильное преобразование "вертикальное объединение" улучшает эргономичность литерального алгоритма**

Чтобы убедиться в этом, обратимся к рис. 26 и 27. В самом деле, схема на рис. 26*а* выглядит неоправданно громоздкой: она содержит семь вертикалей, тринадцать икон и заставляет читателя шесть раз читать идентификатор *B*, чтобы убедиться, что правые шесть икон одинаковые. А равносильная ей схема на рис. 26*б* (полученная методом вертикального объединения) свободна от этого недостатка. Она наглядна, проста и изящна, содержит только две вертикали и восемь икон, занимает втрое меньше места на листе бумаги (на экране) и к тому же имеет всего два прямоугольных излома линий (на рис. 26*а* — семь изломов). Таким образом, литеральная схема на рис. 26*б* более эргономична, чем ее соседка.

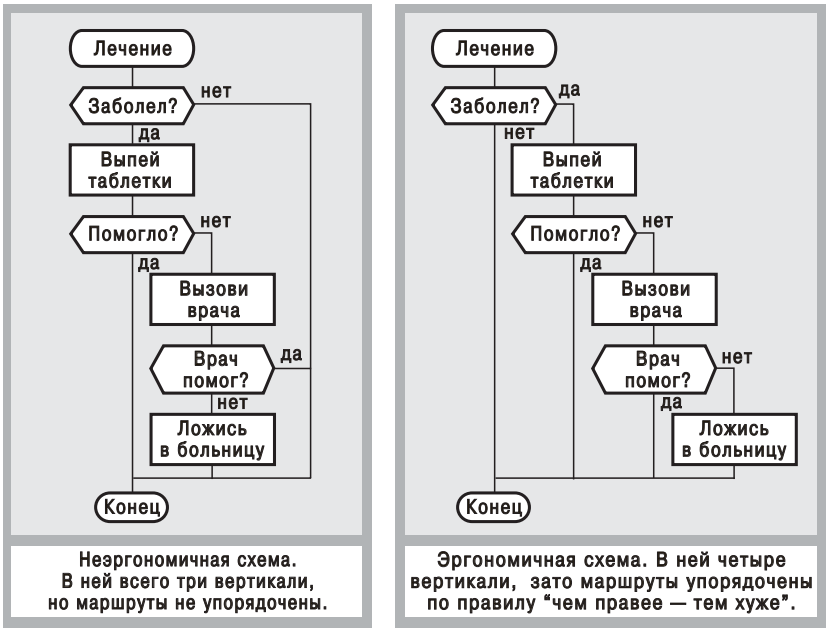
Еще более громоздкой выглядит литеральная схема на рис. 27*а*, в которой насчитывается 14 вертикалей. А равносильная ей схема на рис. 27*б* (полученная методом вертикального и горизонтального объединения) снова выигрывает: позволяет уменьшить число вертикалей почти в пять раз (с 14 до 3), сокращает число икон более чем в три раза (с 65 до 21), обеспечивает более экономное топологическое упорядочивание маршрутов, заметно сокращает суммарную длину соединительных линий.

Проведенный анализ позволяет сделать вывод: в отличие от рокировки, которая полезна только для смысловых дракон-схем, вертикальное и горизонтальное объединение улучшают эргономичность не только смысловых, но и литеральных алгоритмов.



а) *б)*

Рис. 27. **Неэргономичный литеральный алгоритм *(а)* и равносильный ему эргономичный алгоритм *(б)***



а) *б)*

Рис. 28. **Эргономическое правило упорядочивания маршрутов является более приоритетным, чем минимизация числа вертикалей**

ЧТО ДЕЛАТЬ, ЕСЛИ ЭРГОНОМИЧЕСКИЕ ТРЕБОВАНИЯ ПРОТИВОРЕЧАТ ДРУГ ДРУГУ?

До сих пор мы рассматривали простейшие случаи, когда различные эргономические требования не вступали в конфликт. Однако подобные конфликты возможны. Вот два эргономических критерия, которые могут противоречить друг другу:

- правило главного и побочного маршрутов,
- минимизация числа вертикалей.

Чтобы исключить конфликт, следует держаться принципа: *правило маршрутов имеет более высокий приоритет, нежели стремление уменьшить число вертикалей.*

В качестве иллюстрации сравним равносильные схемы на рис. 28. По критерию "минимизация числа вертикалей" выигрывает схема на рис. 28*а*, имеющая на одну вертикаль меньше. Однако в ней нарушается правило маршрутов, причем сразу в двух местах. Во-первых, главный маршрут (когда человек здоров) не совмещен с шампуром. Во-вторых, маршруты не упорядочены слева направо, ибо самое тяжелое заболевание (когда человек вынужден лечь в больницу) находится на средней вертикали, а слева и справа от нее находятся более легкие недомогания. Таким образом, схему на рис. 28*а* нельзя признать эргономичной.

Чтобы исправить недостаток, необходимо выполнить:

- *вертикальное разъединение* в точке *С* (эта операция обратна вертикальному объединению);
- рокировку развилки "Заболел?";
- рокировку развилки "Врач помог?".

В итоге получим схему на рис. 28*б*, где все маршруты упорядочены слева направо по принципу "чем правее — тем хуже". В самом деле, левая вертикаль означает, что дела идут хорошо, ибо человек здоров; значит, главный маршрут идет по шампуру. Вторая вертикаль показывает легкое недомогание, которое можно снять таблеткой. Третья вертикаль говорит: самочувствие ухудшилось, нужен врач. Наконец, четвертая (крайняя правая) вертикаль означает, что дела обстоят плохо — пришлось лечь в больницу.

Как уже упоминалось, схема на рис. 28*б* не удовлетворяет эргономическому требованию минимизации числа вертикалей. Тем не менее мы признаем ее эргономичной, поскольку выполняется более приоритетное правило маршрутов. Отсюда вытекает, что (благодаря наличию приоритетов) комплекс эргономических требований является взаимоувязанным и непротиворечивым.

ИКОНА-ВСТАВКА КАК ЭРГОНОМИЧЕСКИЙ ПРИЕМ

Мы уже знаем, что язык ДРАКОН запрещает применять пересечения, обрывы и соединители. Отсюда вытекает жесткое ограничение: любая дракон-схема должна целиком размещаться на одном листе бумаги. А если она слишком большая и вылезает за рамки прокрустова ложа? Тогда можно взять бумагу больших размеров, например формата А1. А если схема громадная и все равно не помещается? На этот случай предусмотрены специальные приемы, позволяющие уменьшить габариты дракон-схемы и разрубить ее на куски. Эти приемы позволяют

разместить дракон-схему на листах желаемого размера. Рассмотрим проблему на “миниатюрном” примере.

Предположим, линейный алгоритм состоит из четырнадцати икон, а бумажный лист небольшой, так что на нем можно разместить по вертикали не более десяти икон. Как быть?

На рис. 29 показаны два варианта решения проблемы. Алгоритм на рис. 29*а* не годится, так как на участке *AB* рабочая точка движется вверх, что запрещено правилами языка ДРАКОН. Преодолеть затруднение можно двумя способами: применить конструкцию “силуэт” (о которой шла речь в гл. 6) либо разделить алгоритм на две части. Рассмотрим последний способ. Для этого удалим из алгоритма несколько связанных по смыслу икон, а вместо них нарисуем икону-заместитель, которая называется *вставкой* (рис. 29*б*). Вставка нужна, чтобы напомнить об изъятых иконах. Вставка занимает мало места — намного меньше, чем выброшенные иконы, поэтому алгоритм становится короче. Разумеется, выброшенные иконы не пропадают — они образуют новый алгоритм — *алгоритм-вставку*.

Икона-вставка — это команда “Передай управление в алгоритм-вставку” (рис. 30). Икона “конец” алгоритма-вставки означает: “Верни управление в основной алгоритм”. При этом управление возвращается в точку, расположенную после иконы-вставки. На языке программистов алгоритм-вставка — это процедура, а икона-вставка — это оператор “Вызов процедуры”.

ЧТО ТАКОЕ ПОДСТАНОВКА?

Операция “подстановка” связана с использованием иконы-вставки. Она выполняется за три шага.

Шаг 1. Из дракон-схемы удаляется фрагмент, имеющий один вход и один выход.

Шаг 2. Вместо него подставляется икона-вставка с именем *X*.

Шаг 3. К удаленному фрагменту добавляется икона-заголовок с тем же именем *X* и икона-конец; в результате получается алгоритм-вставка.

Два алгоритма на рис. 29 неравносильны: их формулы не совпадают, поскольку маршрут на рис. 29*а* содержит 14 икон, а на рис. 29*б* — 17 икон (см. также рис. 30). Вместе с тем нетрудно убедиться, что подстановка — эквивалентное преобразование алгоритмов, так как исходный и преобразованный алгоритмы дают одинаковые результаты для одних и тех же исходных данных. Попутно заметим, что равносильные алгоритмы всегда эквивалентны (обратное неверно). Таким образом, операция “подстановка” представляет собой эквивалентное (но не равносильное) преобразование алгоритмов.

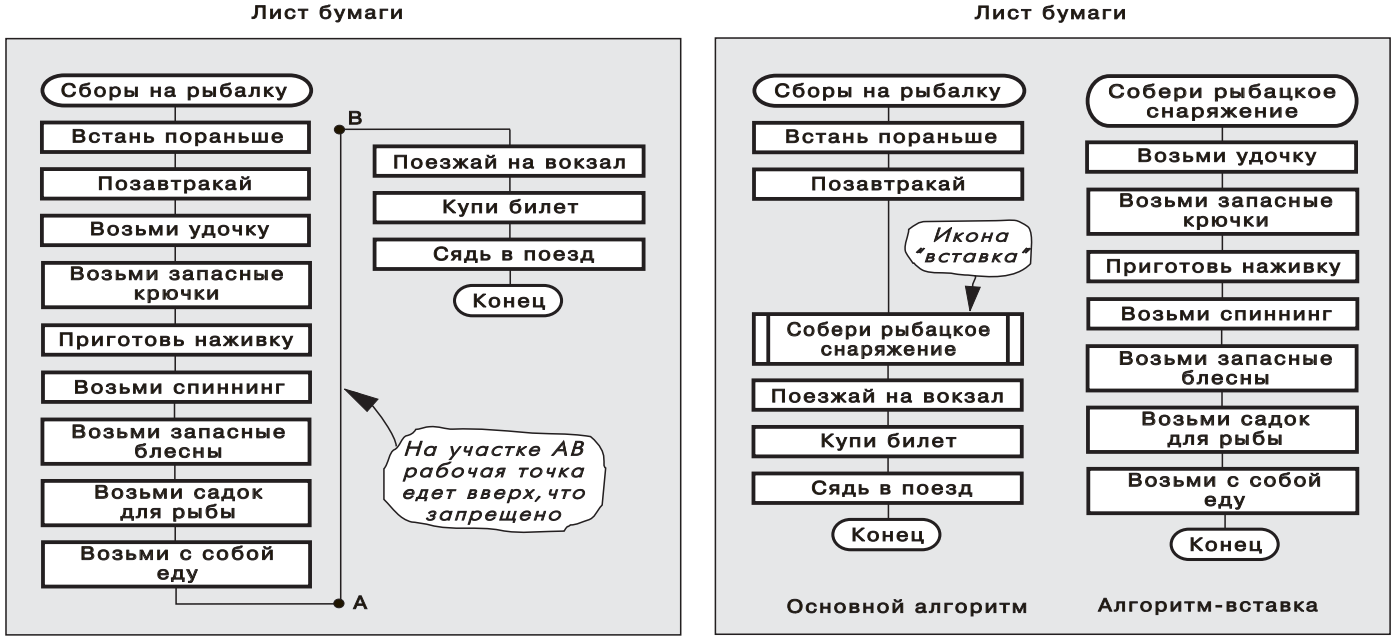


Рис. 29. **Как исправить ошибку “движение вверх”?**

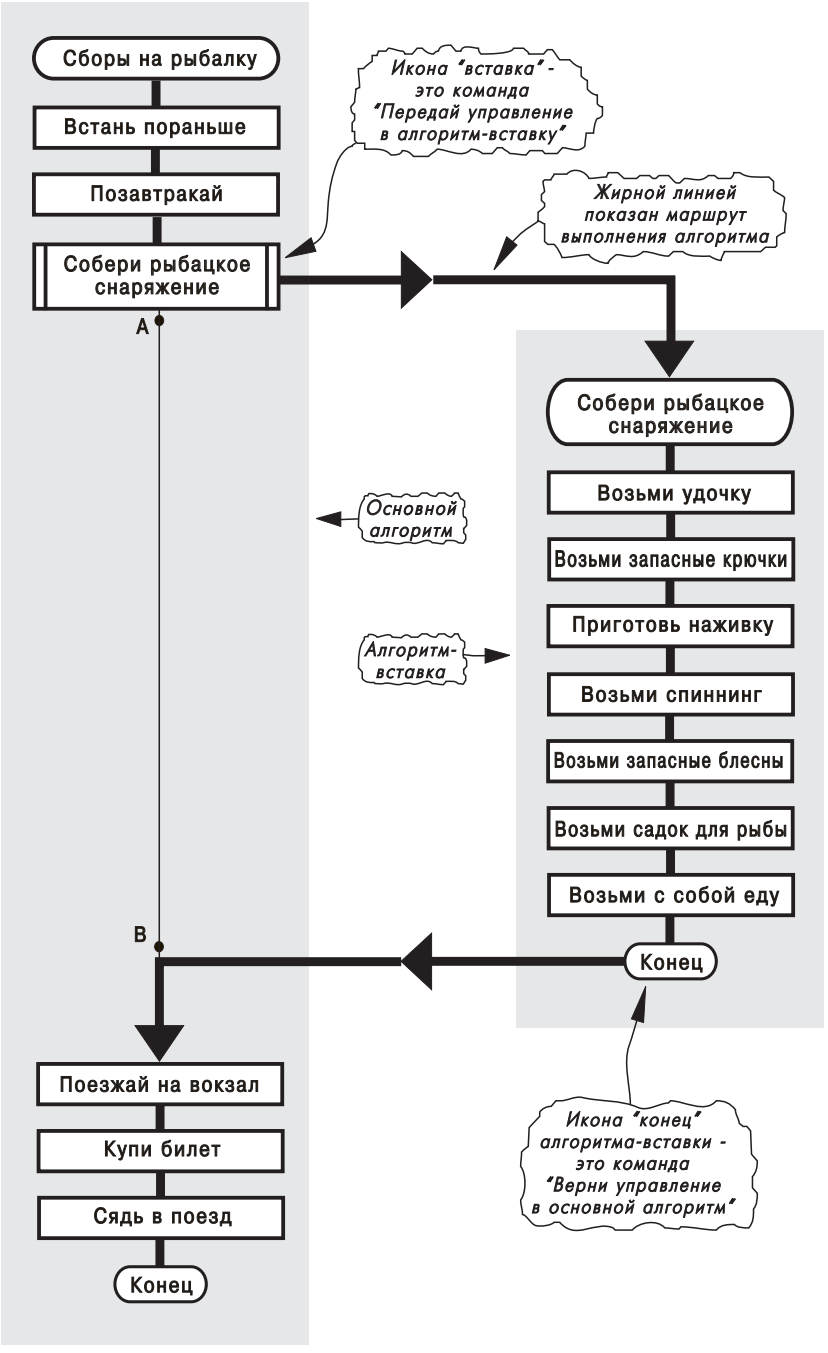
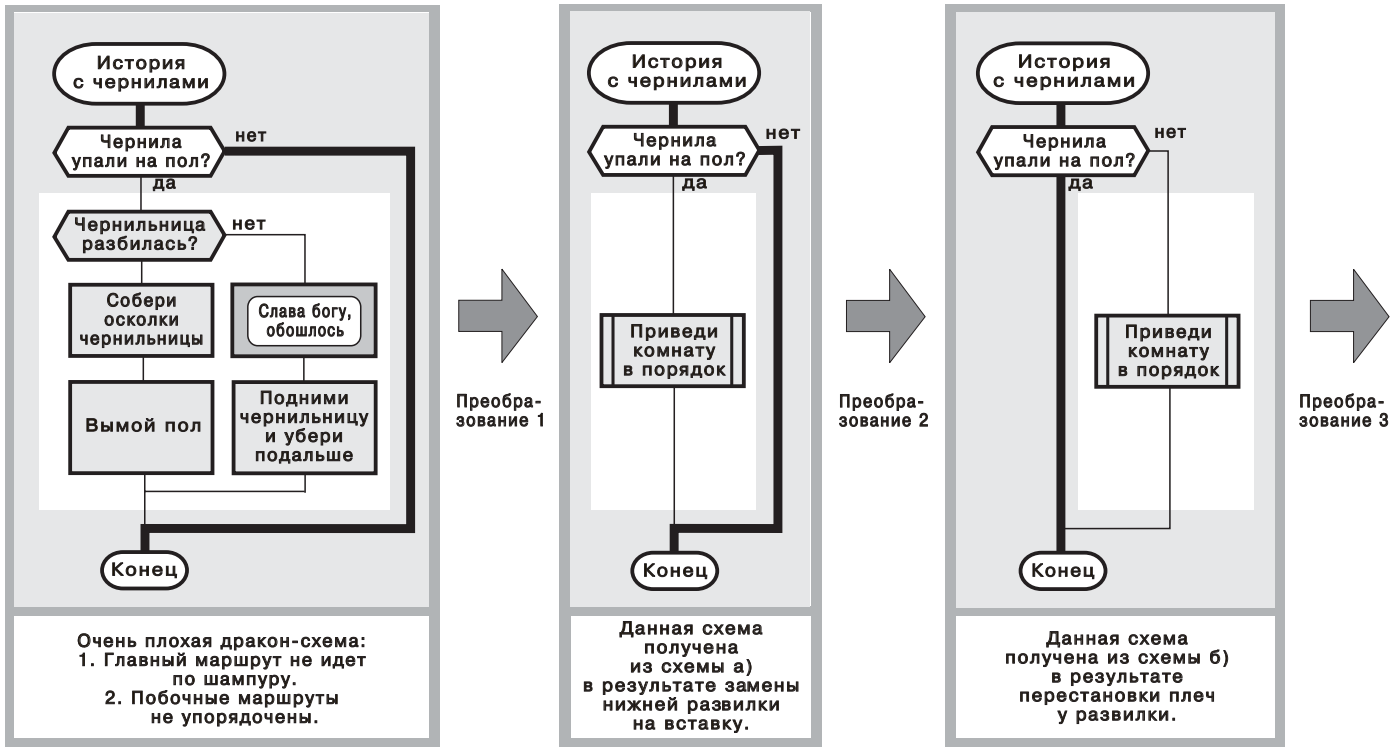


Рис. 30. **Как взаимодействуют основной алгоритм и алгоритм-вставка?**



а) *б)* *в)*

Рис. 31. **Цепочка эквивалентных преобразований, превращающих плохую дракон-схему в хорошую (эргономичную)**

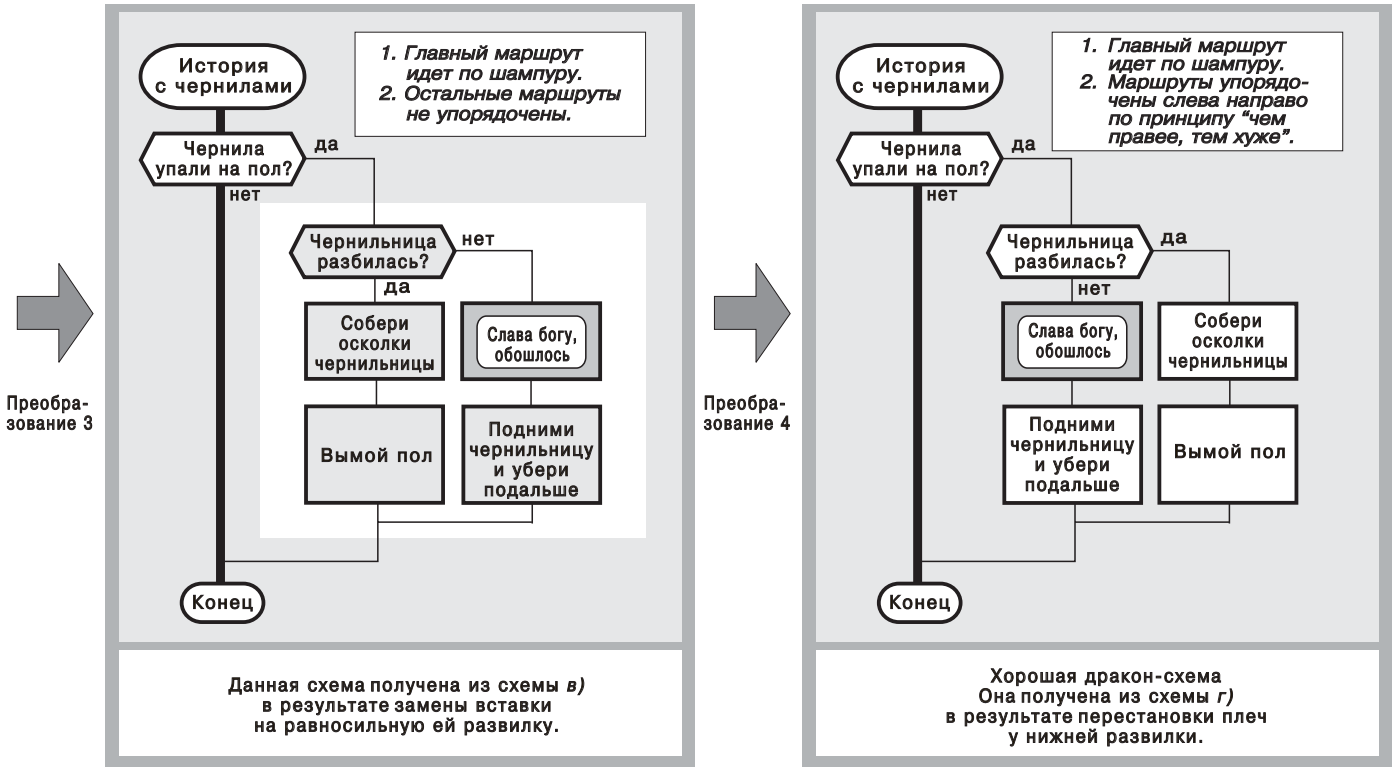


Рис. 31 (окончание)

УЛУЧШЕНИЕ ЭРГОНОМИЧНОСТИ АЛГОРИТМОВ С ПОМОЩЬЮ ЦЕПОЧКИ ЭКВИВАЛЕНТНЫХ ПРЕОБРАЗОВАНИЙ

Когда рисуешь алгоритм, нужно стремиться, чтобы он с самого начала удовлетворял правилам и был эргономичным. А если это не получилось? Если первый блин комом, как на рис. 31*а*? В этом случае необходимо довести алгоритм до ума с помощью последовательности эквивалентных преобразований. Вообще говоря, в примере на рис. 31*а* достаточно сделать всего две рокировки (преобразовав обе развилки) и мы сразу получим нужный ответ — искомую эргономичную схему на рис. 31*д*.

Однако, чтобы сделать изложение более наглядным, лучше избрать другой путь и двигаться к цели мелкими шажками. Для начала используем *визуальную формулу* на рис. 32 и с ее помощью заменим нижнюю развилку на рис. 31*а* на икону-вставку “Приведи комнату в порядок”. Результат подстановки представлен на рис. 31*б*. Затем выполним рокировку и получим схему на рис. 31*в*. После этого произведем обратную подстановку по формуле на рис. 32 и заменим икону-вставку на развилку “Чернильница разбилась?” Полученная схема показана на рис. 31*г*. Наконец, делаем еще одну рокировку и приходим к искомой эргономичной схеме на рис. 31*д*.

Для удобства читателя на рис. 33 дана общая сводка эквивалентных преобразований.

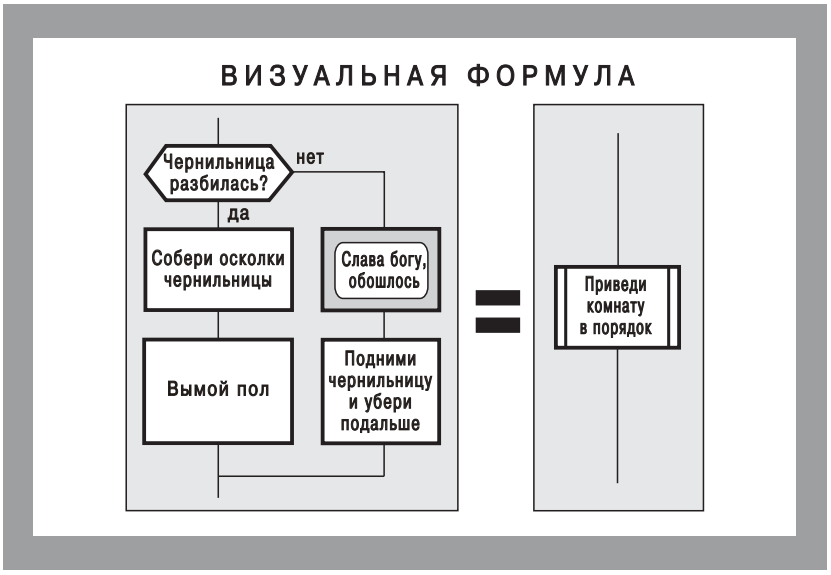


Рис. 32. **Пример операции “подстановка”**

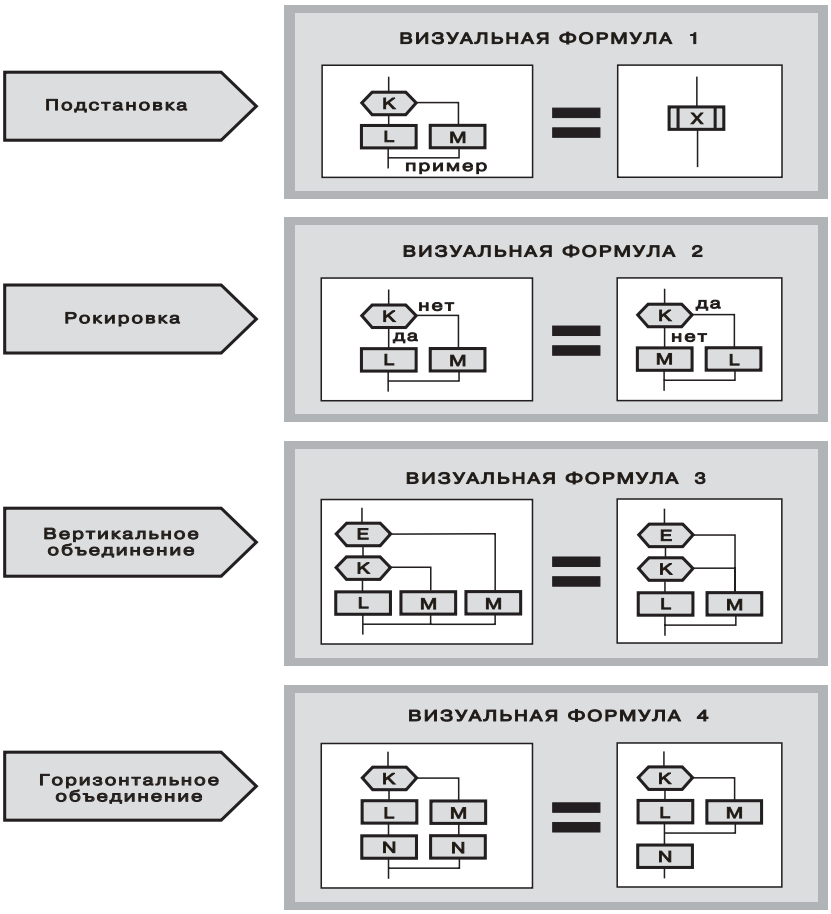


Рис. 33. **Эквивалентные преобразования алгоритмов**

ВЫВОДЫ

1. Понятие эргономичного алгоритма весьма актуально. Применение достижений эргономики к теории алгоритмов позволяет значительно улучшить понимаемость алгоритмов и программ.
2. Понятие эргономичного алгоритма задается с помощью конечного набора четко определенных правил и признаков, таких, как “главный маршрут должен идти по шампуру” и т. д. Следовательно, это понятие является строгим.
3. Рассмотренные выше четыре эквивалентных преобразования алгоритмов подтверждают, что эргономичность алгоритмов можно улучшить с помощью простых и ясных методов, которые в некотором смысле можно считать формальными.

Глава 8

ВИЗУАЛИЗАЦИЯ ЦИКЛОВ

Успешность принятия решения во многом зависит от способности человека “визуализировать проблемную ситуацию”, наглядно представлять ее и оперировать наглядными образами.

Наталья Завалова, Борис Ломов, Владимир Пономаренко

ОБЫЧНЫЙ ЦИКЛ

В языке ДРАКОН имеется следующий ассортимент циклов:

- обычный цикл;
- переключающий цикл;
- цикл ДЛЯ;
- веточный цикл;
- цикл ЖДАТЬ.

Первые четыре цикла рассматриваются в этой главе, цикл ЖДАТЬ — в гл. 11.

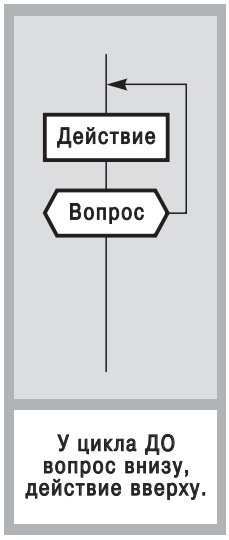


Рис. 34. **Цикл ДО**

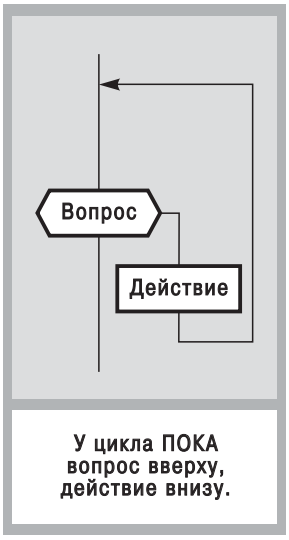


Рис. 35. **Цикл ПОКА**

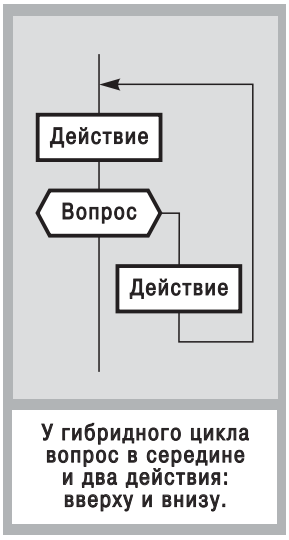


Рис. 36. **Гибридный цикл**

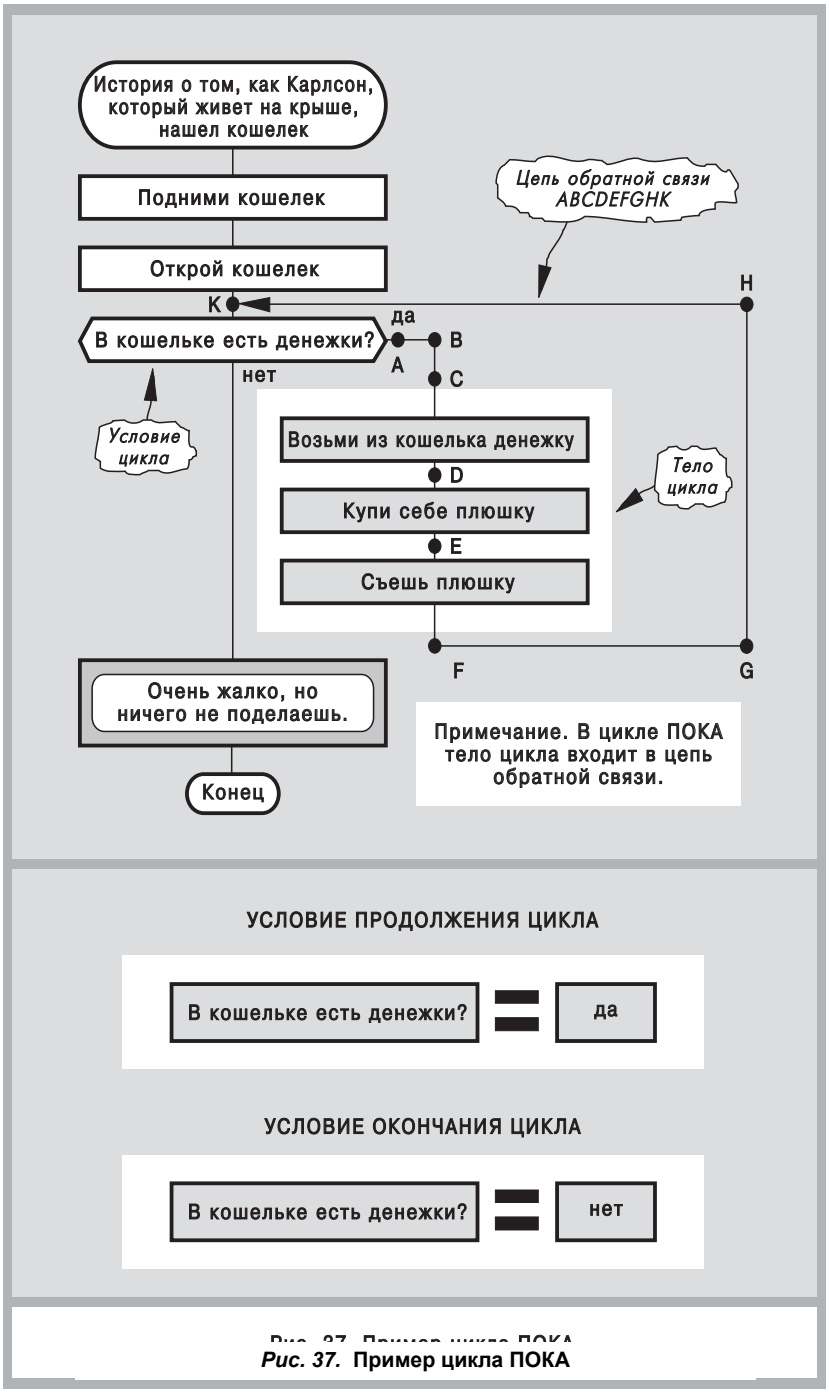


Рис. 37. **Пример цикла ПОКА**

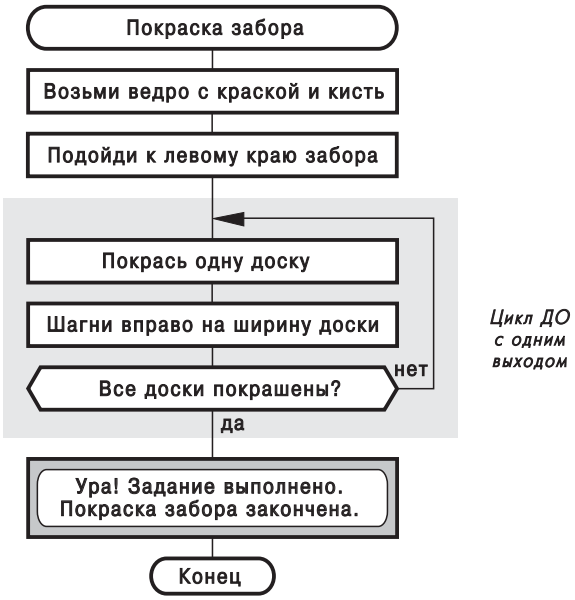


Рис. 38. **Пример цикла ДО**

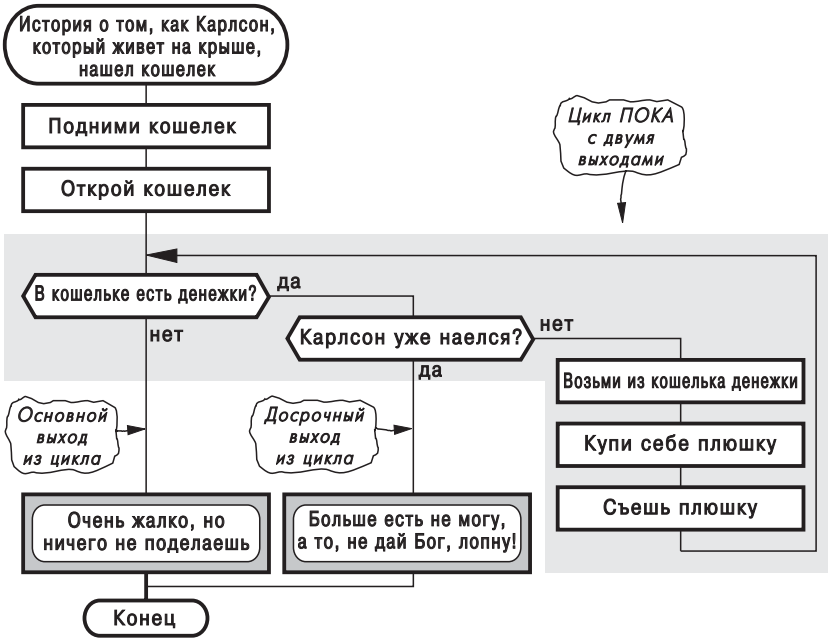


Рис. 39. **Досрочный выход из цикла происходит потому, что Карлсон больше не хочет есть** ***(сравни с рис. 37)***

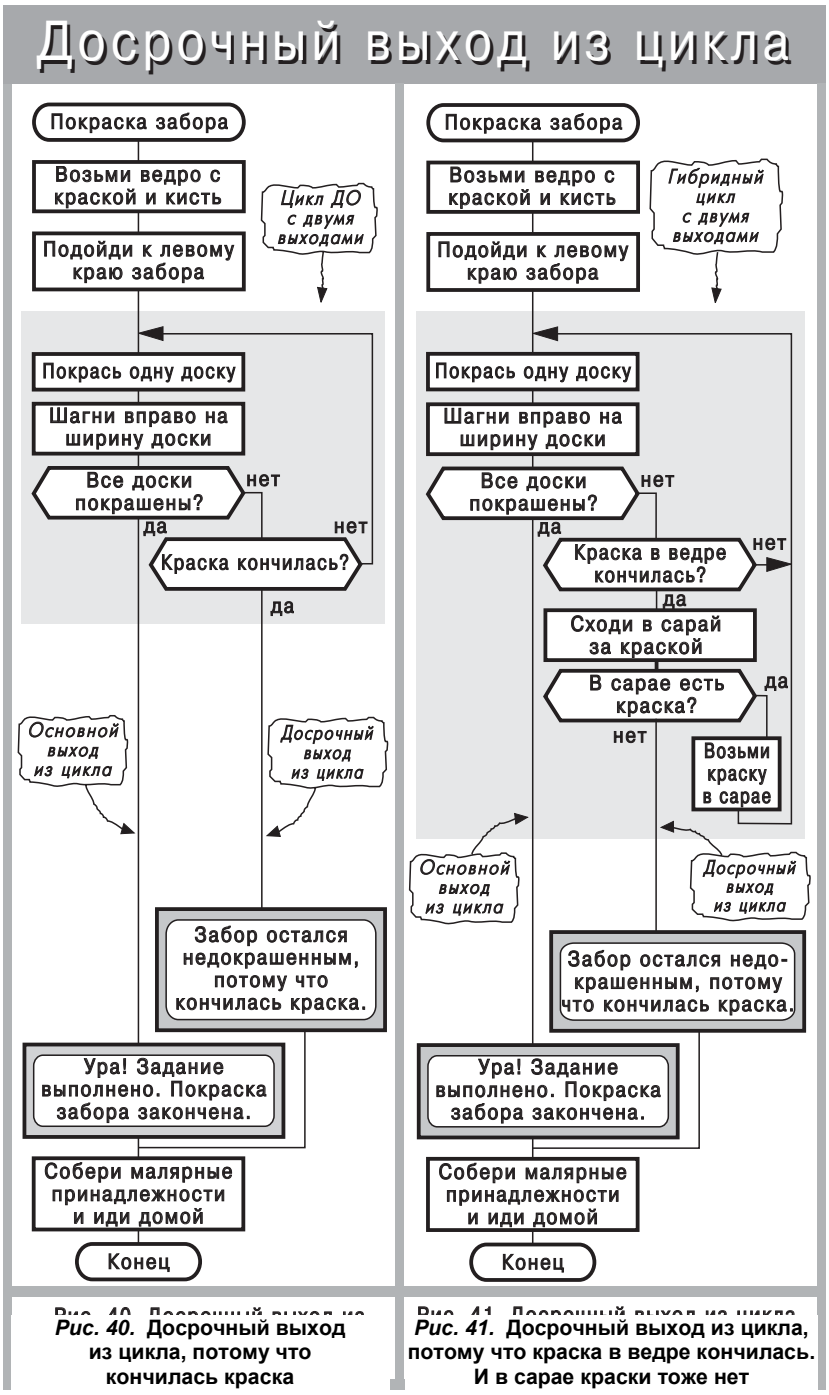


Рис. 40. **Досрочный выход из цикла, потому что кончилась краска**

Рис. 41. **Досрочный выход из цикла, потому что краска в ведре кончилась. И в сарае краски тоже нет**

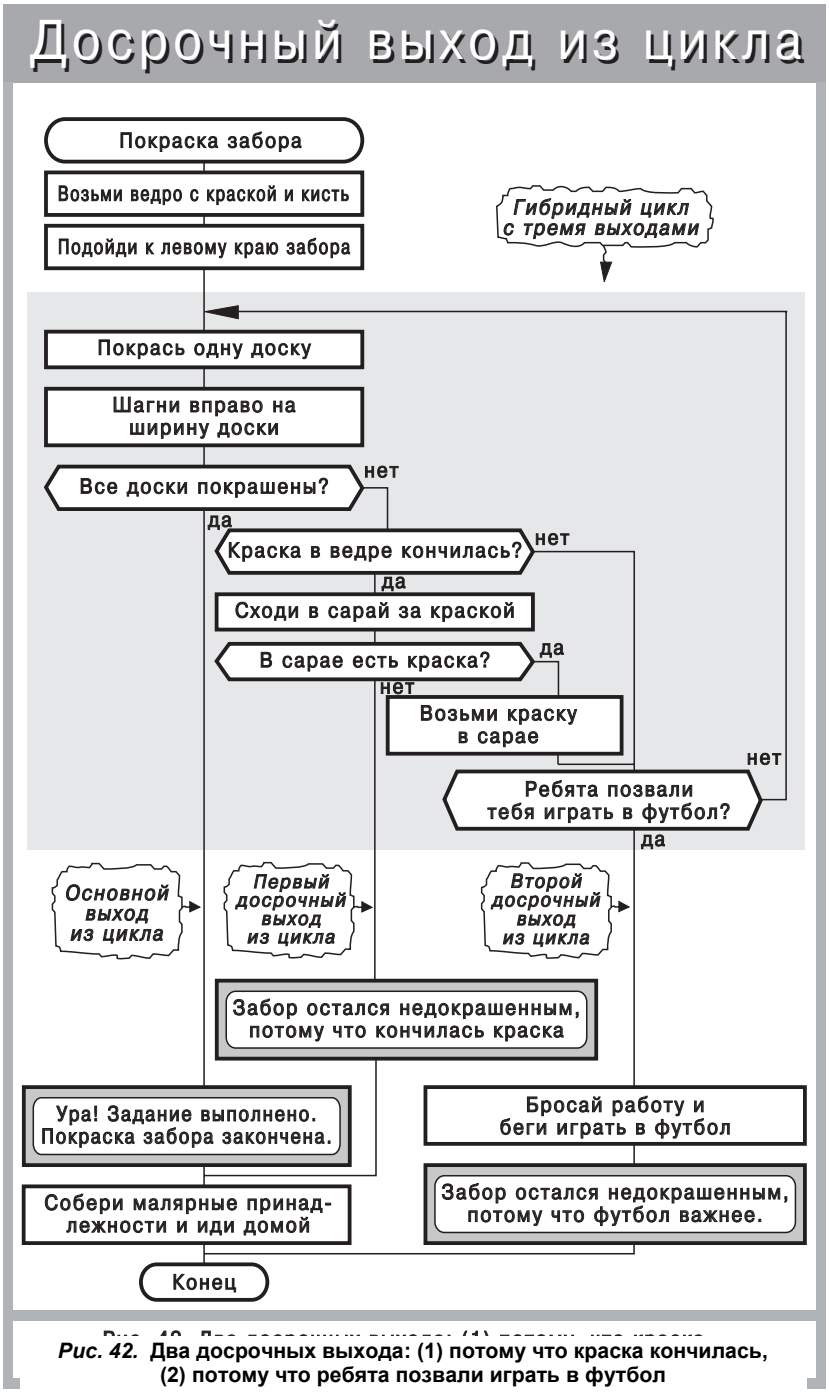


Рис. 42. **Два досрочных выхода: (1) потому что краска кончилась, (2) потому что ребята позвали играть в футбол**

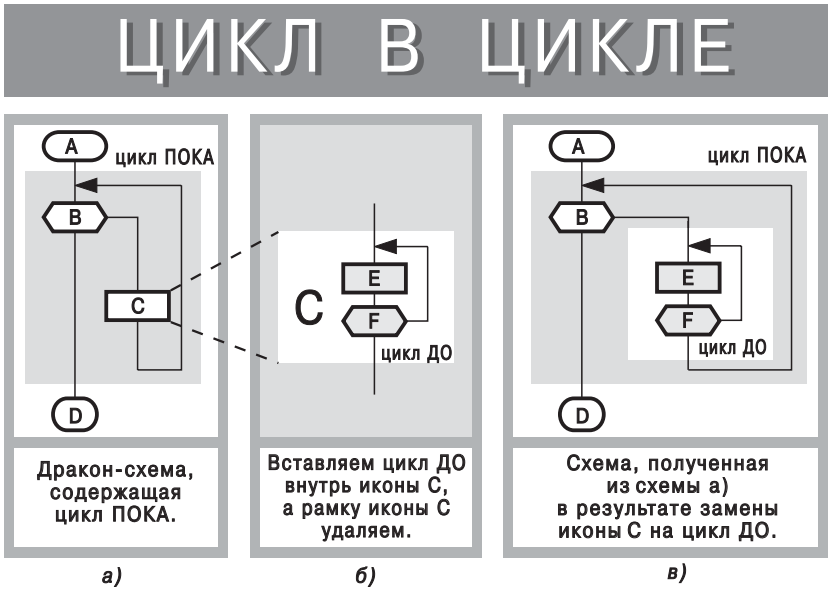


Рис. 43. **Как построить цикл в цикле**

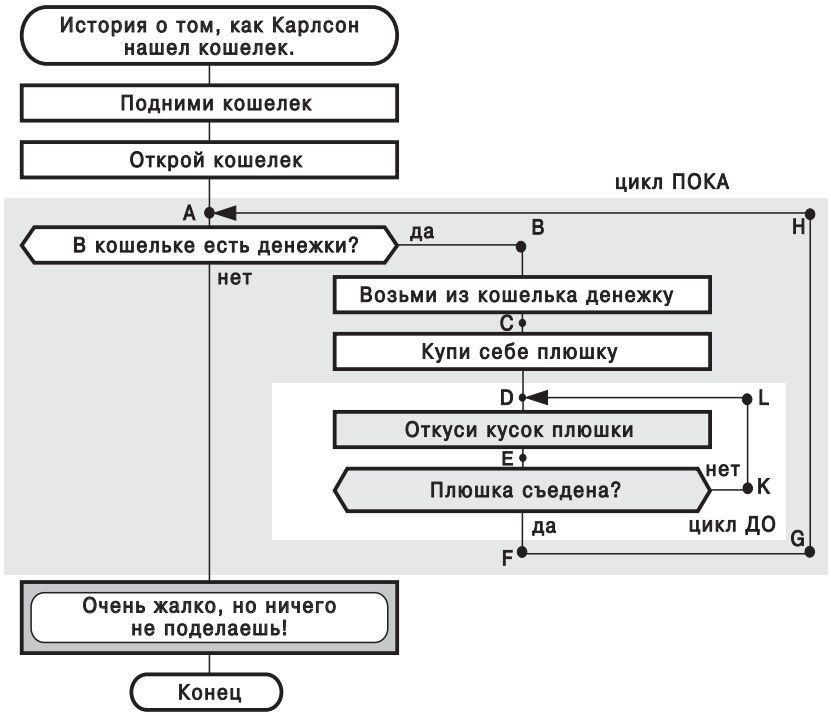


Рис. 44. **Цикл в цикле. Внутри цикла ПОКА находится цикл ДО** ***(сравни с рис. 37)***

Рис. 45. **Четыре варианта конструкции “цикл в цикле”**

Составной визуальный оператор “обычный цикл” (рис. 2, макроикона 4) содержит иконы “вопрос” и “петля цикла” (рис. 1, иконы И4, И24). Он охватывает циклы трех типов (рис. 34—36):

- цикл ДО *(do-while)*,
- цикл ПОКА *(while-do)*,
- гибридный цикл *(do-while-do)*.

Примеры циклов ПОКА и ДО приведены на рис. 37, 38. *Досрочный выход* из цикла показан на рис. 39—42. Конструкция “цикл в цикле” представлена на рис. 43—45.

Анализируя рисунки, можно заметить следующие особенности.

- Оператор “обычный цикл” имеет один вход и один или несколько выходов.
- Цикл с одним выходом представляет собой шампур-блок (вход и выход находятся на одной вертикали).
- Если цикл имеет более одного выхода, основной выход размещается на главной вертикали, дополнительные — правее ее.
- Петля цикла находится правее главной вертикали и закручена против часовой стрелки.
- Икона “вопрос” задает *условие цикла*, которое распадается на две части: условие продолжения и условие окончания (рис. 37).

! *Условие продолжения* соответствует правому выходу иконы “вопрос”, *условие окончания* — нижнему.

! Условие окончания может помечаться как словом “нет”, так и словом “да”. То же самое относится и к условию продолжения.

ПЕРЕКЛЮЧАТЕЛЬ И ПЕРЕКЛЮЧАЮЩИЙ ЦИКЛ

Предположим, в алгоритме нужно организовать разветвление на несколько направлений. Задачу можно решить двумя способами: с помощью иконы “вопрос” (рис. 46*а*) и с помощью переключателя (рис. 46*б*).

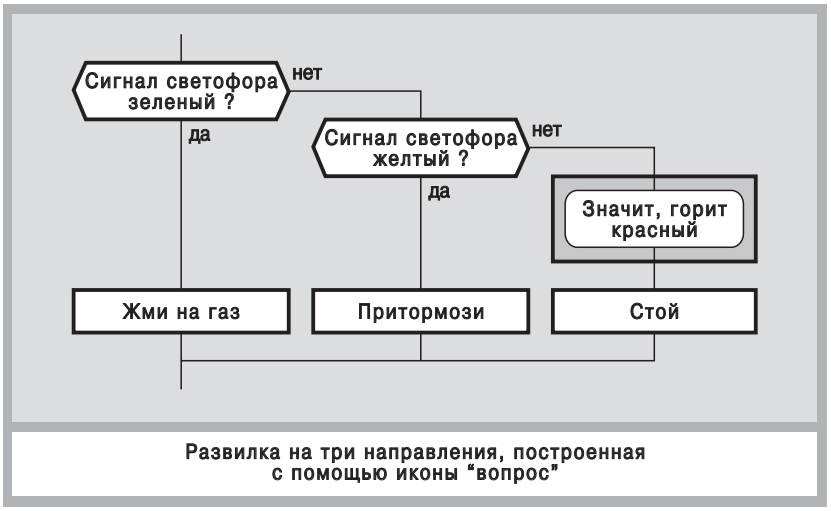
Переключатель — составной визуальный оператор (рис. 2, макроикона 3), имеющий один вход и один выход, содержащий одну икону “выбор” и несколько (две и более) икон “вариант” (рис. 1, иконы И5, И6). Внутри иконы “выбор” делается надпись, обычно в утвердительной форме, которая обозначает вопрос, имеющий строго определенное число ответов (два и более). Ответы записываются в иконах “вариант”. Таким образом, число вариантов равно числу ответов. Говоря формально, в иконе “выбор” записывается переменная, в иконах “вариант” — ее значения. На рис. 46*б* переменная “Светофор” принимает три значения: зеленый, желтый, красный.

Переключатель позволяет создать особый тип цикла — *переключающий цикл* (рис. 2, макроикона 5). Для этого нужно оторвать выход правой ветви переключателя, загнуть его вверх и присоединить стрелку в нужное место (рис. 47).

На рис. 48 изображен цикл с переключателем, однако это не переключающий цикл, а обычный. Как их отличить? В первом случае переключатель имеет два выхода, во втором — только один. Есть еще одно отличие. Если вверх загибается выход иконы “вопрос” — это обычный цикл (ДО, ПОКА или гибридный). А если кверху идет выход переключателя — перед нами переключающий цикл.

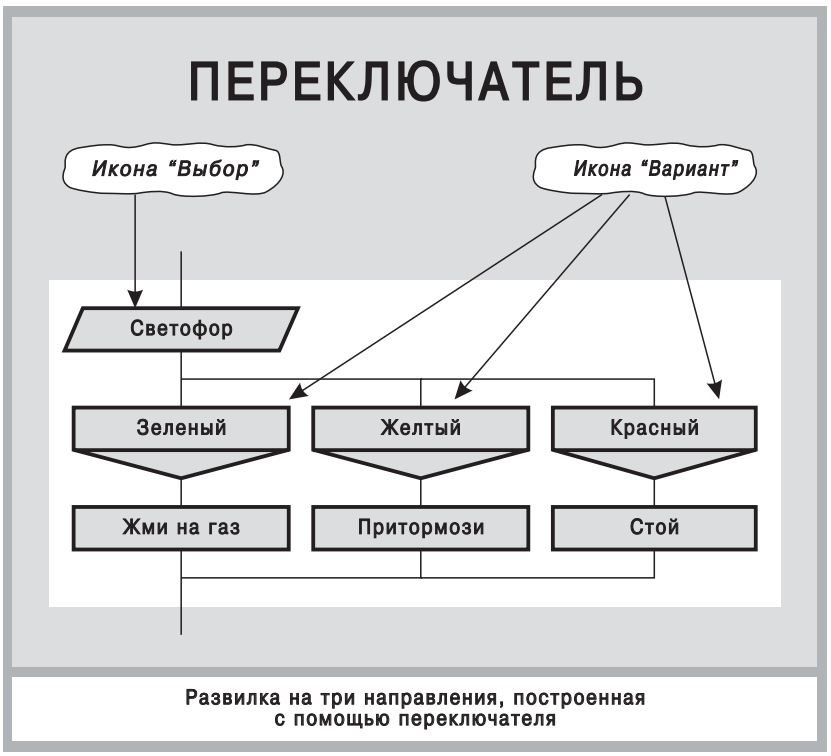
ЦИКЛ ДЛЯ

На рис. 49 и 50 показаны два варианта решения простой математической задачи. В первом случае используется цикл ДО, во втором — цикл ДЛЯ. Цикл ДЛЯ — составной визуальный оператор (рис. 2, макроикона 6), содержащий иконы “начало цикла ДЛЯ” и “конец цикла ДЛЯ” (рис. 1, иконы И12, И13), между которыми располагаются одна или несколько других икон. Внутри иконы “начало цикла ДЛЯ” указываются переменная цикла, ее начальное и конечное значения и шаг. Порядок



Развилка на три направления, построенная с помощью иконы “вопрос”

а)



Развилка на три направления, построенная с помощью переключателя

б)

Рис. 46. **Что лучше: икона “вопрос” или переключатель?**

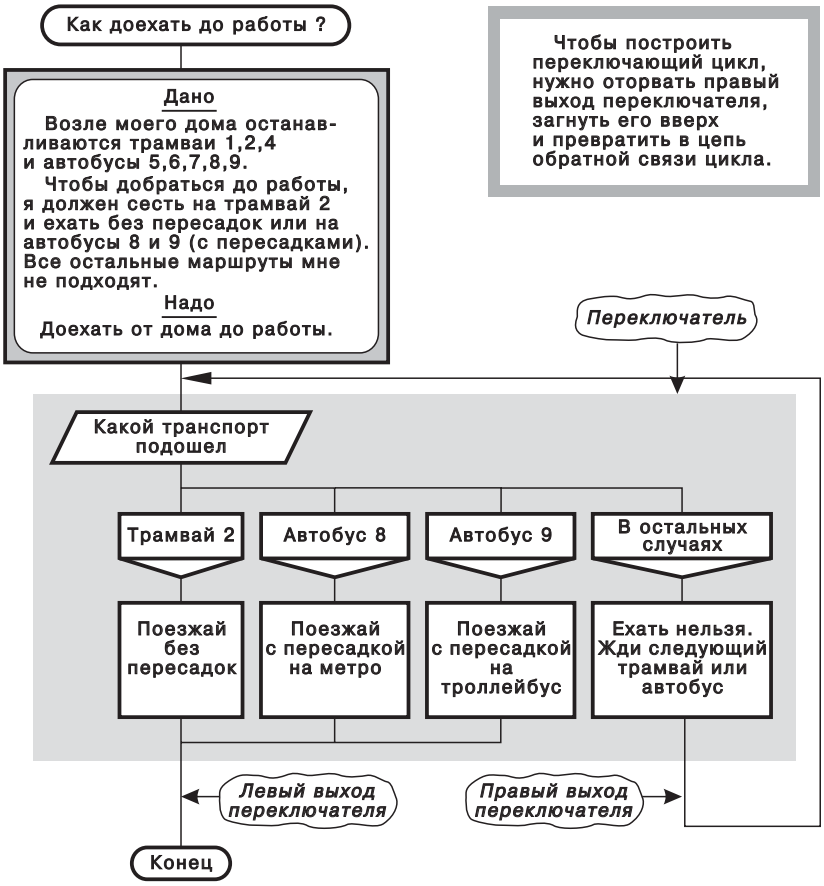


Рис. 47. **Как построить переключающий цикл?**

записи этих величин определяется выбранным вариантом текстового синтаксиса. На рис. 50 изображен вариант, по умолчанию принимающий, что шаг равен 1.

ВЕТОЧНЫЙ ЦИКЛ

Циклы, описанные выше, могут использоваться как в примитиве, так и в силуэте. В этом параграфе речь пойдет о веточном цикле, который встречается только в силуэте.

Веточный цикл образуется, когда метка в иконе “адрес” указывает либо на свою ветку, либо на ветку, которая находится левее. Например, икона-адрес “Покупка плюшек” на рис. 51 указывает на свою ветку. Внутри веточного цикла могут появляться циклы других типов. На рис. 52 изображена конструкция “цикл в цикле”, у которой внутри веточного цикла находится цикл ДО.

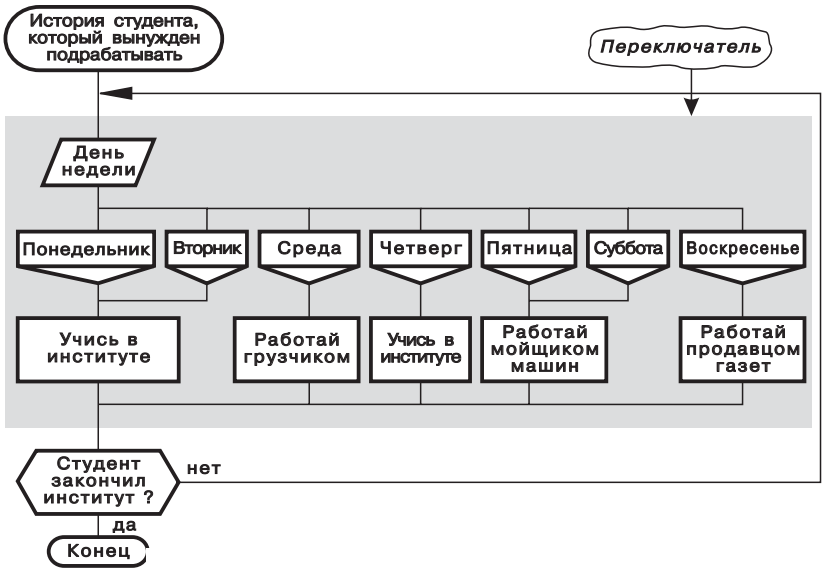


Рис. 48. **Цикл ДО, в котором есть переключатель**

Рис. 49. **Как вычислить факториал X = n! с помощью цикла ДО?**

Рис. 50. **Как вычислить факториал X = n! с помощью цикла ДЛЯ?**

Рис. 51. **Веточный цикл с досрочным выходом** ***(сравни с рис. 39)***

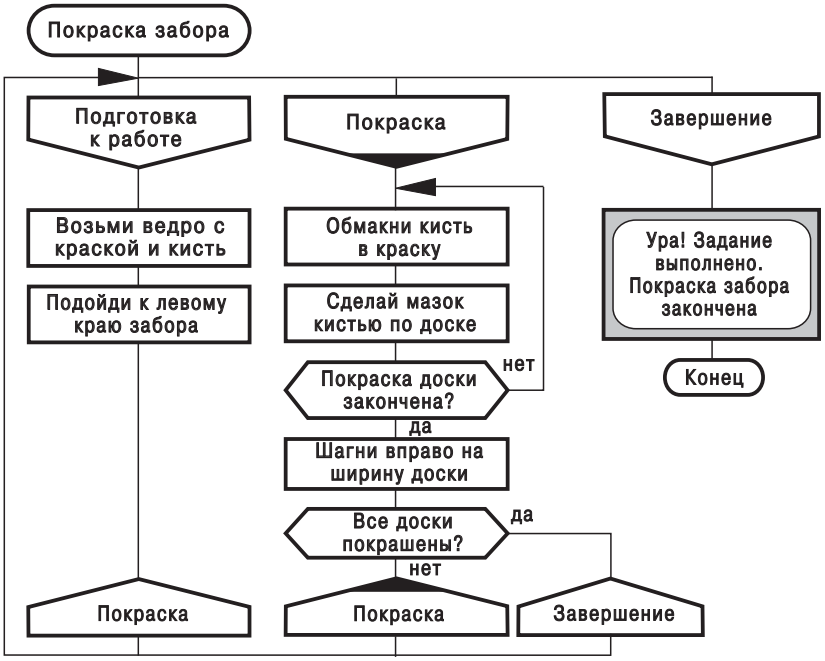


Рис. 52. **Цикл ДО внутри веточного цикла**

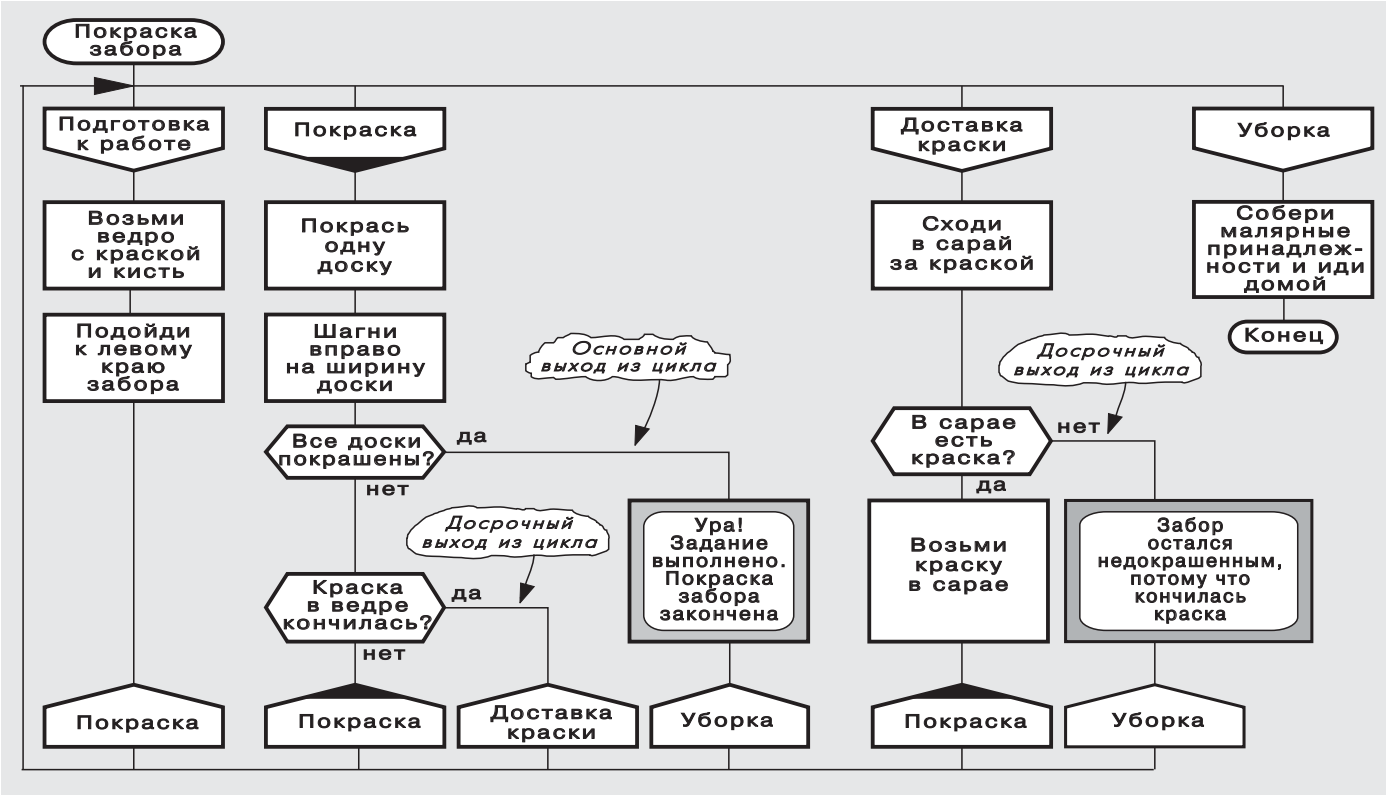


Рис. 53. **Веточный цикл со сложным досрочным выходом** ***(сравни с рис. 41)***

При замене примитива на эквивалентный ему силуэт гибридный цикл нередко превращается в веточный цикл. В этом легко убедиться, сравнив эквивалентные алгоритмы на рис. 41 и 53.

ГЛАВНЫЙ МАРШРУТ СИЛУЭТА

В этом параграфе мы продолжим изучение веточных циклов и попытаемся ответить на вопрос: как найти главный маршрут веточного цикла? Для этого нужно проанализировать понятие “главный маршрут силуэта” (рис. 54).

Линейный (неразветвленный) силуэт имеет один-единственный маршрут, который и является главным. Он проходит по шампурам всех веток и по всем иконам силуэта (рис. 54*а*).

Формула маршрута для силуэта имеет особенность: одноименные иконы “адрес” и “имя ветки” обозначаются одной буквой, которая повторяется в формуле дважды. Например, силуэт на рис. 54*а* имеет формулу

ABEFGC — CHID — DJKM

где парные буквы обозначают переход с первой ветки на вторую (*C — C*) и со второй на третью (*D — D*).

Ветка называется *одноадресной*, если она имеет одну икону “адрес”. Если все ветки одноадресные, силуэт считается одноадресным.

Линейный силуэт всегда одноадресный. Однако одноадресный силуэт может быть и разветвленным. В последнем случае его главный маршрут следует по шампурам всех веток, однако он не проходит по всем иконам (рис. 54*б*).

Если хотя бы одна ветка имеет более одного адреса, может сложиться ситуация, когда какие-то ветки не попадают на главный маршрут. На рис. 54*в* икона-адрес *D* лежит на побочном маршруте. Это приводит к тому, что ветка *D* также оказывается на побочном маршруте. В результате главный маршрут проходит по шампурам всех веток, кроме ветки *D* (рис. 54*в*).

Веточные циклы образуются только в многоадресных силуэтах, в одноадресных их в принципе не может быть. Они бывают нескольких типов:

- одноветочные (если цикл помещается в одной ветке);
- двухветочные (если цикл занимает две ветки);
- трехветочные (цикл в трех ветках)

и т. д.

Как работает одноветочный цикл? Предположим, до начала выполнения цикла на рис. 54*г* имеют место условия

D = нет

E = да

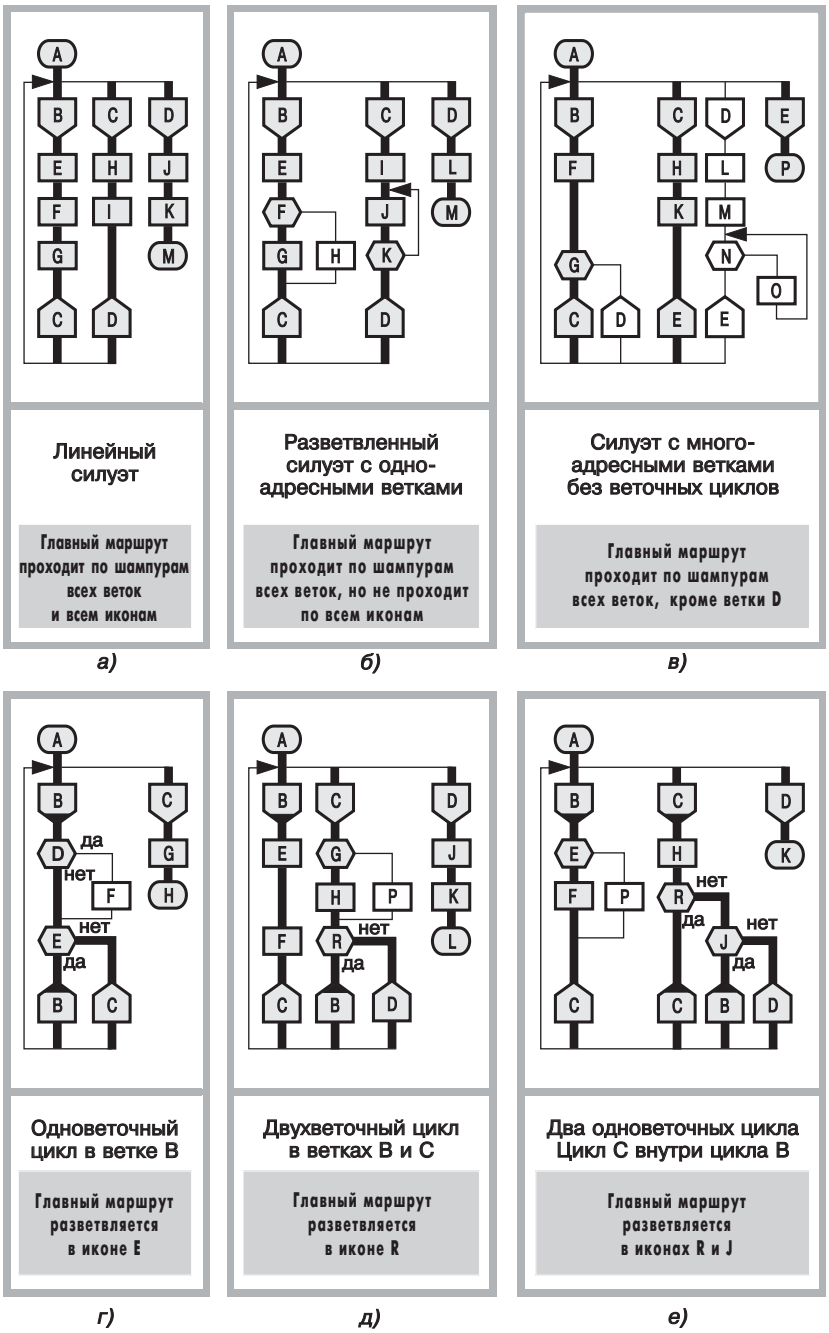
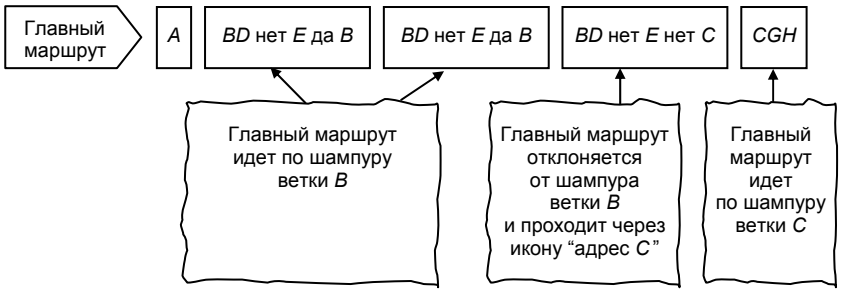


Рис. 54. **Главный маршрут силуэта показан жирной линией. Если в силуэте есть веточный цикл, главный маршрут может разветвляться** ***(см. г, д, е)***

Предположим также, что веточный цикл выполняется два раза, после чего условие E принимает значение “нет”. Это значит, что при третьем проходе по ветке *B* произойдет выход из цикла по пути “*E* нет *C*”. В такой ситуации формула главного маршрута для силуэта на рис. 54*г* принимает вид:



Как выглядит главный маршрут на дракон-схеме? Ответ изображен жирной линией на рис. 54*г*. Мы видим, что главный маршрут как бы разветвляется в иконе *E* и проходит через оба ее выхода. Разумеется, это условность, которая означает следующее. Сначала (когда *E* = да) главный маршрут идет по шампуру, затем (когда выполняется условие окончания цикла *E* = нет) главный маршрут проходит через правый выход иконы *E*.

Чтобы построить одноветочный цикл, нужно в левой иконе “адрес” записать *X*, где *X* — имя данной ветки. Для выхода из цикла следует добавить вторую икону “адрес” и записать в ней *Y*, где *Y* — имя следующей (по порядку исполнения) ветки.

Если в веточном цикле слишком много икон, он может не поместиться в одной ветке. К счастью, его можно разделить на части. Например, веточный цикл на рис. 54*д* содержит пять икон: *E*, *F*, *G*, *H*, *R* (иконы “имя ветки” и “адрес” не в счет). Поместим иконы *E* и *F* в ветку *B*, а иконы *G*, *H*, *R* — в ветку *C*. В результате цикл станет двухветочным. Главный маршрут силуэта с двухветочным циклом имеет разветвление в иконе *R*. Условие *R* = да позволяет вернуться к началу цикла. Если *R* = нет, главный маршрут ведет нас к концу алгоритма (рис. 54*д*).

Таким образом, двухветочный цикл — это цикл, содержащий две ветки *X* и *Y*, причем в ветке *X* имеется икона-адрес *Y*, а в ветке *Y* — икона-адрес *X*.

На рис. 54*е* представлена ситуация “цикл в цикле”: веточный цикл *C* находится внутри веточного цикла *B*. Из рисунка видно, что в этом случае главный маршрут “разветвляется” дважды: в иконах *R* и *J*.

Если выполняется условие *R* = да, происходит повторение внутреннего цикла *C*. При сочетании условий

R = нет	*J* = да

производится выход из цикла *C* и повторение внешнего цикла *B*. Наконец, сочетание условий

означает, что выполнение цикла *B* и алгоритма в целом заканчивается.

ВЫВОДЫ

1. В различных текстовых языках при описании циклов применяются разные наборы ключевых слов, имеющих к тому же разную семантику. Неразбериху усугубляют отличия в логике окончания цикла. Например, в языке Си для циклов *while* и *do-while* условие окончания цикла соответствует значению *false* или 0, условие продолжения — значению *true* или 1. В языке Паскаль картина иная: в цикле *while-do* выход из цикла соответствует значению *false*, а в цикле *repeat-until* по каким-то загадочным причинам применяется диаметрально противоположный принцип: выход из цикла производится, когда логическое выражение принимает значение *true*. Все эти путаные правила программист обязан знать и неукоснительно выполнять.
2. Отсутствие унификации ключевых слов и разнобой в определении условий выхода из цикла является серьезным недостатком: программисты вынуждены зубрить ключевые слова и значения условий, причем освоение каждого следующего языка требует новой зубрежки.
3. С точки зрения визуального программирования, указанные трудности являются надуманными и легко устраняются. Надо лишь отказаться от сложившихся привычек и устаревших стереотипов мышления, связанных с текстовым программированием. Визуализация качественно меняет ситуацию, поскольку текст больше не является единственным носителем информации.
4. Визуальные образы уменьшают нагрузку на память программиста, ликвидируют ошибки, вызванные неправильным пониманием семантики ключевых слов, отменяют ненужные ограничения, предоставляют пользователю богатую палитру выразительных средств и в конечном итоге обеспечивают более высокую понимаемость алгоритмов и программ.
5. Визуализация циклов — весьма полезный инструмент, так как сложные вложенные циклы со многими выходами часто бывают источником трудных ошибок. Многие из них возникают из-за путаницы, связанной с устаревшей привычкой описывать циклы словами. Сегодня никто не пытается заменить конструкторские и строительные чертежи словесными описаниями. По мнению автора, текстовая форма записи циклов во многих случаях является таким же анахронизмом, как словесное описание механического чертежа или электрической схемы.

Глава 9

ВИЗУАЛИЗАЦИЯ ЛОГИЧЕСКИХ ФОРМУЛ

Существует форма представления информации наглядная, броская, понятная всем с детства. Такой формой является графика.

Валерий Венда

ВИЗУАЛИЗАЦИЯ ФУНКЦИИ И

— Где можно купить щенка?

— В нашем городке они продаются на рынке, но сегодня рынок закрыт. К тому же щенков продают не каждый день. Щенки довольно дорогие и какие-то невзрачные — не знаю, понравятся ли они вам.

Из подслушанного разговора ясно, что покупка щенка возможна в том и только в том случае, когда выполняются четыре условия (рис. 55):

- рынок открыт (обозначим это условие через P);
- у покупателя деньги есть (Q);
- щенки есть в продаже (R);
- щенок понравился (S).

В итоге получаем логическую функцию

$$X = P \text{ и } Q \text{ и } R \text{ и } S$$

где X означает “Можно купить щенка?” (рис. 55).

В традиционных языках программирования значениями логических переменных считаются пары (ИСТИНА, ЛОЖЬ) или (1, 0). С эргономической точки зрения, такой подход нельзя признать удачным. В самом деле, использование “шибко мудреных” слов ИСТИНА и ЛОЖЬ или таинственных цифр 1 и 0 в примере о щенках (как и в любом другом конкретном примере) является надуманным, дезориентирующим и не содействует пониманию существа вопроса.

Чтобы поправить дело, в качестве значений логических переменных и логических функций гораздо лучше выбрать простые и ясные слова “да” и “нет”, семантика которых не требует пояснений и понятна даже ребенку. Исходя из этого, в языке ДРАКОН логические функции, переменные и выражения рассматриваются как да-нетные вопросы. *Логическая функция И* определяется как функция, которая принимает значение

Рис. 55. **Логическая переменная “Можно купить щенка?” является логической функцией четырех логических переменных, связанных операцией “И”**

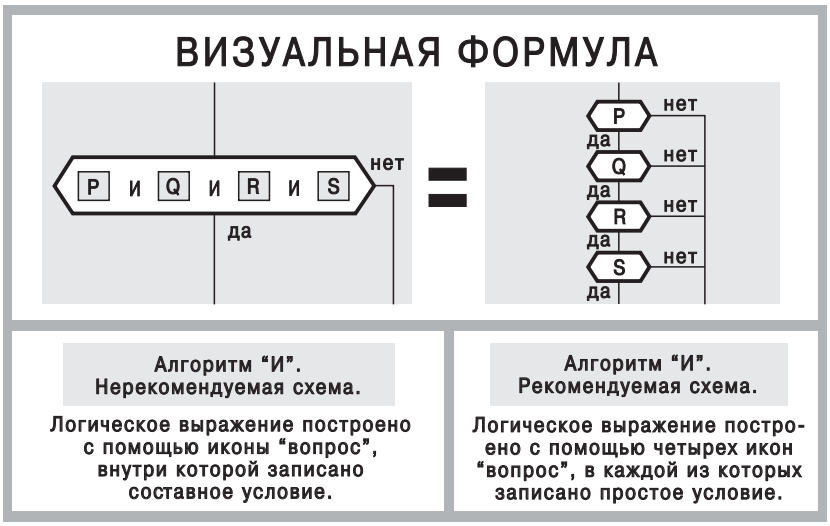


Рис. 56. **Рисуйте дракон-схему “И”, как показано справа. Избегайте нерекомендуемых схем**

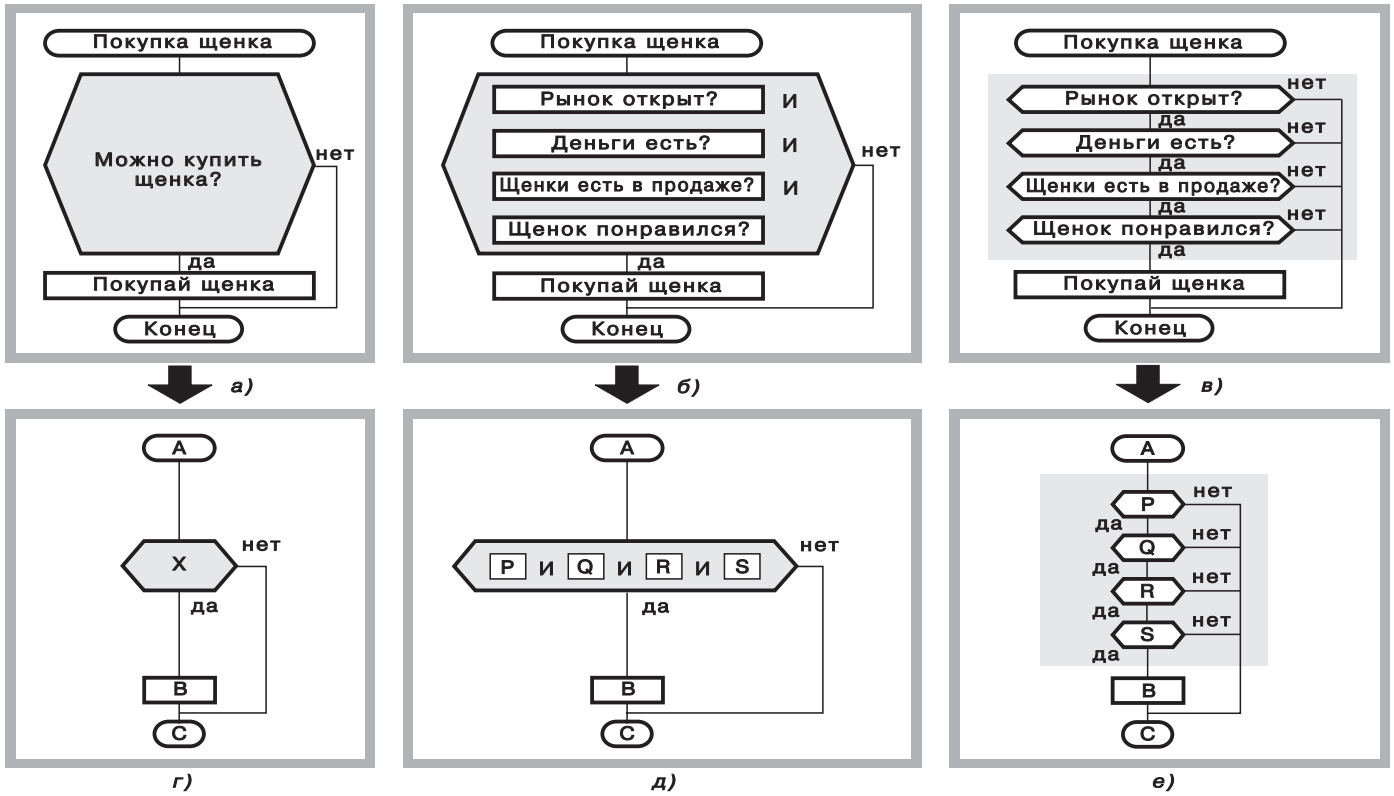


Рис. 57. **Примеры алгоритмов с операцией “И”**

Рис. 58. **Логическая переменная “Петя заболел?” является логической функцией четырех логических переменных, связанных операцией “ИЛИ”**

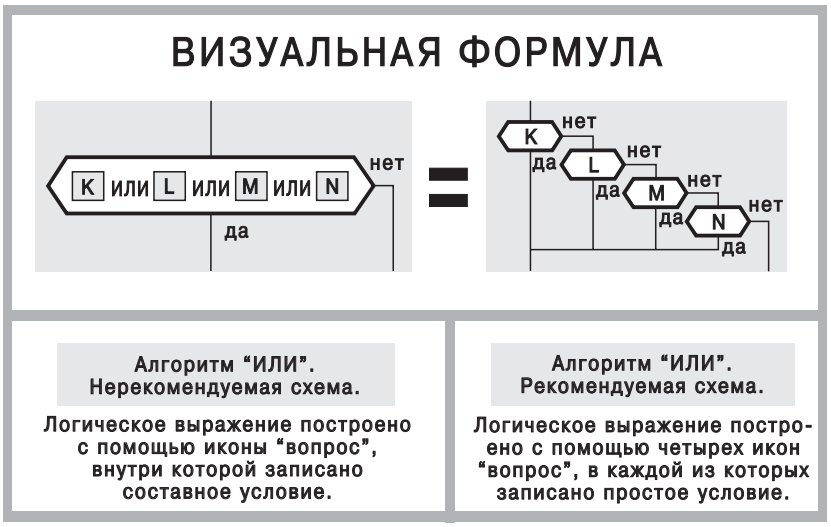


Рис. 59. **Рисуйте дракон-схему “ИЛИ”, как показано справа. Избегайте нерекомендуемых схем**

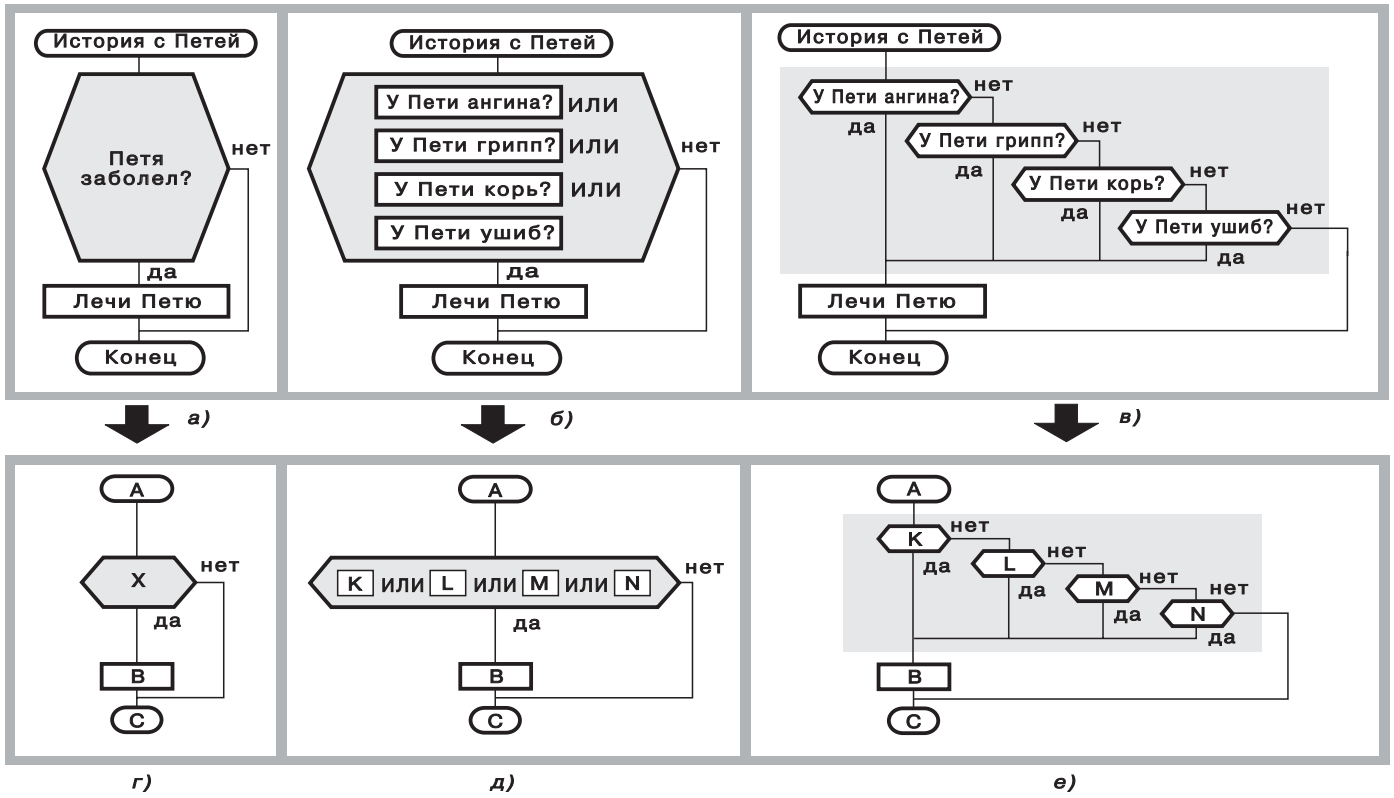


Рис. 60. **Примеры алгоритмов с операцией “ИЛИ”**

“да”, если все логические переменные имеют значение “да”. В остальных случаях функция приобретает значение “нет” (рис. 55)[1].

Существуют два способа изображения функции И на языке ДРАКОН: *текстовый* и *визуальный*. В первом случае используют одну икону “вопрос”, внутри которой пишут логическое выражение, состоящее из логических переменных, соединенных знаками логической операции И (рис. 56 слева). В другом случае на одной вертикали рисуют N икон “вопрос”, где N — число логических переменных, причем в каждой иконе записывают одну логическую переменную (рис. 56 справа).

Визуальная формула на рис. 56 показывает, что оба способа эквивалентны. Примеры на рис. 57 подтверждают это. Для практического использования рекомендуется визуальный способ, так как он более нагляден и позволяет быстрее найти ошибку в сложном алгоритме. Следует подчеркнуть, что текстовый способ не является запрещенным, но пользоваться им следует с осторожностью и лишь в тех случаях, когда пользователь убежден в своих способностях гарантировать отсутствие ошибок. Опыт показывает, что большинство людей выбирает визуальный способ как более легкий. Однако подготовленные специалисты, знакомые с основами математической логики, иногда предпочитают текстовый метод. Таким людям можно посоветовать освоить оба метода.

ВИЗУАЛИЗАЦИЯ ФУНКЦИИ ИЛИ

Логическая функция ИЛИ принимает значение “да”, если хотя бы одна логическая переменная имеет значение “да”. Функция принимает значение “нет”, если все логические переменные имеют значение “нет” (рис. 58).

На языке ДРАКОН функцию ИЛИ можно записать двумя способами. Если выбран текстовый способ, рисуют одну икону “вопрос”, содержащую логическое выражение (рис. 59 слева). При визуальном способе используют несколько икон “вопрос”, у которых объединяют нижний выход; в каждой иконе пишут одну логическую переменную (рис. 59 справа). Из рис. 59 и 60 видно, что оба метода эквивалентны.

ВИЗУАЛИЗАЦИЯ ФУНКЦИИ НЕ

Функция $W = \overline{Z}$ называется *функцией НЕ*, если логические переменные Z и W принимают инверсные значения, т. е. удовлетворяют условиям:

если Z = да, то W = нет;
если Z = нет, то W = да.

---

[1] Здесь необходимо уточнение. В дракон-схемах слова “да” и “нет” записываются только у выходов иконы “вопрос” и больше нигде. В следующей главе будет показано, что язык ДРАКОН не нуждается в специальных обозначениях для значений логических переменных. Использование слов “да” и “нет” в качестве значений переменных — это скорее педагогический прием для облегчения объяснений, а не принадлежность языка.

Рис. 61. **Визуальные формулы, позволяющие освободиться от знака логического отрицания (верхней черты)**

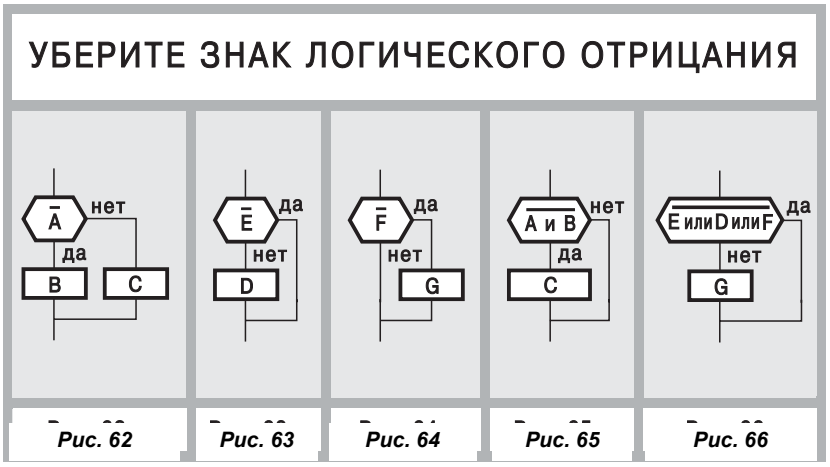


Рис. 62 ***Рис. 63*** ***Рис. 64*** ***Рис. 65*** ***Рис. 66***

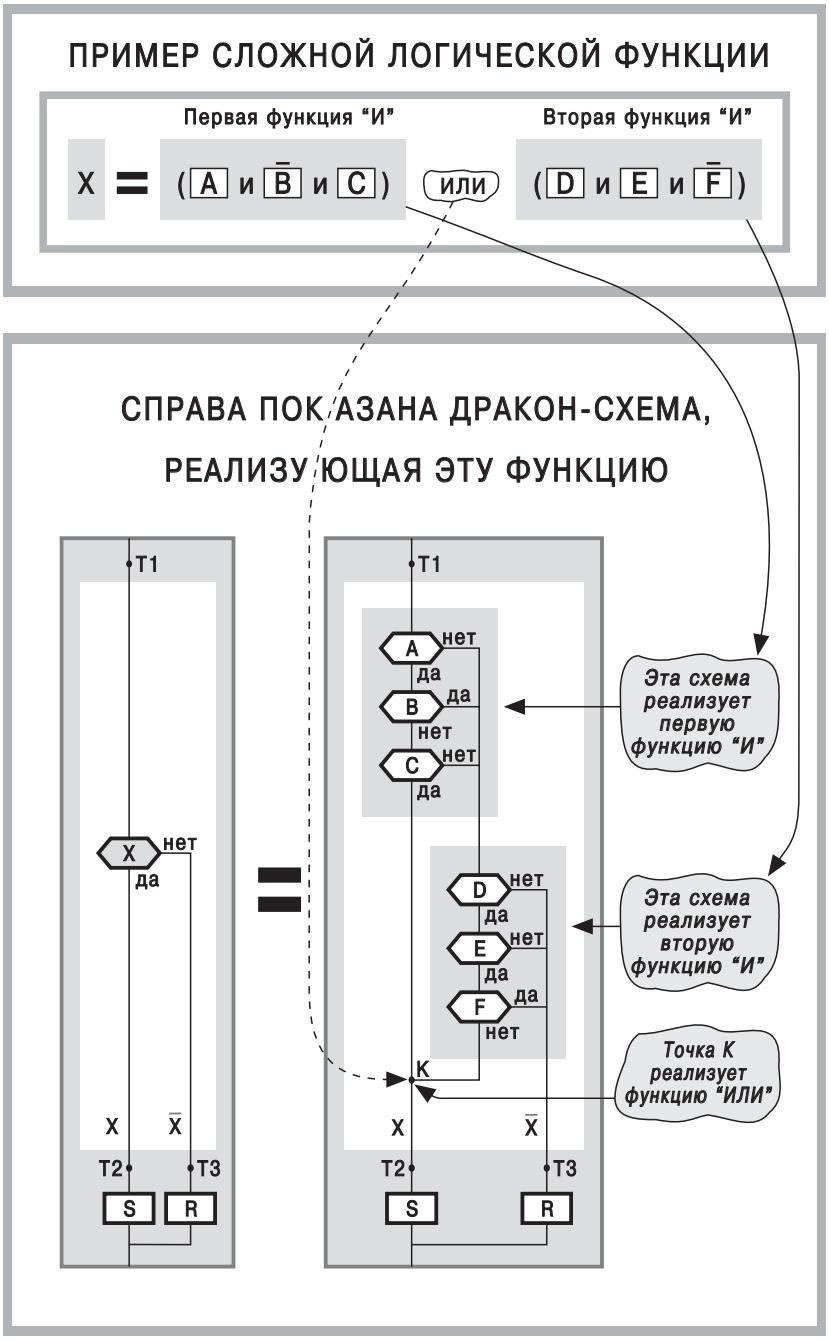
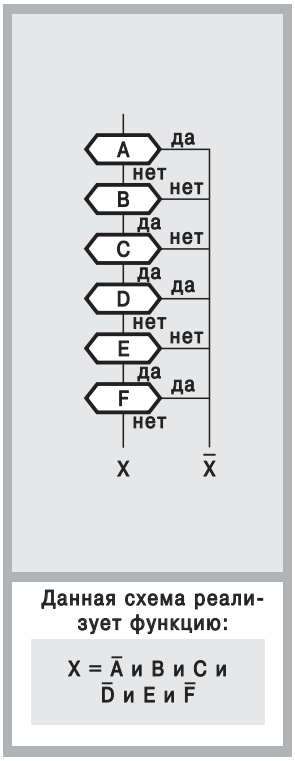


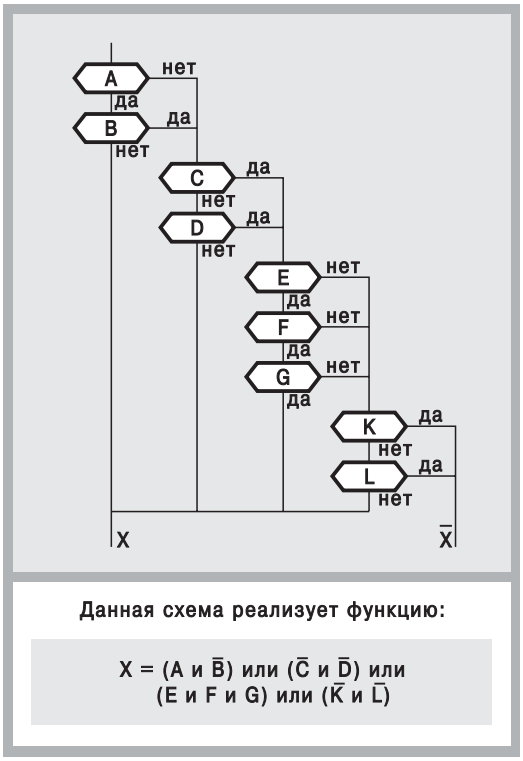
Рис. 67. **Как нарисовать дракон-схему для сложной логической функции?**



Данная схема реализует функцию:

$X = \bar{A}$ и B и C и $\bar{D}$ и E и $\bar{F}$

Рис. 68. **Пример сложной функции “И” с логическими отрицаниями**



Данная схема реализует функцию:

$X = (A$ и $\bar{B})$ или $(\bar{C}$ и $\bar{D})$ или $(E$ и F и $G)$ или $(\bar{K}$ и $\bar{L})$

Рис. 69. **Пример сложной логической функции**

КАКУЮ ЛОГИЧЕСКУЮ ФУНКЦИЮ ВЫЧИСЛЯЮТ ЭТИ СХЕМЫ?

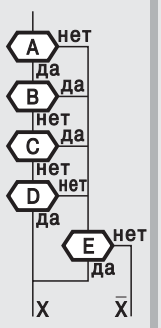


Рис. 70

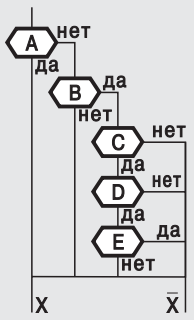


Рис. 71

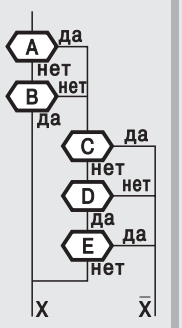


Рис. 72

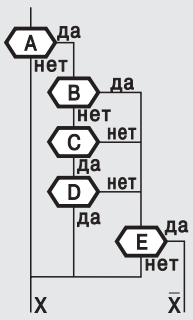


Рис. 73

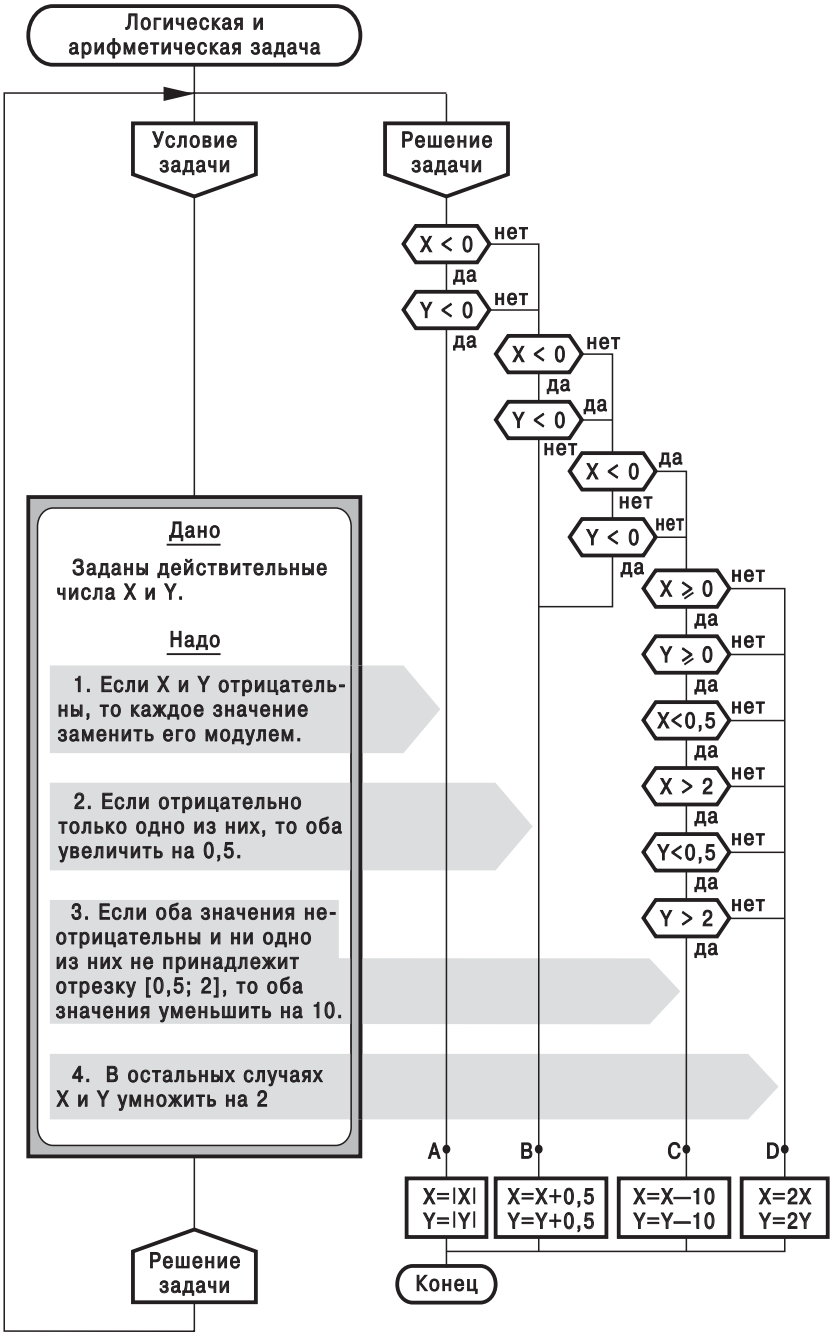


Рис. 74. **Сложная логическая задача с простыми вычислениями**

Визуальные формулы на рис. 61 показывают, что знак логического отрицания можно исключить из дракон-схемы, если поменять местами слова “да” и “нет” на выходах иконы “вопрос” (при этом иконы, находящиеся в плечах развилки, следует оставить на своих местах).

Упражнения на рис. 62—66 помогут читателю закрепить материал.

ВИЗУАЛИЗАЦИЯ СЛОЖНЫХ ЛОГИЧЕСКИХ ФУНКЦИЙ

Рассмотрим функцию

$$X = (A \text{ и } \overline{B} \text{ и } C) \text{ или } (D \text{ и } E \text{ и } \overline{F}) \qquad (1)$$

На рис. 67 показан визуальный способ записи этой функции. Из рисунка видно, что формула (1) разбивается на три части:

1) A и $\overline{B}$ и C; 2) D и E и $\overline{F}$; 3) Операция “или”.

Функция A и $\overline{B}$ и C изображается с помощью трех икон A, B, C, расположенных на одной вертикали. Аналогично рисуют функцию D и E и $\overline{F}$. Связка “или” реализуется с помощью линий, объединяющих нижние выходы икон C и F в точке K (рис. 67).

В формуле (1) некоторые члены записаны без логического отрицания (A, B, D, E), другие — с отрицанием ($\overline{B}$, $\overline{F}$). Члены без отрицания превращаются в иконы A, B, D, E, у которых нижний выход помечен словом “да”. Членам с отрицанием соответствуют иконы B и F, где нижний выход помечен словом “нет” (рис. 67). Другие примеры алгоритмов, вычисляющих сложные логические функции, представлены на рис. 68—74.

Изложенные соображения позволяют сформулировать две теоремы.

Теорема 1. Дракон-схему, содержащую логические связки И, ИЛИ, НЕ внутри икон “вопрос”, всегда можно преобразовать в эквивалентную дракон-схему, не содержащую указанных связок.

Теорема 2. Если некоторый фрагмент дракон-схемы имеет один вход, два выхода и содержит только иконы “вопрос”, причем первый выход вычисляет функцию X, то второй выход вычисляет ее логическое отрицание $\overline{X}$ (рис. 67—73).

Доказательство теорем предоставляем читателю.

ВЫВОДЫ

1. В алгоритмах со сложной логикой часто используются условные операторы с логическими выражениями. Опыт показывает, что такие операторы во многих случаях трудны для понимания, что нередко приводит к ошибкам.
2. В языке ДРАКОН используются визуальные логические выражения, позволяющие при желании полностью исключить логические связки И, ИЛИ, НЕ из условных операторов.
3. Визуализация логических формул во многих практически важных случаях заметно облегчает их понимание и уменьшает вероятность ошибок.

Глава 10
ЧТО ТАКОЕ ЭРГОНОМИЧНЫЙ ТЕКСТ?

Все в алгоритме понятно и ясно,
Если он сделан эргономично.
Эргономично — это прекрасно!
Эргономично — значит отлично!

МОЖНО ЛИ СДЕЛАТЬ ЛОГИЧЕСКИЕ ВЫРАЖЕНИЯ ЭРГОНОМИЧНЫМИ?

Одна из основных целей языка ДРАКОН — улучшение понимаемости алгоритмов, программ и технологий. До сих пор мы решали эту задачу методом визуализации, превращая часть текста в эргономичный графический образ. А как должна выглядеть другая часть текста — та, что не подлежит визуализации и записывается внутри икон? Как изменить эргономические характеристики текстовых надписей на дракон-схемах, чтобы в максимальной степени улучшить их понимаемость?

Поставленный вопрос слишком обширен и сложен. Поэтому сузим тему и ограничимся частной задачей: как следует записывать идентификаторы логических переменных и логические выражения, чтобы сделать их более понятными?

Для обсуждения темы лучше всего подходят формальные идентификаторы и формальные логические выражения. Это значит, что визуальный псевдоязык ДРАКОН-1 для указанной цели не годится, так как его текстовый синтаксис неформальный. По этой причине материал настоящей главы опирается на идеи визуального языка программирования ДРАКОН-2, у которого обе части синтаксиса (и визуальная, и текстовая) являются строго формальными. Таким образом, в данной главе мы впервые коснемся вопроса о программировании на языке ДРАКОН. Точнее говоря, речь пойдет об одном частном вопросе программирования, касающемся правил записи логических выражений.

ПРИМЕР ДЛЯ ИССЛЕДОВАНИЯ ЭРГОНОМИЧНОСТИ ЛОГИЧЕСКИХ ВЫРАЖЕНИЙ

Итак, мы собираемся найти эргономичный способ записи сложных логических выражений. Чтобы разобраться в сути вопроса, желательно иметь под рукой какой-нибудь пример, на котором мы будем “проигрывать” различные методы улучшения эргономичности.

Предположим, нужно создать алгоритм, управляющий автомобилем-роботом, проезжающим через перекресток со светофором в условиях реального уличного движения. Примем соглашение, что автомобиль-робот движется только по прямой, и выберем самый простой алгоритм управления (рис. 75).

Логический признак, разрешающий (или запрещающий) роботу ехать вперед, имеет идентификатор “Можно.ехать.через.перекресток”. Будем считать, что данный признак принимает значение “да” в трех случаях:

- горит зеленый сигнал светофора и нет помех движению;
- желтый сигнал загорелся, когда робот уже выехал на перекресток, и нет помех движению;
- светофор сломался (нет ни зеленого, ни желтого, ни красного сигнала) и нет помех движению.

В остальных случаях признак имеет значение “нет”, запрещающее роботу движение через перекресток.

Введем обозначения, показанные на рис. 76, которым соответствуют очевидные равенства:

Y = Можно.ехать.через.перекресток (1)
A = Зеленый.сигнал.светофора (2)
B = Желтый.сигнал.светофора (3)
C = Красный.сигнал.светофора (4)
D = Робот.выехал.на.перекресток (5)
E = Помехи.для.движения (6)

Если принять указанные условия и обозначения, логическая функция Y задается формулой

$$Y = (A \,\&\, \neg E) \vee (B \,\&\, D \,\&\, \neg E) \vee (\neg A \,\&\, \neg B \,\&\, \neg C \,\&\, \neg E) \qquad (7)$$

Пример, представленный на рис. 75 и 76, позволяет приступить к изучению проблемы. Ниже мы рассмотрим несколько вариантов записи логических выражений и сравним их между собой с эргономической точки зрения. При этом предполагается, что робот имеет пять датчиков, формирующих логические сигналы A, B, C, D, E, которые поступают в бортовой компьютер, управляющий движением робота.

ЛОГИЧЕСКОЕ ВЫРАЖЕНИЕ С АБСТРАКТНЫМИ ИДЕНТИФИКАТОРАМИ

Произведем эквивалентное преобразование алгоритма на рис. 75. Учитывая равенство (1) заменим идентификатор “Можно.ехать.через. перекресток” буквой Y, после чего вместо Y подставим логическое выражение из формулы (7). В результате получим алгоритм на рис. 77.

Некоторые математики скорее всего похвалят этот алгоритм. Они, возможно, скажут, что с математической точки зрения выражение в иконе “вопрос” является компактным, лаконичным, изящным и обозримым [1].

[1] Указанное выражение можно еще больше упростить — вынести член ¬E за скобки, однако для наших целей это несущественно.

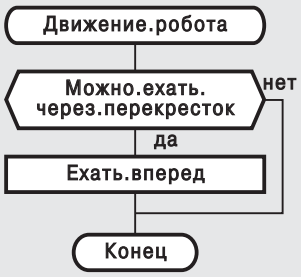


Рис. 75. **Алгоритм “Движение робота”**

Логическая функция Y описывается формулой

$$Y=(A\&\neg E)\ V\ (B\&D\&\neg E)\ V\ (\neg A\&\neg B\&\neg C\&\neg E)$$

где Y,A,B,C,D,E–логические переменные,
Y – означает “Можно.ехать.через.перекресток”
A – означает “Зеленый.сигнал.светофора”
B – означает “Желтый.сигнал.светофора”
C – означает “Красный.сигнал.светофора”
D – означает “Робот.выехал.на.перекресток”
E – означает “Помехи.для.движения”
& – логическая операция “И”
V – логическая операция “ИЛИ”
¬ – логическая операция “НЕ”

Рис. 76. **Логическая функция**

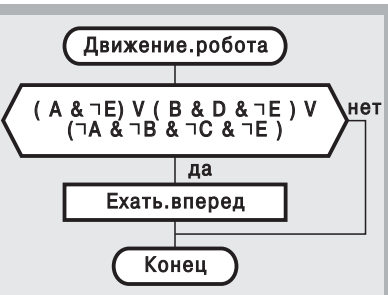


Рис. 77. **Эргономически неудачная дракон-схема, содержащая логическое выражение с абстрактными идентификаторами**

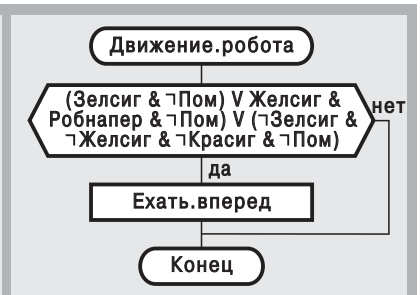


Рис. 78. **Эргономически неудачная дракон-схема, содержащая логическое выражение с короткими смысловыми идентификаторами**

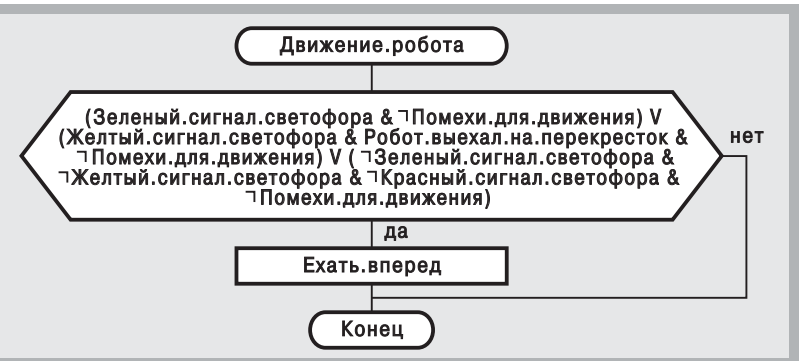


Рис. 79. **Эргономически неудачная дракон-схема, содержащая логическое выражение с длинными смысловыми идентификаторами**

К сожалению, подобная позиция не учитывает эргономических соображений и является устаревшей. Сразу оговоримся: речь, разумеется, идет не о том, чтобы заменить математику эргономикой, а всего лишь о том, чтобы, сохраняя математическую строгость в неприкосновенности, решительно отказаться от эргономической беспечности традиционных математических построений. Это означает, что однобокий математический подход должен уступить место системному подходу, в котором органически сочетаются математические и эргономические методы. Для обозначения нового подхода можно предложить термин “когнитивная формализация знаний”.

Сделаем еще одну оговорку. Формула (7) была бы вполне приемлемой, если бы речь шла об абстрактной задаче, цель которой — выявить математическую сущность проблемы. Однако в данном случае речь идет о прикладной задаче — создании программы управления автомобилем-роботом.

Недостаток формулы (7) и алгоритма на рис. 77 состоит в том, что идентификаторы *A*, *B*, *C*, *D*, *E* не смысловые, а абстрактные. Они оставляют наши знания о предметной области за пределами программного текста.

Чем это плохо? Вспомним, что сегодня критической проблемой являются не машинные, а человеческие ресурсы, причем экономия последних теснейшим образом связана с проблемой понимания, которая превращается в центральную проблему информатики. Производительность труда при создании информационных систем и систем управления напрямую зависит от успешного решения проблемы понимания, обеспечивающего быструю и безошибочную разработку алгоритмов и программ. Это общее положение тесно связано с обсуждаемым вопросом. В самом деле, люди, которые прекрасно знают прикладную задачу и предметную область, но не знают или забыли обозначения (1)—(6), например заказчики, постановщики задач, комплексники и т. д., воспринимают идентификаторы *A*, *B*, *C*, *D*, *E* и составленные из них формулы как бессмысленный набор символов. Следовательно, эти люди лишаются возможности принять участие в проверке правильности алгоритмов и программ и внести свой вклад в устранение ошибок.

Чтобы обнаружить ошибку в логическом выражении, необходимо хорошо понимать его смысл. Чтобы уяснить суть логического выражения на рис. 77, человек вынужден помнить не только смысловые понятия, но и абстрактные идентификаторы, твердо знать соответствие между ними. Это создает двойную нагрузку на память человека (алгоритмиста, программиста и т. д.), порождает дополнительные и ничем не обоснованные трудности при поиске и выявлении ошибок.

Для большинства специалистов, знающих прикладную задачу, логическое выражение на рис. 77 не дает никакой подсказки о семантике логических переменных и фактически представляет собой загадочный ребус. Оно служит типичным примером эргономической неряшливости традиционных методов математического описания прикладных задач. Отсюда проистекает вывод: в прикладных задачах использование абстрактных идентификаторов в логических выражениях эргономически недопустимо.

ЛОГИЧЕСКОЕ ВЫРАЖЕНИЕ С КОРОТКИМИ СМЫСЛОВЫМИ ИДЕНТИФИКАТОРАМИ

Абстрактные идентификаторы использовались на первом этапе развития программирования. Сегодня в прикладных программах они встречаются гораздо реже, уступив место так называемым мнемоническим именам, т. е. коротким смысловым идентификаторам, которые в большинстве случаев имеют длину до восьми символов. Преобладание восьмисимвольных идентификаторов характерно для второго этапа развития языков программирования. Вот типичная рекомендация этого периода: “не оправдано применение имен, подобных *X* или *I*, тогда как имена *MAX* или *NEXT* передают смысл гораздо точнее”.

Последуем совету и, продолжая наш пример с автомобилем-роботом, заменим абстрактные идентификаторы на мнемонические имена согласно табл. 2.

Таблица 2

Абстрактный идентификатор	Мнемоническое имя
Y	Можнех
A	Зелсиг
B	Желсиг
C	Красиг
D	Робнапер
E	Пом

На рис. 78 показан алгоритм, полученный в результате такой замены. Можно ли назвать логическое выражение на рис. 78 эргономичным? Очевидно, что мнемонические имена лучше абстрактных. Они были придуманы с благородной целью — облегчить запоминание понятий, чтобы формальное имя создавало намек на содержательную сторону дела. Увы! Говорить намеками — вовсе не значит говорить понятно. Причина неудачи в том, что длина идентификатора восемь символов слишком мала и явно недостаточна для хорошего, ясного и доходчивого описания сложных понятий. Поэтому при создании восьмисимвольных идентификаторов приходится экономить каждый символ и часто использовать невразумительные слова-обрубки, такие, как Зелсиг (зеленый сигнал светофора), Красиг, Желсиг и т. д.

В самом деле, глядя на идентификатор “Робнапер” (рис. 78) мало кто догадается, что речь идет о признаке “Робот.выехал.на.перекресток”.

Сравнивая логические выражения на рис. 77 и 78, можно сказать, что в последнем случае понимаемость алгоритма, если и увеличилась, то ненамного. Таким образом, восьмисимвольные смысловые идентификаторы не могут обеспечить требуемое улучшение эргономических характеристик логических выражений.

ЛОГИЧЕСКОЕ ВЫРАЖЕНИЕ С ДЛИННЫМИ СМЫСЛОВЫМИ ИДЕНТИФИКАТОРАМИ

Для третьего этапа развития языков программирования, который начался сравнительно недавно, характерен переход к длинным смысловым идентификаторам, содержащим до 32 символов. Здесь необходимо уточнение. Некоторые трансляторы делят идентификаторы на две части, обрабатывая только первую часть и игнорируя вторую. В результате разные идентификаторы, имеющие различие в последних символах, рассматриваются транслятором как тождественные. Такие случаи мы исключаем из рассмотрения. При дальнейшем изложении подразумевается, что все инструментальные программы, включая транслятор, обеспечивают необходимую программную обработку всех 32 символов идентификатора.

Увеличение длины идентификатора до 32 символов позволяет получить два важных эргономических преимущества. Во-первых, во многих (хотя и не во всех) случаях появляется возможность отказаться от сокращений и использовать полные слова. Во-вторых, для разграничения слов, входящих в состав идентификатора, можно ввести эргономичные разделители, например точку или нижнюю черту.

Пример логического выражения, в котором используются идентификаторы с точками-разделителями и полными (несокращенными) словами, представлен на рис. 79. Легко видеть, что оно обладает более высокой понимаемостью, чем предыдущие примеры. Тем не менее здесь есть одно “но”.

ВАЖНЫЙ МОМЕНТ, О КОТОРОМ ЧАСТО ЗАБЫВАЮТ

На рис. 75 записан вопрос “Можно.ехать.через.перекресток”, который объясняет принцип разветвления алгоритма: при ответе “да” выполняется команда “Ехать.вперед”, при ответе “нет” действия отсутствуют. Данный вопрос играет ключевую роль: если его удалить, алгоритм становится непонятным. В связи с этим целесообразно ввести понятие *главный вопрос условного оператора*.

А теперь взглянем на рис. 77—79. Нетрудно заметить, что в иконе “вопрос” записана масса любопытных подробностей, однако интересующий нас вопрос отсутствует. Он бесследно исчез.

Таким образом, логические выражения на рис. 77—79 имеют общий недостаток, причем весьма существенный. В них нет главного вопроса, нет ключа, объясняющего сущность алгоритма. Налицо парадокс: логические выражения не дают явной информации о том, *на какой именно вопрос* мы отвечаем “да” или “нет”. Более того, они не позволяют читателю легко и быстро восстановить формулировку главного вопроса и не стимулируют у него стремления к получению подобной информации. Не будет преувеличением сказать, что перечисленные логические выражения затуманивают суть дела, поскольку отсутствие главного вопроса ничем нельзя компенсировать. В итоге алгоритмы на рис. 77—79 оказываются непонятными и эргономически неприемлемыми. Отсюда вытекает, что использование эргономически правильных длинных смысло-

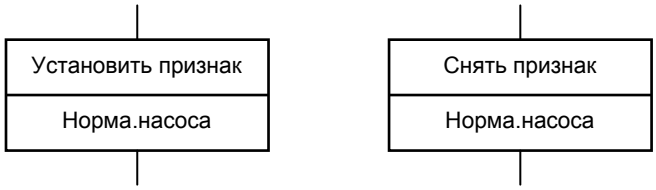
вых идентификаторов является необходимым, но отнюдь не достаточным условием для построения эргономичного логического текста.

КАК ПРИСВОИТЬ ЗНАЧЕНИЕ ЛОГИЧЕСКОЙ ПЕРЕМЕННОЙ?

Мы уже говорили, что в традиционных языках для значений логических переменных используют слова *TRUE* и *FALSE*, ИСТИНА и ЛОЖЬ, 1 и 0. Однако логико-эргономические исследования показывают, что указанные обозначения являются избыточными и могут быть безболезненно и с пользой для дела исключены из программных текстов. Стремление "уничтожить" лишние обозначения объясняется эргономическими причинами, так как все ненужные записи являются визуальными помехами, которые засоряют текст программы и путают читателя.

Язык ДРАКОН позволяет решить задачу двумя способами. В первом случае применяется икона "действие", внутри которой записывается оператор присваивания (рис. 80). Визуальный оператор на рис. 80 означает, что идентификатору "Можно.ехать.через.перекресток" присваивается некоторое значение. Какое именно? Для этого нужно вычислить *рамочное логическое выражение*, записанное в трех рамках, соединенных знаками ИЛИ. Результатом вычисления будет "1" или "0". Таким образом, цель достигнута, хотя обозначения "1" и "0" в программном тексте отсутствуют.

Во втором случае используется икона "полка" (рис. 1, икона И10), на верхнем этаже которой пишут зарезервированное предложение "Установить признак" или "Снять признак". На нижнем этаже указывают идентификатор признака. Операторы языка ДРАКОН



означают, что логической переменной "Норма.насоса" присваивается значение "1" и "0" соответственно. Еще один пример использования иконы "полка" показан на рис. 81.

Легко видеть, что на рис. 81 используется та же хитрость, что и на рис. 80, а именно: логической переменной присваивается значение "1" или "0", хотя обозначения "1" и "0" в тексте программы нигде не встречаются!

ПРАВИЛА ЗАПИСИ РАМОЧНЫХ ЛОГИЧЕСКИХ ВЫРАЖЕНИЙ

Увеличение длины идентификаторов приводит к тому, что традиционная горизонтальная запись логических выражений становится невозможной. В связи с этим применяется вертикальная запись, пример которой показан на рис. 80. Вертикальный логический текст на языке ДРАКОН пишут в соответствии со следующими правилами.

- В иконе “действие” размещают один оператор присваивания.
- В верхней строке пишут идентификатор логической переменной и знак присваивания.
- Ниже пишут логическое выражение, причем каждая конъюнкция заключается в прямоугольную рамку.
- Для операций И, ИЛИ, НЕ используют обозначения &, ИЛИ, ¬ соответственно.
- Используют идентификаторы длиной до 32 символов.
- Первые символы всех идентификаторов располагают на одной вертикали.
- Знак отрицания ¬ пишут слева от идентификатора.
- Все знаки отрицания (если они есть) помещают на одной вертикали.
- Знаки конъюнкции & записывают справа от идентификатора.
- Все знаки конъюнкции пишут на одной вертикали.
- Вертикальные линии рамок располагают на одной вертикали.
- Знаки = и ИЛИ помещают на одной вертикали.

КАК ПОСТРОИТЬ ЭРГОНОМИЧНЫЙ ЛОГИЧЕСКИЙ ТЕКСТ?

Ранее мы пришли к выводу, что алгоритм на рис. 79 является эргономически неудачным. Каким образом можно его исправить? Вопрос отнюдь не простой. По-видимому, в разных ситуациях он может приводить к разным ответам. В связи с этим изложенные ниже соображения и советы имеют не обязательный, а всего лишь рекомендательный характер. Их нужно рассматривать как один из возможных способов решения проблемы.

- В иконе “вопрос” не следует записывать логическое выражение, в особенности сложное. Вместо него рекомендуется поместить один-единственный идентификатор, содержащий ясную и четкую словесную формулировку главного вопроса.
- В общем случае указанный идентификатор может оказаться неопределенным. Чтобы исключить эту неприятность, необходимо заблаговременно присвоить ему нужное значение. Для этого существуют два метода: рамочный и визуальный.
- В *рамочном методе* используется рамочное логическое выражение, записанное в иконе “действие”. Производится вычисление рамочного выражения, результат присваивается идентификатору главного вопроса (рис. 80).
- В *визуальном методе* применяется визуальное логическое выражение и два оператора “Установить признак” и “Снять признак”, записанные в иконах “полка”.

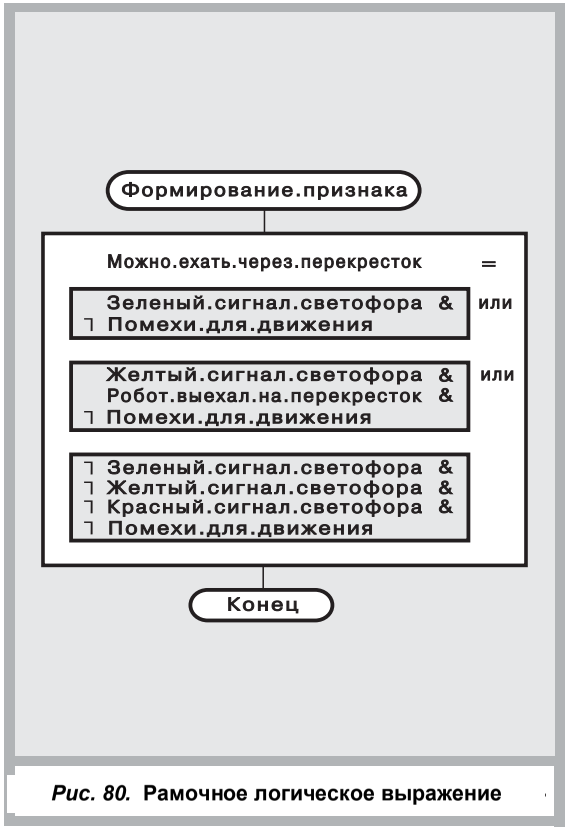


Рис. 80. **Рамочное логическое выражение**

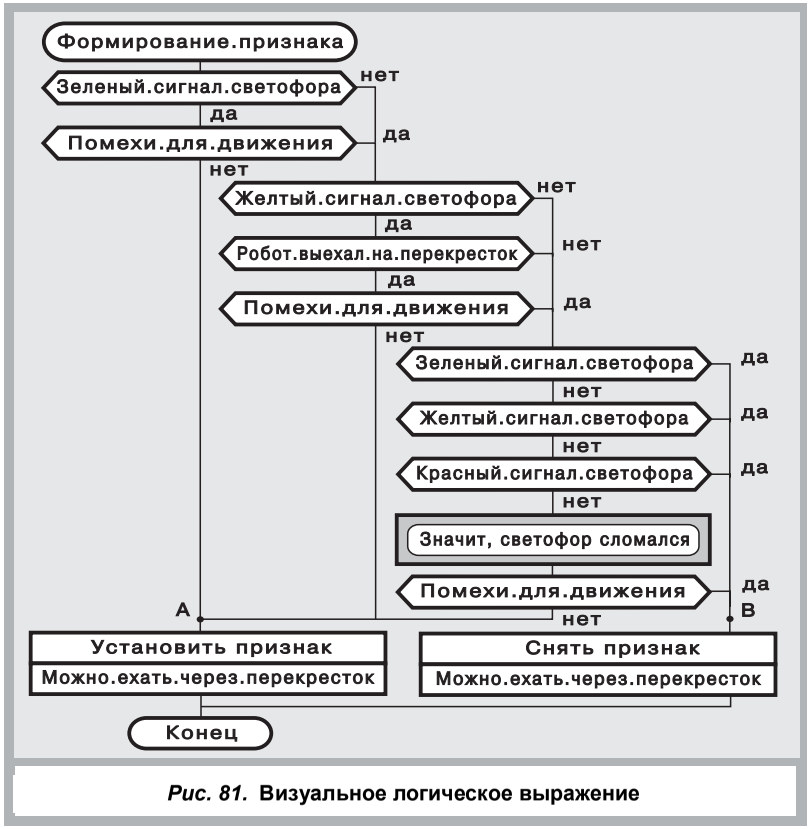


Рис. 81. **Визуальное логическое выражение**

В отличие от рамочного при визуальном методе вычисление логического выражения как таковое отсутствует. Визуальное выражение разветвляет процесс и приводит его в одну из двух точек (*A* или *B* на рис. 81). В первой точке выполняется оператор “Установить признак”, во второй — “Снять признак”. Алгоритмы на рис. 80 и 81 эквивалентны.

Инструментальные программы языка ДРАКОН должны обеспечить автоматический перевод рамочного алгоритма (рис. 80) в визуальный (рис. 81) и наоборот. Предоставление такой услуги пользователю создает для него дополнительный интеллектуально-эргономический комфорт, позволяет сравнить две формы представления логических знаний и выбрать ту, которая ему больше по душе. Поскольку вкусы автора алгоритма и его читателей могут отличаться, каждый из них может получить листинг (чертеж) программы в том виде, который лично ему больше нравится, реализуя тем самым свое право на индивидуальное предпочтение той или иной формы представления знаний.

Оптимальная длина формального смыслового идентификатора составляет примерно 32 символа. Имеется в виду, что инструментальные программы осуществляют обработку 32-байтового поля идентификатора. Желательно, чтобы конкретные идентификаторы в зависимости от сложности понятия имели длину не менее 25 и не более 32 символов. Чтобы исключить ошибки при ручном вводе столь длинных идентификаторов в компьютер, целесообразно ввести запрет повторного ввода. Это значит, что идентификатор вводится в систему только один раз и запоминается в базе данных. При необходимости повторного ввода осуществляется копирование из базы данных. Такой способ требует наличия специальных инструментальных средств, но гарантирует идентичность всех копий одного и того же идентификатора.

Вывод об оптимальности 32-символьных идентификаторов согласуется с анализом истории развития языков программирования, который обнаруживает отчетливую тенденцию: от абстрактных кодов и имен к 8-символьным мнемоническим именам, а затем — к 32-символьным смысловым идентификаторам. Вместе с тем многие программисты, следуя устоявшимся привычкам, “застряли” на этапе 8-символьных имен, так что опыт использования новых возможностей, связанных с разрешением использовать 32 символа, пока еще относительно невелик. Между тем, эргономические перспективы, открывающиеся с увеличением длины до 32 символов, обещают существенно изменить наши прежние представления и привычки, так как благодаря этому замечательному нововведению язык формальных идентификаторов по своей доходчивости значительно приближается к естественному человеческому языку, что отчетливо видно на рис. 80 и 81. В самом деле, множество 32-символьных идентификаторов образует весьма выразительный, хотя и своеобразный язык, законы и правила оптимизации которого еще предстоит открыть, обсудить и подвергнуть экспериментальной проверке.

Специальные обозначения для значений логических переменных как принадлежность языка программирования — это анахронизм, который следует исключить из всех языков как совершенно не нужное и даже вредное “архитектурное излишество”. Чтобы оправдать этот вывод, сравним два выражения в условном операторе:

Если (Норма 1 = 1) & (Норма 2 = 1) & (Авария = 0), то... (10)

Если Норма 1 & Норма 2 & ¬Авария, то.... (11)

Формула (10) читается так:

Если признак “Норма 1” равен единице и признак “Норма 2” равен единице и признак “Авария” равен нулю, то…	(10*a*)

Формула (11) читается так:

Если есть признак “Норма 1” и есть признак “Норма 2” и нет признака “Авария”, то…	(11*a*)

Фраза (11*a*) по своему лексическому строю соответствует обычным речевым оборотам, которыми пользуются специалисты предметной области, не являющиеся программистами. Она точно отражает суть дела и понятна всем работникам, в то время как фраза (10*a*) содержит искусственные и нарочитые вкрапления “равен единице” и “равен нулю”, появление которых неоправданно удлиняет текст и разрушительно действует на процесс восприятия, делая предложение непонятным для всех, кроме программистов.

ВЫВОДЫ

1. Точкой роста современной науки являются междисциплинарные исследования, в частности на стыке логики и эргономики.
2. Сегодня, когда критической проблемой являются не машинные, а человеческие ресурсы, традиционные методы записи логических выражений следует признать во многом устаревшими, ибо они не учитывают эргономических соображений.
3. Предложен двухэтапный метод эргономизации логических выражений. На первом этапе производится разделение логических записей на две части, из которых одна подлежит визуализации, а другая сохраняется в текстовом виде. Второй этап — эргономизация обеих частей: визуальной и текстовой.
4. Эргономизация текстовой части включает, в частности, следующие приемы:
   - оптимизацию длины и правил записи идентификаторов;
   - выбор альтернативы: логическое выражение или идентификатор главного вопроса;
   - исключение обозначений для значений логических переменных;
   - сравнительный анализ визуальной и рамочной форм записи и выбор одной из них.

Глава 11

ВИЗУАЛЬНЫЕ ОПЕРАТОРЫ РЕАЛЬНОГО ВРЕМЕНИ

> Удачный рисунок иногда не только позволяет сделать наглядной и понятной суть сложного вопроса, но нередко способен подсказать принципиально новое соображение, идею, гипотезу, которые без такого рисунка просто, что называется, не приходят в голову.
>
> *Александр Зенкин*

СПИСОК ОПЕРАТОРОВ РЕАЛЬНОГО ВРЕМЕНИ

В языке ДРАКОН имеется пять икон реального времени (рис. 1, иконы И16 — И20):

- пауза;
- период;
- пуск таймера;
- синхронизатор (по таймеру);
- параллельный процесс.

Три из них (пауза, пуск таймера и параллельный процесс) — простые операторы. Две другие (период и синхронизатор) служат "кирпичиками" для построения составных операторов и вне последних не используются.

Икона "период" является принадлежностью цикла ЖДАТЬ (рис. 2, макроикона 7). Икона "синхронизатор" служит для образования тринадцати составных операторов (рис. 2, макроиконы 8—20).

Назначение операторов поясним, как всегда, на примерах.

ОПЕРАТОРЫ ВВОДА-ВЫВОДА

В языке ДРАКОН предусмотрены два визуальных оператора ввода-вывода: "вывод" (рис. 1, икона И14) и "ввод" (рис. 1, икона И15). Они не относятся к операторам реального времени и рассматриваются здесь только потому, что встречаются в ближайшем примере.

Из рис. 1 видно, что иконы ввода-вывода имеют мнемоническую форму: икона И14 содержит полую стрелку, направленную наружу, что символизирует "вывод", а икона И15 — стрелку, направленную внутрь (ввод). Оба оператора "двухэтажные", причем на верхнем

Рис. 82. **Пример использования оператора “пауза”**

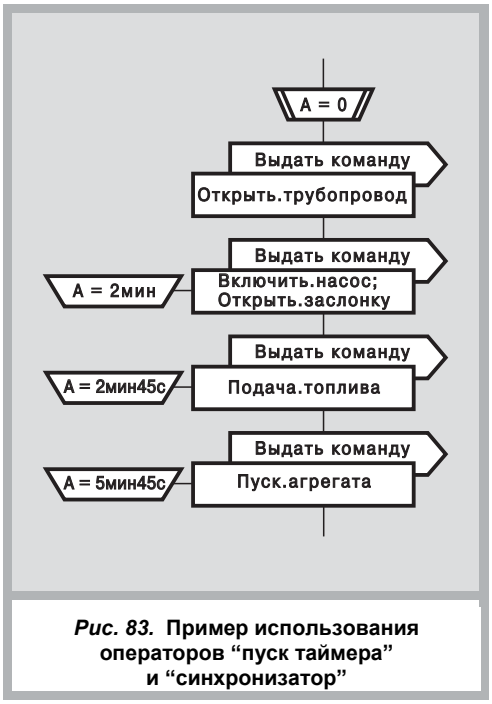


Рис. 83. **Пример использования операторов “пуск таймера” и “синхронизатор”**

этаже пишется ключевое слово или ключевая фраза, а на нижнем (в прямоугольнике) — содержательная информация, подлежащая вводу и выводу (рис. 82, 83).

ОПЕРАТОР “ПАУЗА”

Предположим, управляющий компьютер должен выдать серию электрических команд, которые по линиям связи передаются в исполнительные органы и вызывают срабатывание электромеханических реле. В результате происходит открытие трубопровода, включение насоса и другие операции, необходимые для функционирования управляемого объекта. Такая ситуация может встретиться во многих системах управления реального времени: при заправке топливом баллистических ракет, на атомных электростанциях, нефтеперерабатывающих заводах и т. д.

Рассмотрим конкретный пример. Предположим, управляющий компьютер должен:

- выдать команду ОТКРЫТЬ.ТРУБОПРОВОД;
- подождать две минуты;
- выдать две команды: ВКЛЮЧИТЬ.НАСОС и ОТКРЫТЬ.ЗАСЛОНКУ;
- подождать 45 секунд;
- выдать команду ПОДАЧА.ТОПЛИВА;
- подождать три минуты;
- выдать команду ПУСК.АГРЕГАТА.

Соответствующая программа на языке ДРАКОН-2 представлена на рис. 82. Задержка выдачи команд реализуется с помощью иконы

“пауза”, внутри которой указывается время необходимой задержки, например, 2 мин (2 минуты), 45 с (45 секунд) и т. д. Если говорить более точно, верхний оператор “пауза” на рис. 82 работает так: после выдачи команды ОТКРЫТЬ.ТРУБОПРОВОД в управляющем компьютере запускается виртуальный счетчик времени на 2 минуты, по окончании которых компьютер выдает в линию связи команды ВКЛЮЧИТЬ.НАСОС и ОТКРЫТЬ.ЗАСЛОНКУ.

ОПЕРАТОРЫ “ПУСК ТАЙМЕРА” И “СИНХРОНИЗАТОР”

Вернемся еще раз к задаче, описанной в предыдущем параграфе, и слегка изменим ее. Будем считать, что разработчик управляемого объекта хочет указать время выдачи команд не по принципу “задержка после предыдущей команды”, а по принципу секундомера, когда все времена отсчитываются от единого начального момента (совпадающего с пуском секундомера).

Исходя из этого, сформулируем задачу управляющего компьютера. Он должен:

- включить “секундомер”, т. е. обнулить и запустить виртуальный таймер;
- выдать команду ОТКРЫТЬ.ТРУБОПРОВОД;
- когда таймер отсчитает две минуты, выдать пару команд ВКЛЮЧИТЬ.НАСОС и ОТКРЫТЬ.ЗАСЛОНКУ;
- когда таймер отсчитает 2 минуты 45 секунд, выдать команду ПОДАЧА.ТОПЛИВА;
- когда таймер отсчитает 5 минут 45 секунд, выдать команду ПУСК. АГРЕГАТА.

Программа, реализующая описанный алгоритм, изображена на рис. 83. В ней используются операторы “пуск таймера” и “синхронизатор”, совместная работа которых обеспечивает нужный эффект.

Оператор “пуск таймера” порождает, обнуляет и запускает виртуальный таймер и присваивает ему имя *A*. Оператор “синхронизатор” задерживает выполнение размещенного справа от него визуального оператора до наступления момента, описанного в иконе “синхронизатор”. Например, синхронизатор *A* = 2мин 45с на рис. 83 задерживает выдачу команды ПОДАЧА.ТОПЛИВА до момента, когда таймер *A* отсчитает 2 минуты 45 секунд.

Сравнивая программы на рис. 82 и 83, можно заметить, что они почти эквивалентны. Почему почти? Если взять идеальный случай и пред-

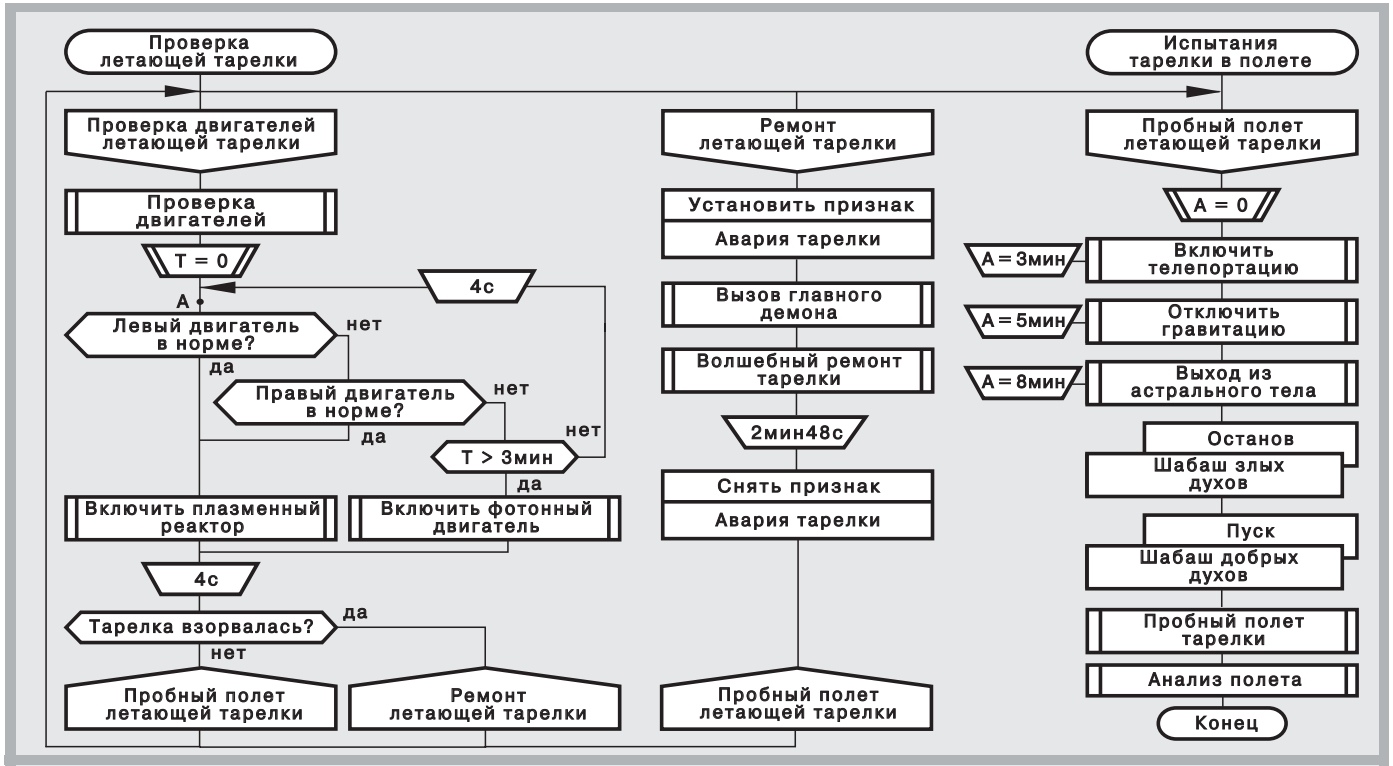


Рис. 84. **Алгоритм реального времени**

положить, что время, необходимое для выдачи одной команды равно нулю, обе программы будут выдавать команды синхронно. Однако в действительности указанное время больше нуля, поэтому в реальной жизни программы работают по-разному.

Практика разработки систем управления показывает, что в некоторых ситуациях предпочтительным является *принцип паузы*, а в других — *принцип таймера*, поэтому оба инструмента оказываются в равной степени необходимыми и полезными.

На рис. 84 представлен более сложный алгоритм, в котором используются операторы “пауза”, “пуск таймера” и “синхронизатор”.

В средней ветке изображена икона “пауза” с записью 2мин48с. Это означает, что после завершения процедуры ВОЛШЕБНЫЙ РЕМОНТ ТАРЕЛКИ отсчитывается пауза длительностью 2 минуты 48 секунд и только после этого производится снятие признака АВАРИЯ ТАРЕЛКИ. Еще одна четырехсекундная пауза предусмотрена в левой ветке.

В правой ветке есть икона “пуск таймера” с записью $A = 0$. Данный оператор порождает, обнуляет и запускает виртуальный таймер A. В той же ветке установлены три иконы “синхронизатор по таймеру” с записями $A = 3$мин, $A = 5$мин и $A = 8$мин. При этом вызов процедуры ВКЛЮЧИТЬ ТЕЛЕПОРТАЦИЮ произойдет не сразу, а только после того, как таймер A отсчитает 3 минуты. Соответственно включение в работу процедур ОТКЛЮЧИТЬ ГРАВИТАЦИЮ и ВЫХОД ИЗ АСТРАЛЬНОГО ТЕЛА будет задержано до тех пор, пока таймер A не примет значения 5 и 8 минут соответственно.

Из рис. 84 видно, что оператор “пуск таймера” можно применять двумя способами. Во-первых, совместно с иконой “синхронизатор” (этот случай мы обсудили), во-вторых, совместно с иконой “вопрос”. Последний случай рассмотрен в следующем параграфе.

ЦИКЛ ЖДАТЬ

Предположим, требуется в течение трех минут ждать появления хотя бы одного из двух признаков ЛЕВЫЙ ДВИГАТЕЛЬ В НОРМЕ и ПРАВЫЙ ДВИГАТЕЛЬ В НОРМЕ. При наступлении этого события (появлении признака) необходимо включить плазменный реактор. Если же названные признаки отсутствуют, по истечении трех минут следует включить фотонный двигатель.

Для решения задачи на рис. 84 используются два оператора: пуск таймера T, отсчитывающего три минуты, и цикл ЖДАТЬ. В состав последнего входит икона “период” и три иконы “вопрос”, в которых размещены надписи ЛЕВЫЙ ДВИГАТЕЛЬ В НОРМЕ?, ПРАВЫЙ ДВИГАТЕЛЬ В НОРМЕ? и $T >$ 3мин (последний оператор проверяет: значение таймера T больше трех минут?). Если оба признака отсутствуют, а значение таймера не превышает трех минут, опрос условий периодически повторяется, причем период опроса указывается в иконе “период”. В данном примере он равен 4 секундам.

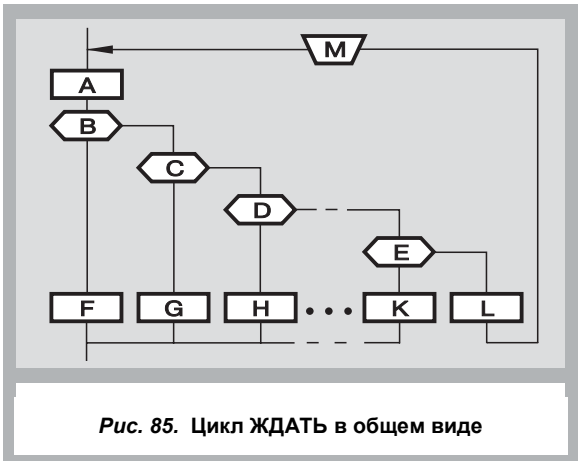


Рис. 85. **Цикл ЖДАТЬ в общем виде**

Как явствует из рисунка, работа цикла ЖДАТЬ закончится в момент обнаружения одного из ожидаемых признаков, а если они так и не появятся — через три минуты.

В общем виде цикл ЖДАТЬ показан на рис. 85. Он позволяет организовать режим ожидания признаков *B*, *C*, *D*, ..., *E*. Если первым появится признак *B*, выполняется действие *F*. Если *B* отсутствует и первым придет признак *C*, реализуется действие *G*. И так далее. Операторы *A* и *L* обычно не используются.

Задача ожидания нескольких признаков (когда система должна поразному реагировать на каждый признак) является одной из наиболее типичных при разработке систем управления реального времени. Цикл ЖДАТЬ предлагает чрезвычайно простое, удобное, наглядное и эффективное средство для ее решения, удовлетворяя тем самым важную потребность практики.

ОПЕРАТОР “ПЕРИОД”

Сравнивая макроиконы 4 и 7 на рис. 2 (обычный цикл и цикл ЖДАТЬ), мы видим, что они очень похожи. Поэтому во избежание путаницы нужно иметь какой-то различительный признак. Эту функцию выполняет икона “период”. Если она есть в петле цикла — перед нами цикл ЖДАТЬ. Если нет — обычный цикл.

Человек, который стоит на остановке и ждет появления трамвая или троллейбуса, воспринимает ожидание как нечто непрерывное. Однако программа реального времени организует ожидание как дискретный процесс и запускает цикл ЖДАТЬ периодически. Отсюда вытекает, что период — важная характеристика цикла ЖДАТЬ.

А теперь зададим самый интересный вопрос: как работает оператор “период”? Фокус в том, что на этот вопрос придется дать два совсем разных ответа.

С точки зрения человека, читающего программу на рис. 84, все обстоит очень просто: цикл ЖДАТЬ “крутится” по своей петле с периодичностью 4 секунды, пока не выполнится одно из трех условий, после чего произойдет выход из цикла. Таким образом, оператор “период” задает период повторения цикла ЖДАТЬ.

С точки зрения функционирования программы реального времени, дело обстоит иначе. Суть в том, что длительность периода отсчитывает не прикладная программа на рис. 84, а дракон-диспетчер, входящий в состав операционной системы реального времени. Оператор “период” означает выход из прикладной программы: управление переходит к дракон-диспетчеру (с одновременной передачей параметра 4с). Через каждые четыре секунды дракон-диспетчер передает управление в начало цикла ЖДАТЬ (точка *A* на рис. 84), и если все три условия дают ответ “нет”, оператор “период” всякий раз возвращает управление в дракон-диспетчер. Таким образом, функционирование цикла ЖДАТЬ обеспечивается совместными усилиями прикладной программы и дракон-диспетчера.

Нередко имеет место ситуация, когда разработчик программы реального времени использует цикл ЖДАТЬ, но считает, что для его программы конкретное значение периода не играет роли. В этом случае икону “период” следует оставить пустой; система по умолчанию присвоит периоду максимальное значение из того ассортимента, которым располагает дракон-диспетчер.

ОПЕРАТОР “ПАРАЛЛЕЛЬНЫЙ ПРОЦЕСС”

Пусть заданы два алгоритма *A* и *B*, причем *A* — основной алгоритм, а *B* — вспомогательный. Алгоритмы *A* и *B* могут работать последовательно (рис. 86) или параллельно (рис. 87). Чтобы организовать последовательную работу, необходимо в дракон-схеме основного алгоритма *A* нарисовать икону-вставку с надписью *B*. Например, на рис. 84 в основном алгоритме ПРОВЕРКА ЛЕТАЮЩЕЙ ТАРЕЛКИ имеется икона-вставка ПРОВЕРКА ДВИГАТЕЛЕЙ. Эти алгоритмы действуют последовательно. Основной алгоритм передает управление алгоритму ПРОВЕРКА ДВИГАТЕЛЕЙ и прекращает работу. Возобновление работы алгоритма ПРОВЕРКА ЛЕТАЮЩЕЙ ТАРЕЛКИ произойдет только тогда, когда алгоритм-вставка ПРОВЕРКА ДВИГАТЕЛЕЙ закончится. В общем виде ситуация показана на рис. 86.

Отличие параллельного режима состоит в том, что после начала вспомогательного алгоритма *B* основной алгоритм *A* не прекращает работу и действует одновременно с алгоритмом *B* (рис. 87).

Чтобы организовать параллельную работу, нужно в дракон-схеме основного алгоритма *A* нарисовать икону “параллельный процесс” (рис. 1, икона И20). Икона “двухэтажная”: на верхнем этаже пишут ключевое слово, обозначающее команду, изменяющую состояние параллельного процесса, например, “Пуск”, “Останов” и т. д. На нижнем этаже помещают идентификатор (название) параллельного процесса.

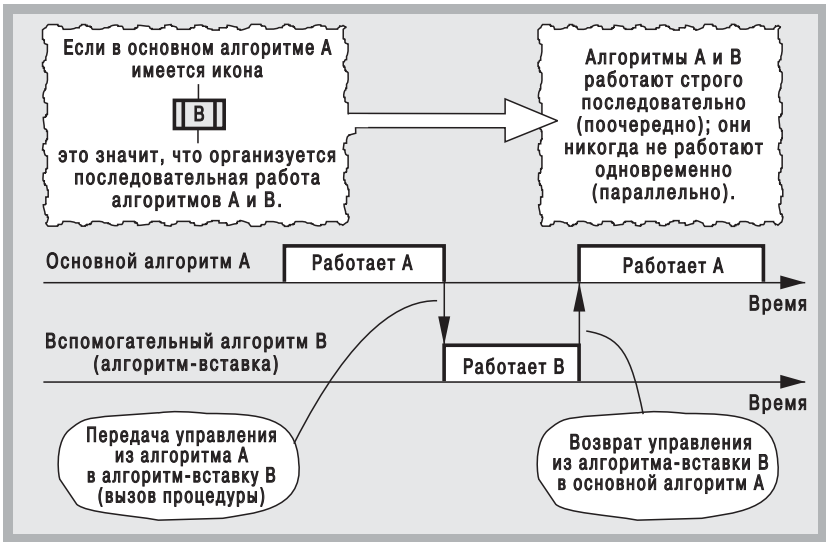


Рис. 86. **Последовательная работа алгоритмов**

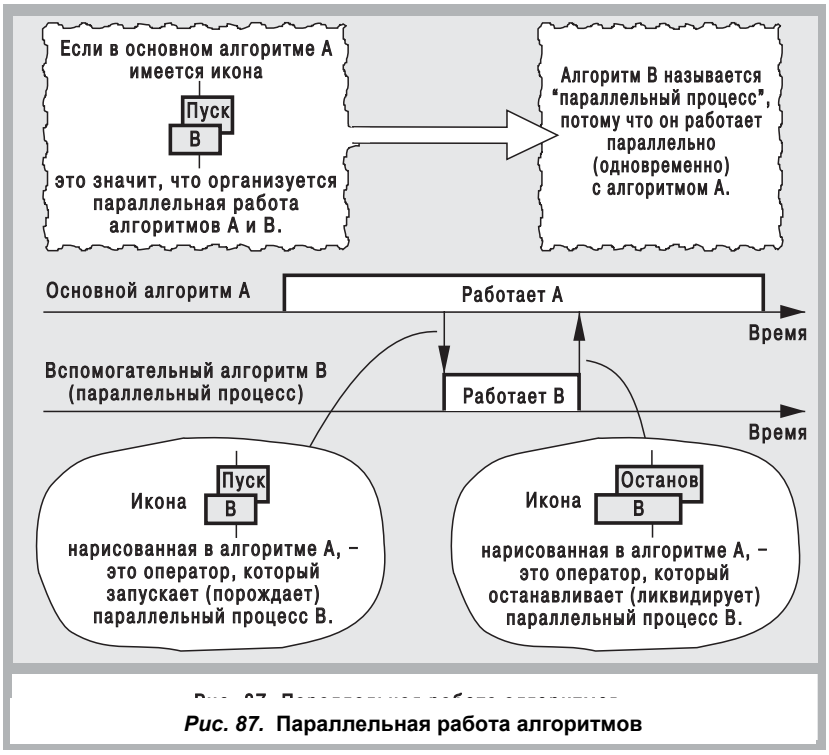


Рис. 87. **Параллельная работа алгоритмов**

Обратимся к примеру на рис. 84. В правой ветке находятся два оператора управления параллельными процессами. После окончания процедуры ВЫХОД ИЗ АСТРАЛЬНОГО ТЕЛА производится останов параллельного процесса ШАБАШ ЗЛЫХ ДУХОВ и пуск процесса ШАБАШ ДОБРЫХ ДУХОВ.

При этом предполагается, что до начала алгоритма ПРОВЕРКА ЛЕТАЮЩЕЙ ТАРЕЛКИ некий третий алгоритм выдал команду “Пуск” и запустил параллельный процесс ШАБАШ ЗЛЫХ ДУХОВ. Последний работает одновременно с алгоритмом ПРОВЕРКА ЛЕТАЮЩЕЙ ТАРЕЛКИ вплоть до момента выдачи команды “Останов” (см. последнюю ветку на рис. 84). Указанная команда ликвидирует параллельный процесс ШАБАШ ЗЛЫХ ДУХОВ, в этот момент одновременная работа заканчивается. Однако следующая команда “Пуск” запускает другой параллельный процесс — ШАБАШ ДОБРЫХ ДУХОВ, который начинает работать одновременно с алгоритмом ПРОВЕРКА ЛЕТАЮЩЕЙ ТАРЕЛКИ.

ОСОБЕННОСТИ ОПЕРАТОРОВ РЕАЛЬНОГО ВРЕМЕНИ

Мы уже говорили, что цикл ЖДАТЬ выполняется прикладной программой при участии дракон-диспетчера. Этот вывод относится ко всем операторам реального времени. Вместе с тем следует подчеркнуть, что данное утверждение относится не к языку, а к реализации системы и для разных реализаций может быть различным.

Операторы реального времени — это формальные операторы языка визуального программирования ДРАКОН-2. Однако их можно использовать и в псевдоязыке ДРАКОН-1 при неформальном изображении алгоритмов — для построения наглядных “картинок”, позволяющих легко объяснить ту или иную идею, относящуюся к системам реального времени. Примеры таких картинок представлены на рис. 88 и 89. При этом в цикле ЖДАТЬ икону “период” обычно опускают, чтобы не загромождать рисунок (см. последнюю ветку на рис. 88). Однако если длительность периода нужна для понимания, икону “период” можно сохранить (рис. 89).

В отличие от обычных вычислительных и информационных программ в программах реального времени икона “конец” может отсутствовать. Это имеет место, когда нужно организовать бесконечный цикл, который прекращается особой внешней причиной, например выключением питания системы или разрушением объекта (рис. 88, 89).

Дракон-программа может иметь более одного входа. Чтобы организовать дополнительный вход, нужно разместить икону “заголовок” над иконой “имя ветки”, как показано на рис. 84 справа. Таким образом любая ветка может быть объявлена дополнительным входом. Однако есть исключение: если несколько веток образуют веточный цикл, вход разрешается только в начало цикла. Остальные ветки конструкции “веточный цикл” не могут являться входами в программу.

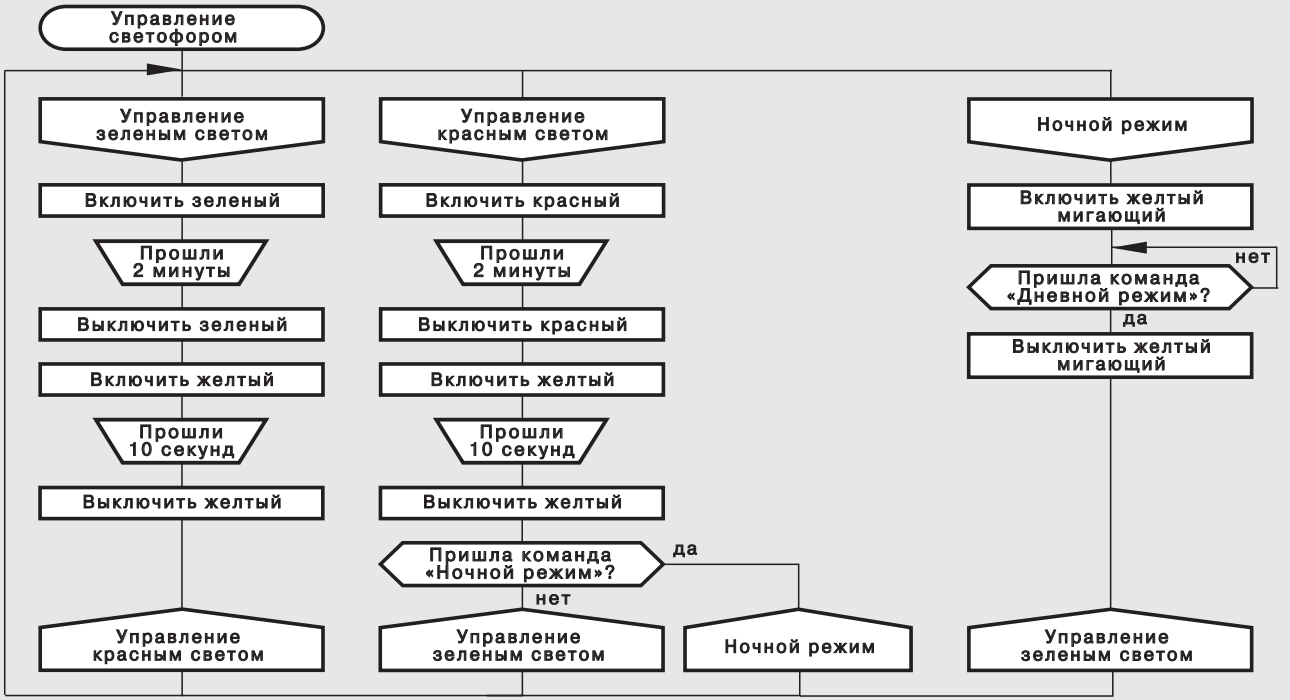


Рис. 88. **Алгоритм “Управление светофором”**

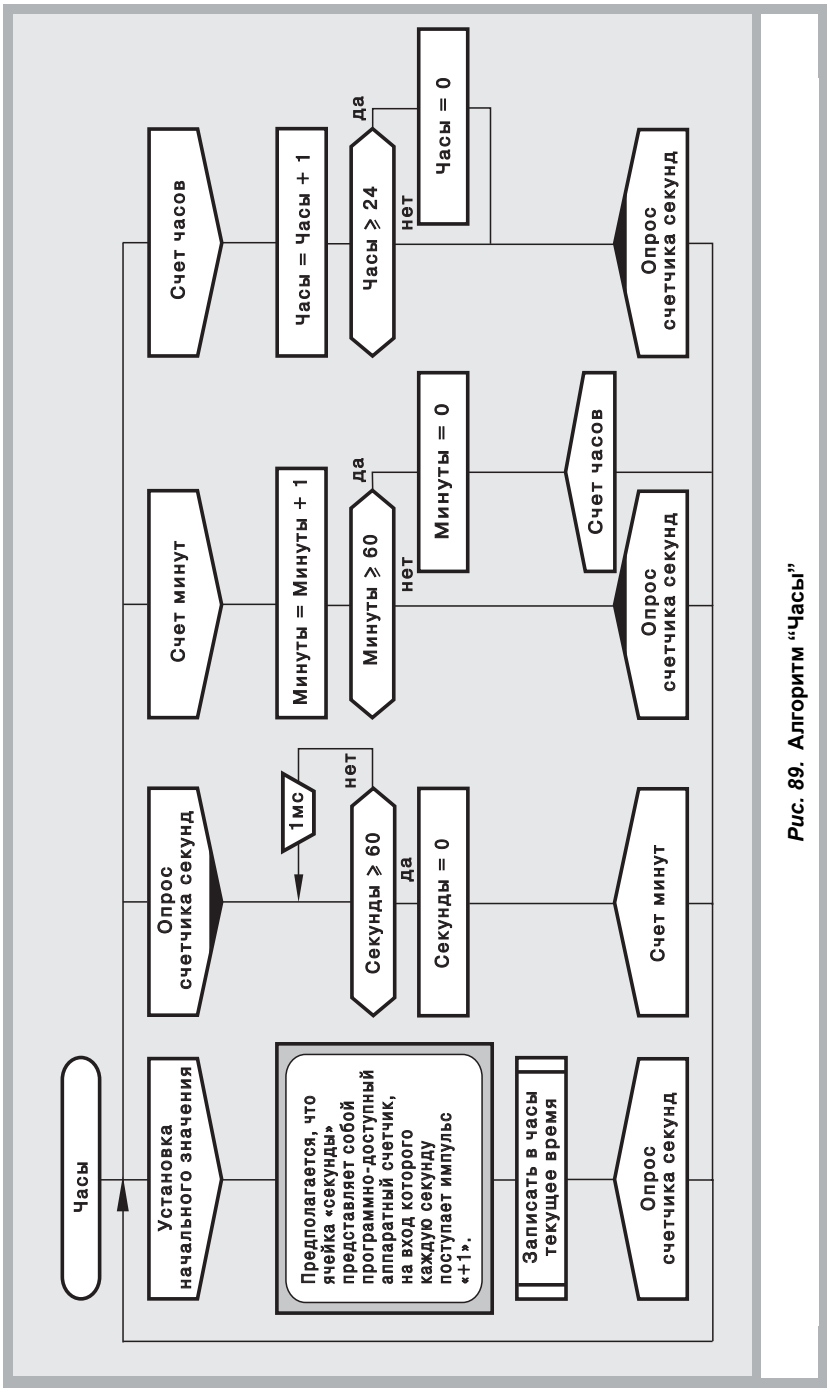


Рис. 89. Алгоритм “Часы”

ВЫВОДЫ

1. Наличие операторов реального времени резко расширяет изобразительные возможности языка ДРАКОН и позволяет использовать его при проектировании и разработке не только информационных, но и управляющих систем. Это обстоятельство существенно увеличивает область применения языка.

2. Дополнительным преимуществом является лаконичность выразительных средств, их универсальность. В языке всего пять икон реального времени, однако их алгоритмическая мощь — в сочетании с другими возможностями языка — позволяет охватить обширный спектр задач, связанных с созданием программного обеспечения для управляющих систем.

3. Важную роль играет эргономичность операторов реального времени. Как и другие операторы языка ДРАКОН, они имеют визуальный характер, что позволяет сделать операции реального времени более наглядными и легкими для понимания по сравнению с традиционной текстовой записью.

4. Четыре иконы (пауза, период, пуск таймера и синхронизатор) — “близкие родственники” в том смысле, что внутри каждой из них указывается значение времени. Эта родственная связь находит свое эргономическое отражение в том, что перечисленные операторы имеют визуальное “фамильное сходство” — все они построены (с вариациями) на основе одной и той же геометрической фигуры — перевернутой равнобедренной трапеции.

5. Операторы реального времени порождают сложные действия компьютера, связанные с частыми передачами управления между прикладной программой и операционной системой (дракон-диспетчером). Эргономическая изюминка состоит в том, что эти передачи намеренно скрыты от читателя программы, чтобы не загромождать ее текст (чертеж) второстепенными подробностями. Благодаря этому внимание читателя не отвлекается на мелочи и он имеет возможность сосредоточиться на главном, поскольку дракон-схема предоставляет ему ясную, четкую и целостную картину алгоритмического процесса, очищенную от “мелкого мусора”.

ГЛАВА 12

ДРУЖЕЛЮБНОЕ ПРОГРАММИРОВАНИЕ

Индустрии программирования необходимо чудо — чудо, которое воплотило бы в жизнь мечту о быстрой и легкой разработке программ.

Том Мануэль

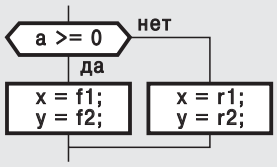
ГИБРИДНЫЙ ЯЗЫК ПРОГРАММИРОВАНИЯ ДРАКОН-СИ

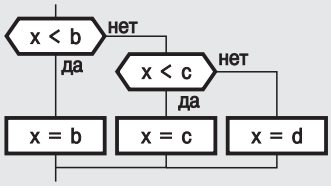
Предположим, нужно построить систему визуального программирования на гибридном языке ДРАКОН-СИ. Задачу можно решить, например, с помощью трех программ: дракон-редактора, дракон-конвертора и компилятора языка СИ. Пользователь с помощью дракон-редактора рисует на экране компьютера программу на языке ДРАКОН-СИ (рис. 90, правая графа). Затем дракон-конвертор преобразует внутреннее представление графических кодов в исходный текст языка СИ (рис. 90, средняя графа), после чего стандартный компилятор СИ превращает исходный текст в объектный код.

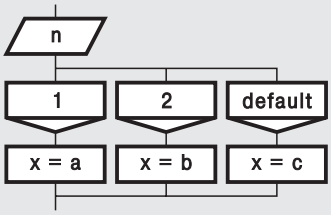
Чтобы лучше уяснить преимущества языка ДРАКОН-СИ, произведем мысленно обратное преобразование. Как видно из рис. 90, при преобразовании текстовой программы в визуальную исходный текст СИ-программы разбивается на две части. Операторы присваивания, условные выражения и декларативные описания почти без изменения переносятся в визуальную программу и размещаются внутри ее икон. Остальные текстоэлементы языка СИ (которые можно назвать удаляемыми или “паразитными”) становятся ненужными, превращаясь в графические линии и ключевые слова “да” и “нет” (*yes* и *no*). Рисунок 90 показывает, что список паразитных (удаляемых) элементов языка СИ оказывается внушительным: он включает все ключевые слова в примерах 1—7 кроме *default*, все фигурные, круглые и косые скобки, двоеточия, метки, комментарии в примерах 3—5, и кроме того, точки с запятой в примерах 2, 3, 7 и отчасти 6.

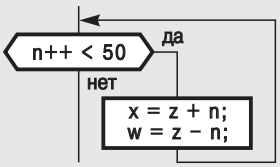
Таким образом, чтобы построить язык ДРАКОН-СИ, надо по определенным правилам соединить визуальный синтаксис ДРАКОНА с текстовым синтаксисом языка СИ, удалив из последнего все элементы, функции которых реализуются визуальными операторами ДРАКОНА. Пара языков СИ и ДРАКОН-СИ эквивалентна в том смысле, что может быть построен конвертор, выполняющий как прямое, так и обратное преобразование. Такой конвертор может превращать исходный текст программы на языке ДРАКОН-СИ (рис. 90, правая графа) в эквивалентную СИ-программу (рис. 90, средняя графа), и наоборот.

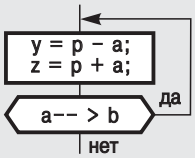
Операторы языка СИ	Примеры программ на языке СИ	Примеры программ на языке ДРАКОН-СИ

① if-else	if (a >= 0) {
	  x = f1;
	  y = f2;
	}
	else {
	  x = r1;
	  y = r2;
	}
		

② if-elseif-else	if (x < b)
	  x = b;
	elseif (x < c)
	  x = c;
	else
	  x = d;
		

③ switch, case, break, default	switch (n) {
	  case 1:
	  x = a;
	  break;
	  case 2:
	  x = b;
	  break;
	  default:
	  x = c;
	} /* switch */
		

④ while	while (n++ < 50) {
	  x = z + n;
	  w = z - n;
	} /* while */
		

⑤ do-while	do {
	  y = p - a;
	  z = p + a;
	} while (a-- > b);
	/* do while */
		

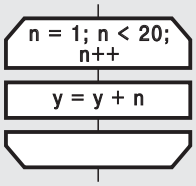
⑥ for	for (n=1; n<20; n++) y = y + n;	

Рис. 90. **Примеры программ на языке СИ и эквивалентные им программы на языке ДРАКОН-СИ**

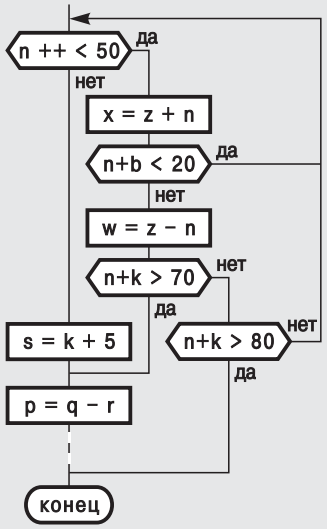
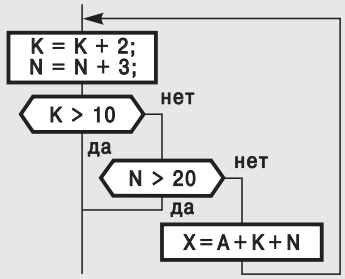
⑦
while,
continue,
goto,
return,
	while (n ++ < 50) {
	x = z + n ;
	if (n + b < 20)
	continue ;
	w = z − n ;
	if (n + k > 70)
	go to m1 ;
	if (n + k > 80)
	return ;
	} /* while */
	s = k + 5 ;
	m1 : p = q − r ;
	.....
	} /* end */
		

Рис. 90 (окончание)

Ключевые слова	Программа на языке МОДУЛА-2	Программа на языке ДРАКОН-МОДУЛА

LOOP,
IF-THEN-
ELSEIF-
ELSE,
EXIT,
END
	LOOP
	K := K + 2;
	N := N + 3;
	IF K > 10 THEN
	EXIT
	ELSEIF N>20 THEN
	EXIT
	ELSE
	X := A + K + N;
	END
	END
		

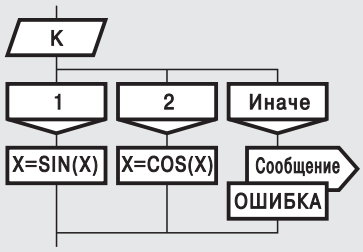
Ключевые слова	Программа на языке ПАСКАЛЬ	Программа на языке ДРАКОН-2
IF-THEN- ELSE, CASE-OF, OR, END, WRITELN	IF (K=1 OR K=2) THEN CASE K OF 1: X := SIN(X); 2: X := COS(X); END (* CASE *) ELSE WRITELN ("ОШИБКА");	

Рис. 91. **Примеры программ на языках МОДУЛА-2 и ПАСКАЛЬ и эквивалентные им программы на языках ДРАКОН-МОДУЛА и ДРАКОН-2**

Создание любого гибридного языка (например, ДРАКОН-СИ) вряд ли стоит считать оригинальной разработкой, так как последний почти полностью сохраняет концепцию, структуру, типы данных и другие особенности исходного языка (СИ). Правильнее говорить о том, что построение гибридного языка (ДРАКОН-СИ) есть технический прием, при котором в строго определенном числе случаев текстовая нотация исходного языка заменяется на визуальную. Однако этот технический прием позволяет существенно улучшить эргономический облик исходного языка.

ГИБРИДНЫЙ ЯЗЫК ПРОГРАММИРОВАНИЯ ДРАКОН-МОДУЛА

Обратимся к верхнему примеру на рис. 91. В средней графе представлена программа на языке МОДУЛА-2, в правой — эквивалентная ей программа на языке ДРАКОН-МОДУЛА. В левой графе приводится список ключевых слов, которые используются в модула-программе и являются "жизненно необходимыми" для языка МОДУЛА, но которые совершенно не нужны в дракон-программе.

С эргономической точки зрения, эти и многие другие ключевые слова, присутствующие в текстовых языках, есть не что иное как визуальные помехи, притягивающие к себе внимание читателя и отвлекающие его внимание от содержательной стороны дела. Эргономическое преимущество ДРАКОНА состоит в том, что вместо ключевых слов используется визуальный образ, который воспринимается читателем бессознательно, на интуитивном уровне, в то время как канал сознательного внимания действует более продуктивно — для восприятия наиболее важных, содержательных аспектов задачи.

ПРИМЕР ЭРГОНОМИЧЕСКОЙ ОПТИМИЗАЦИИ ПРОГРАММЫ

На рис. 91 (внизу, в средней графе) написана программа на языке ПАСКАЛЬ. Действуя по аналогии с предыдущими примерами, ее можно легко преобразовать в программу на языке ДРАКОН-ПАСКАЛЬ. Для этого нарисуем визуальный оператор "развилка" и в иконе "вопрос" поместим запись

$K = 1$ OR $K = 2$

Нижний выход иконы "вопрос" пометим словом "да" и присоединим к нему переключатель с двумя иконами "вариант", а правый выход (ответ "нет") подключим к иконе "вывод", в которой сверху напишем WRITELN, снизу — ОШИБКА. В итоге получим дракон-схему, которая несомненно является совершенно правильным решением поставленной задачи. (Для наглядности советуем читателю выполнить описанные построения на бумаге.)

А теперь изменим условие задачи. Попытаемся создать программу, которая была бы не только эквивалентной паскаль-программе на рис. 91, но и эргономически оптимальной для русскоязычного читателя. Искомая программа, написанная на языке ДРАКОН-2, представлена на том же рисунке внизу справа.

Бросается в глаза структурное различие между программами. Паскаль-программа содержит две конструкции: *if-then-else* и *case-of*. Эргономическая оптимизация состоит в том, что в дракон-программе используется всего один визуальный оператор (переключатель с тремя вариантами), который тем не менее "в одиночку" выполняет те же самые функции, что и два текстовых оператора языка ПАСКАЛЬ. В итоге сложное условие $K = 1$ OR $K = 2$ и другие излишества паскаль-программы устраняются, а дракон-схема заметно упрощается и становится лаконичной, прозрачной, элегантной.

ДИАЛОГОВЫЕ ПРОГРАММЫ

Продолжим изложение одного из возможных подходов к построению языка программирования ДРАКОН-2. Еще раз напомним: читатель не найдет здесь описания языка. Наша цель гораздо скромнее: показать, что формализация текстового синтаксиса для языка ДРАКОН вполне осуществима, и привести несколько примеров, подтверждающих эту мысль.

Рассмотрим диалоговые программы на рис. 92 и 93, имеющие улучшенные дидактические (педагогические) характеристики. Для этого применяется обширный набор эргономических средств. В частности, при заполнении иконы "комментарий" используется *зонирование текста*. Текст комментария для облегчения восприятия пространственно делится на две зоны, которые, во-первых, имеют четко очерченные и легко различимые границы, во-вторых, отличаются по цвету фона (белый и серый). В серой зоне помещается текст, появляющийся на экране компьютера, в белой — пояснения к нему. Отграничение экранного текста от пояснений облегчает чтение комментариев и улучшает их понимаемость.

Эргономический прием "зонирование текста" полезно использовать не только в комментариях, но и в других случаях, например в операторах ввода-вывода.

Оператор "Сообщение"

Оператор "Сообщение" служит для вывода информации на экран компьютера. Он содержит икону "вывод", на верхнем этаже которой помещают ключевое слово "Сообщение", на нижнем — выводимую информацию. При описании последней применяется зонирование текста: в серой зоне пишут имена переменных или выражения (значения которых следует вывести на экран), в белой зоне — постоянную информацию (которая выводится на экран без изменений). Признаком новой

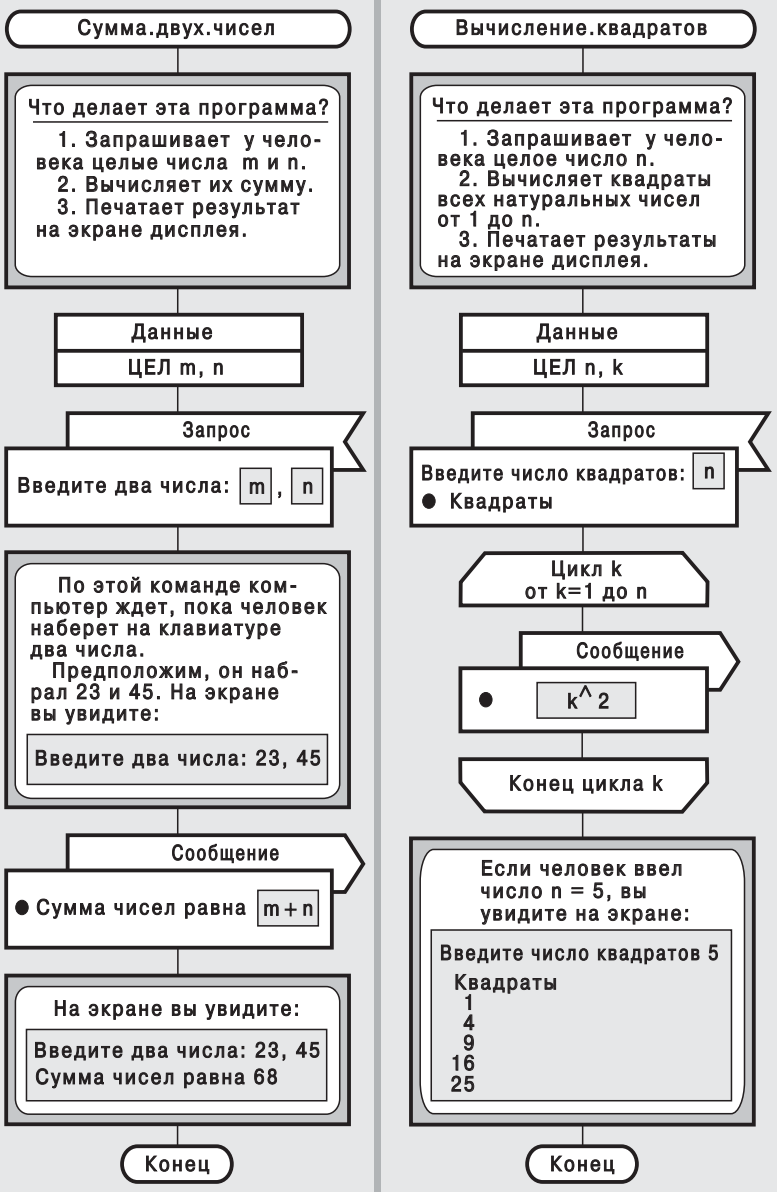


Рис. 92. **Программа, вычисляющая сумму двух чисел**

Рис. 93. **Программа, вычисляющая квадраты натуральных чисел**

строки служит черный кружок. Например, на рис. 92 с помощью оператора “Сообщение” на экран выводится фраза “Сумма чисел равна” и значение выражения $m + n$.

Оператор “Запрос”

Оператор “Запрос” осуществляет ввод в компьютер значений переменных, вывод на экран постоянной информации, имен переменных и введенных значений. В верхней части иконы “ввод” пишут ключевое слово “Запрос”, в нижней — вводимую и выводимую информацию. Здесь также присутствует зонирование текста: в серой зоне указывают имена переменных, подлежащих вводу в компьютер, в белой — помещают постоянную информацию.

Предположим, нужно ввести значения $m = 23$ и $n = 45$ (рис. 92). Делается это, например, так: подводят курсор в зону m, набирают на клавиатуре число 23 и нажимают клавишу “возврат каретки”. При этом зона m на экране гаснет и вместо нее загорается число 23. Значение n вводится аналогично. Таким образом, оператор “Запрос” запрашивает у пользователя значения переменных, записывает их в память и одновременно отображает на экране вместе с постоянной информацией (если последняя указана на нижнем этаже оператора “Запрос” в белой зоне).

Описание данных

Для описания данных служит икона “полка”. На верхнем этаже пишут ключевое слово “Данные”, на нижнем — описание данных. Например, на рис. 92 в иконе “полка” указано, что переменные m и n имеют тип “целый”.

Можно предложить и другой способ: описание данных выносят за рамки дракон-схемы и помещают в отдельную таблицу.

ИДЕНТИФИКАТОРЫ

Приведем правила записи идентификаторов.

- Длина идентификатора 1...32 символа.
- Разрешается использовать любые русские и латинские буквы, цифры, точку и, возможно, специальные символы.
- Первый символ должен быть буквой (не цифрой и не точкой).
- Внутри идентификатора запрещается использовать пробелы.
- Слова следует разделять точками, чтобы облегчить чтение.
- Запрещается использовать сокращение слов, если длина идентификатора меньше 32 символов.
- Если длина идентификатора больше 32 символов, надо заменить некоторые слова сокращениями или уменьшить число слов.
- Нужно стремиться придумывать доходчивые идентификаторы, позволяющие легко уяснить смысл понятия, чтобы читатель быстро понял суть дела.

Примеры правильных идентификаторов

Номер.вагона.скорого.поезда

Номер.вагона.пассажир.поезда
Цена.билета.поездом.до.Магадана
Цена.билета.самолет.до.Магадана

Примеры неправильных идентификаторов

Номер.вагона.пассажирского.поезда	(здесь 33 символа, а можно не более 32)
Число.вагонов товарного поезда	(используются пробелы)
3-й.запуск.аварийного.насоса	(здесь две ошибки: первый символ — цифра; кроме того, есть дефис)

Пример сокращения длины сложного понятия

Предположим, нужно создать идентификатор для следующего понятия: “Радиус-вектор центра Земли в центре взлетно-посадочной полосы в посадочной системе координат”. Словесное описание понятия содержит 92 символа. Задача состоит в том, чтобы сократить 92-символьное описание до 32-символьного, сохранив по возможности ясный смысл понятия.

Сокращение проведем по следующему плану:

- “Радиус-вектор центра Земли” заменим на “Радиус.земли”.
- Вместо “В центре взлетно-посадочной полосы” напишем “на.полосе”.
- “В посадочной системе координат” заменим на ПСК, поскольку такое сокращение является общеупотребительным в коллективе разработчиков данной системы.

В итоге получим 26-символьный идентификатор

```
Радиус.земли.на.полосе.ПСК
```

который сохраняет почти все опорные слова исходного понятия и обеспечивает довольно высокую понимаемость.

Правила записи арифметических выражений в операторах присваивания

Следует различать два случая. Если выражение простое, рекомендуется использовать 32-символьные идентификаторы и “вертикальную” запись математических формул, как показано на рис. 94 и 95.

Однако если речь идет о сложных математических вычислениях, описанный способ не годится, поскольку “вертикальные” формулы с 32-символьными идентификаторами не позволяют читателю увидеть математическую структуру вычислений, отвлекая его внимание на чтение длинных идентификаторов, которые парадоксальным образом превращаются из полезной подсказки в свою противоположность и начинают играть негативную роль визуальной помехи. Таким образом, возникает эргономический тупик: короткие идентификаторы не позволяют

быстро уяснить смысл понятий, а длинные — затемняют структуру сложных формул.

В качестве одного из возможных подходов к развязыванию этого гордиева узла можно предложить план из трех пунктов.

- Для каждого математического понятия предусматриваются два идентификатора: длинный (32-символьный) и короткий (алиас).
- В арифметических выражениях используются только алиасы, что делает структуру формул прозрачной.
- В начале программы предусматривается икона “комментарий”, в которой размещается таблица соответствий между алиасами и длинными идентификаторами. Эта таблица играет роль шпаргалки, которая находится в одном поле зрения с операторами присваивания и позволяет быстро вспомнить, что означает тот или иной алиас.

ОБРАБОТКА МАССИВОВ

На рис. 94 и 95 даны примеры программ, в которых имеются операции с массивами.

Описание данных размещается на нижнем этаже иконы “полка”.

Запись

```
МАССИВ ВЕЩ Вес.кролика [100]
```

означает, что задан одномерный массив с именем “Вес.кролика”, содержащий 100 элементов, каждый из которых является вещественным числом.

Основным элементом обеих программ служит цикл ДЛЯ. Рассмотрим правила оформления цикла. В иконе “начало цикла ДЛЯ” в верхней строке пишут слово “Цикл” и после пробела односимвольный алиас, обозначающий переменную цикла (буква k на рис. 94, 95). В нижней строке указывают диапазон ее изменения, например,

```
от k = 1    до    100
```

Но что обозначает буква k? Каков физический смысл переменной цикла в данной прикладной программе? Авторы многих программ забывают (или не считают нужным) ответить на этот вопрос, в результате чего программа нередко превращается в ребус. Чтобы этого не

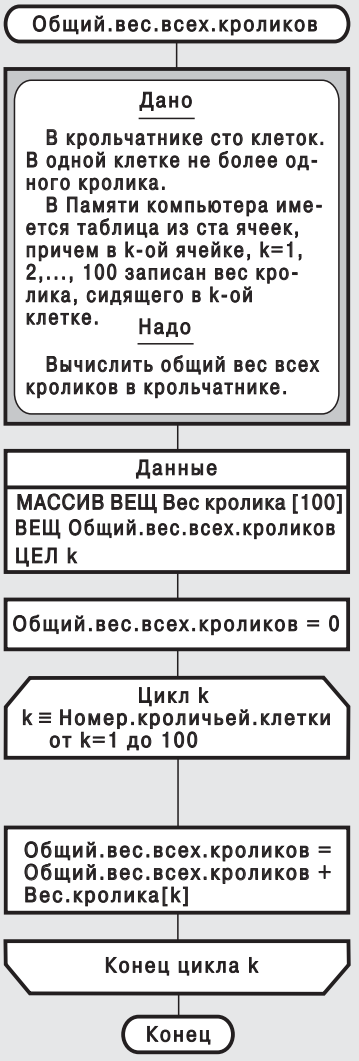


Рис. 94. **Программа, вычисляющая общий вес всех кроликов в крольчатнике**

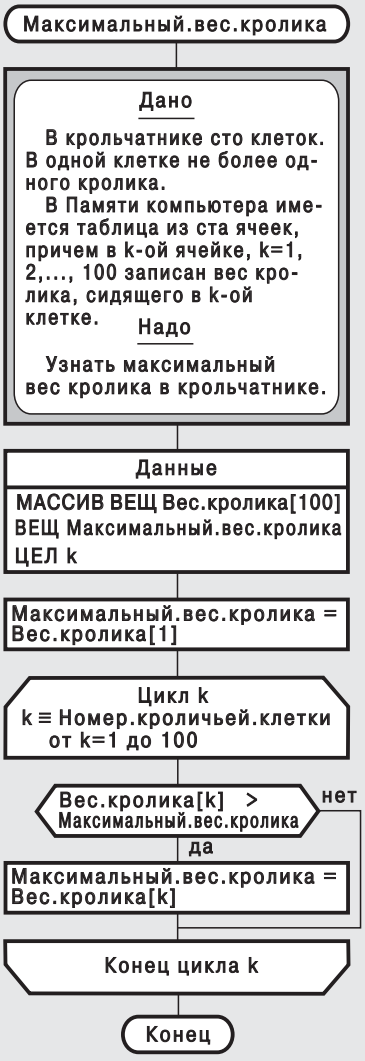
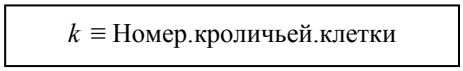


Рис. 95. **Программа, определяющая максимальный вес одного кролика в крольчатнике**

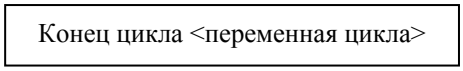
случилось, в средней строке следует написать формализованный комментарий, например



Знак тождественного равенства ≡ показывает, что после него следует имя-комментарий, т. е. комментарий, который пишется по правилам записи идентификаторов.

Эргономический "навар" формализованного комментария включает два преимущества. Во-первых, он позволяет устранить традиционную "забывчивость" программистов и по-человечески объяснить читателю смысл абстрактного идентификатора: дескать, *k* — это номер кроличьей клетки. Во-вторых, что немаловажно, объяснение размещается на поле чертежа именно там, где нужно (в иконе "начало цикла ДЛЯ"), по принципу "дорого яичко ко Христову дню". Это значит, что читатель получает ответ моментально — в ту самую секунду, когда он впервые увидел алиас *k* и в его голове забрезжил вопрос: а что такое *k*?

В иконе "конец цикла ДЛЯ" делают запись



Смысл операторов, организующих обработку массивов, ясен из рис. 94 и 95 и не требует пояснений.

АБСТРАКТНЫЕ ДРАКОН-СХЕМЫ

В этом параграфе мы рассмотрим преобразование визуальной программы на языке ДРАКОН-2 в текстовую программу на БЕЙСИКе. Это преобразование полезно в двух отношениях: оно позволит глубже уяснить суть визуализации и познакомиться с важным понятием *абстрактной дракон-схемы*.

В качестве примера возьмем школьную программу под названием "Игра.угадайка" и напишем ее на языке ДРАКОН-2 (рис. 96). Затем полностью устраним из нее текст и получим чертеж-слепыш, который называется "абстрактная дракон-схема" (рис. 97). Эта схема представляет собой инвариант программы, который можно за два шага преобразовать в программу на любом языке программирования.

Выберем в качестве цели язык БЕЙСИК и приступим к делу. На первом шаге заполним пустые иконы абстрактной схемы текстом на языке БЕЙСИК. В результате получится эквивалентная программа на языке ДРАКОН-БЕЙСИК (рис. 98). На втором шаге переходим к обычной бейсик-программе (мы сознательно выбрали старомодную версию БЕЙСИКа, чтобы для разнообразия продемонстрировать использование операторов *goto* при описании эквивалента дракон-программы) — см. рис. 99.

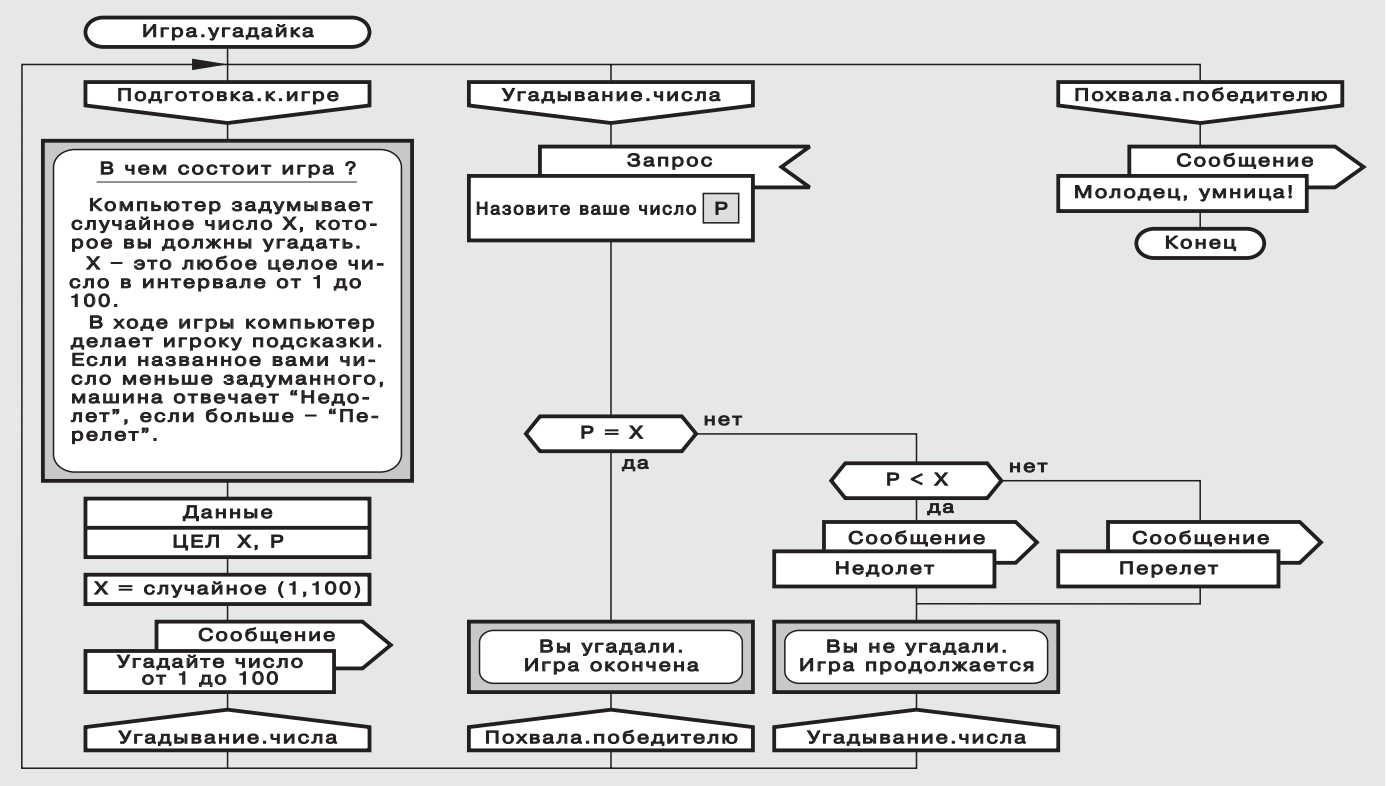


Рис. 96. **Программа игры "Угадайка" на языке ДРАКОН-2**

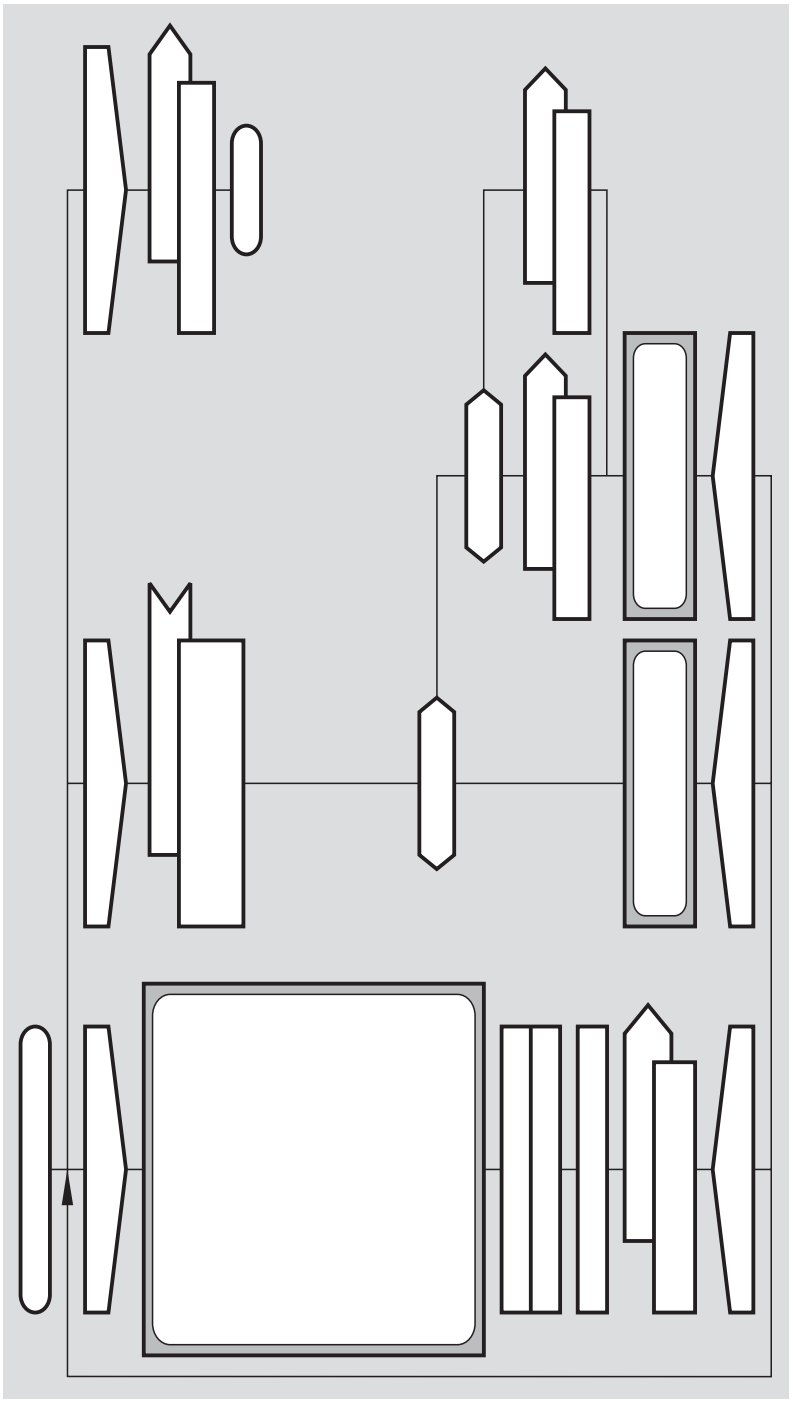


Рис. 97. **Абстрактная дракон-схема**

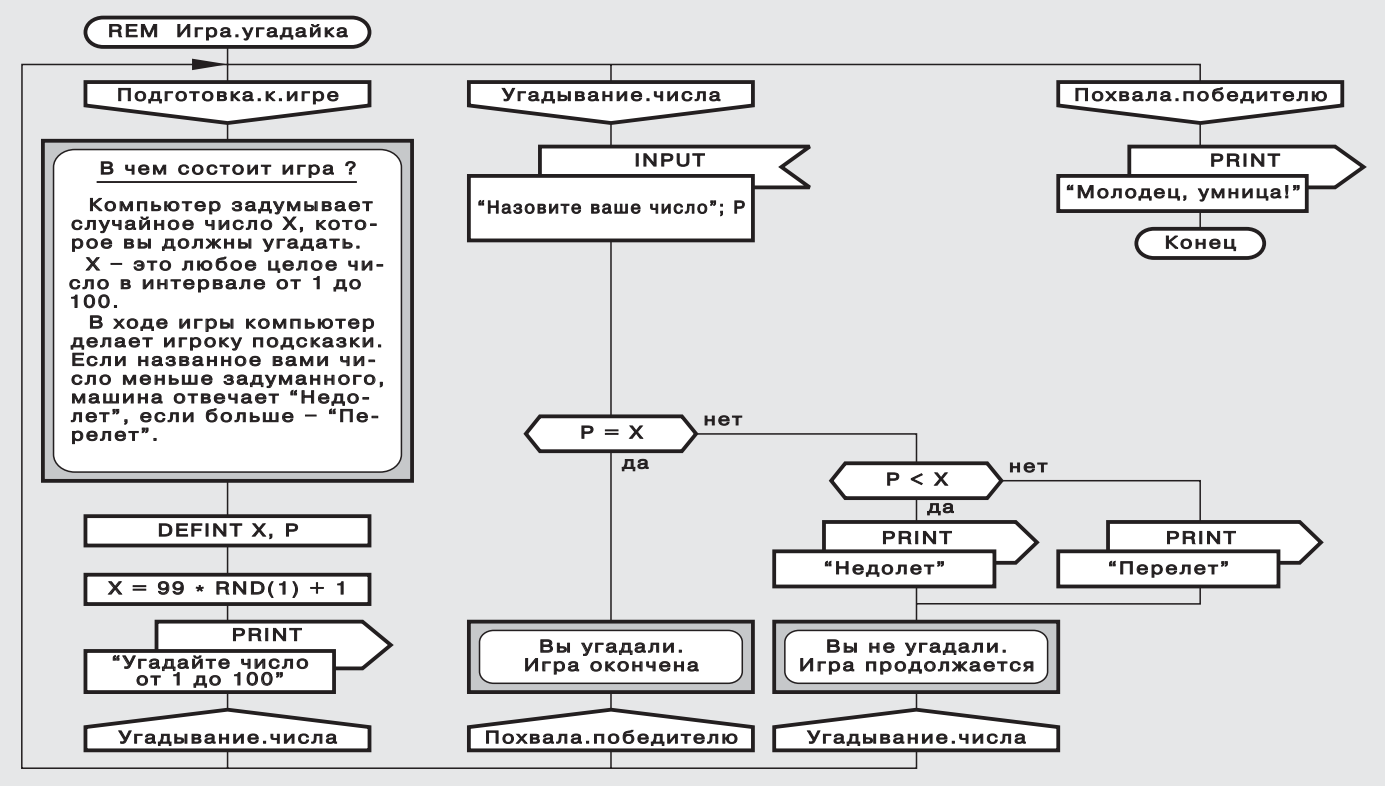


Рис. 98. **Программа игры "Угадайка" на языке ДРАКОН-БЕЙСИК**

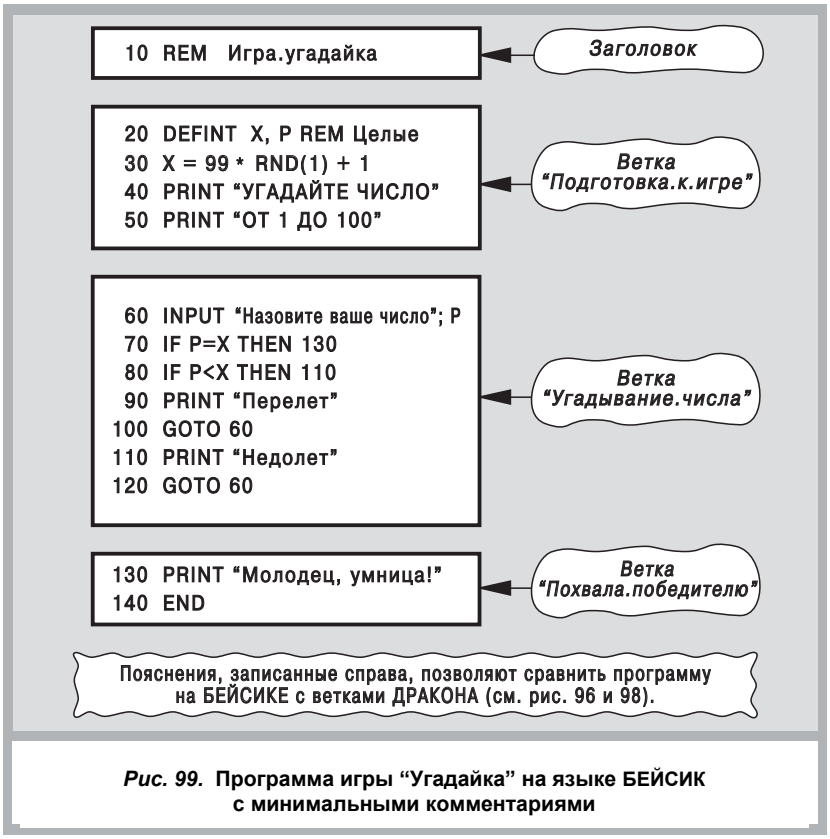


Рис. 99. **Программа игры "Угадайка" на языке БЕЙСИК с минимальными комментариями**

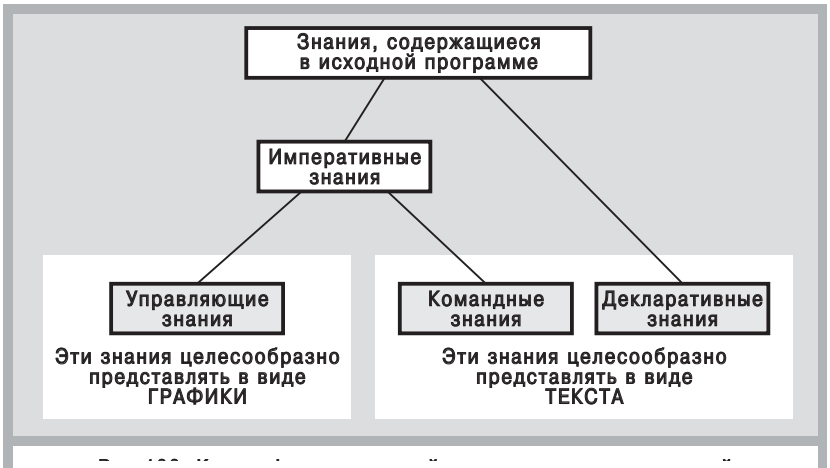


Рис. 100. **Классификация знаний, содержащихся в исходной программе, с точки зрения ДРАКОН-концепции**

ФИЛОСОФИЯ ЯЗЫКА ДРАКОН

Любой императивный язык (СИ, ПАСКАЛЬ, АДА, МОДУЛА, БЕЙСИК и т. д.) можно разбить на три части, три относительно самостоятельных языка: маршрутный, командный и декларативный.

Маршрутный язык — совокупность управляющих операторов. *Командный язык* содержит все неуправляющие операторы, например оператор присваивания, правила записи арифметических и логических выражений, идентификаторов, ключевые слова и т. д. *Декларативный язык* служит для описания данных, классов и др.

Поясним сказанное на примере. Абстрактная дракон-схема, приведенная на рис. 97, есть некоторая "фраза" маршрутного языка. Чтобы сделать ее содержательной, внутри икон следует поместить текст. Этот текст пишется на командном языке. Однако описания данных и классов иногда целесообразно вынести за рамки дракон-схемы и поместить их где-нибудь в другом месте, например, в виде отдельной записи или таблицы.

Отсюда вытекает принцип разграничения трех подъязыков. Маршрутный язык — это язык "картинок" (абстрактных дракон-схем, в которых полностью отсутствует текст). Командный язык служит для записи текста внутри дракон-схемы, декларативный — для тех записей, которые можно вынести за ее пределы.

Конструируя очередной язык дракон-семейства, можно выбирать командный и декларативный подъязыки любым способом (заимствуя из других языков или выдумывая заново). Благодаря этому обеспечивается богатство возможностей ДРАКОНА и гибкая настройка на различные приложения. Таким образом, дракон-семейство имеет только одно жесткое звено — маршрутный язык, который есть не что иное, как визуальные компоненты ДРАКОНА (визуальный синтаксис и семантика).

Маршрутный язык — это визуальный стандарт дракон-семейства, поддерживаемый визуальным дракон-редактором (см. гл. 14), который имеет небольшие размеры и легко запоминается. Он является неизменной визитной карточкой ДРАКОНА, его стандартизованным зрительным образом (рис. 97).

КЛАССИФИКАЦИЯ ЗНАНИЙ

Любая программа есть некоторая сумма знаний, которую можно расчленить на императивную и декларативную часть. Изложенные выше соображения позволяют уточнить этот тезис. Новый взгляд на проблему представлен на рис. 100 и сводится к следующему:

- Для анализа знаний, содержащихся в исходном тексте компьютерной программы, написанной на императивном языке, целесообразно использовать две классификации: базовую и альтернативную.
- *Базовая классификация* состоит в том, что все знания, содержащиеся в исходной программе, разбиваются на императивные и декларативные.
- В свою очередь императивные знания делятся на управляющие и командные.

! В качестве критерия для альтернативной классификации предлагается вопрос: какие средства лучше использовать для представления знаний — графику или текст?

! Ответ состоит в следующем. Для представления управляющих знаний лучше применять графику (маршрутный язык), для командных и декларативных знаний — текст.

! Таким образом, при *альтернативной классификации* знания делятся на визуальные (управляющие) и текстовые (командные и декларативные).

Напоследок добавим: использование текста для представления сложных управляющих знаний выглядит столь же нелепо, как попытка описать географическую карту словами. Тот факт, что текст все еще применяется для этой цели, можно объяснить только одним: программирование намного моложе географии!

ВЫВОДЫ

1. Если в нашем распоряжении имеется формальный визуальный синтаксис, то для построения визуального языка программирования достаточно построить формальный текстовый синтаксис. Мы убедились, что эта задача вполне разрешима, причем несколькими способами. В итоге образуется семейство языков программирования как оригинальных (ДРАКОН-2), так и гибридных (ДРАКОН-СИ, ДРАКОН-МОДУЛА, ДРАКОН-ПАСКАЛЬ, ДРАКОН-БЕЙСИК и т. д.).

2. Можно утверждать, что понимаемость визуальных языков существенно выше, чем понимаемость их текстовых собратьев. Поэтому во всех случаях, когда понимаемость рассматривается как главный критерий качества программ (а таких случаев немало), визуальные языки оказываются вне конкуренции. Здесь уместна оговорка: сам по себе термин “визуальный” ничего не гарантирует. Успех дела достигается за счет тщательного и скрупулезного применения методов науки о человеческих факторах (эргономики). Если говорить точнее, речь идет о синтезе методов информатики и эргономики, формировании нового междисциплинарного направления — инфоэргономики, перестройке всего здания современного программирования на эргономической основе.

3. Создание любого языка программирования следующего поколения должно начинаться с анализа эргономических требований и заканчиваться оценкой полученных эргономических характеристик языка. Одно из основных препятствий для реализации этого плана состоит в инерции мышления многих специалистов, недооценке важности эргономических методов. Чтобы изменить сложившиеся стереотипы мышления, нужно внести серьезные изменения в программу и методы преподавания информатики в школе и вузе.

Глава 13

ЧЕЛОВЕЧЕСКАЯ ДЕЯТЕЛЬНОСТЬ И ФОРМАЛИЗАЦИЯ ЗНАНИЙ: ЖИВОПИСНЫЕ ПРИМЕРЫ

Процесс формализации профессиональных знаний... — исторически новая форма интеллектуальной деятельности.

Григорий Громов

ЧТО ТАКОЕ ПРОФЕССИОНАЛЬНЫЕ ЗНАНИЯ?

Профессиональные знания охватывают всю совокупность сведений, которые человек использует в своей профессиональной деятельности. Мы ограничим это понятие с двух сторон. Во-первых, исключим из рассмотрения профессиональную деятельность в области религии, искусства, спорта и некоторых других сфер, сосредоточив внимание на научной, технической, производственной, управленческой, экономической, медицинской, учебной и сходными с ними деловыми видами деятельности. Во-вторых, нас будут интересовать лишь те профессиональные знания, которые целесообразно в той или иной степени формализовать и представить в письменном виде на бумаге или экране дисплея.

Профессиональные знания должны в конечном итоге вести к желаемому (теоретическому или практическому) результату, а также удовлетворять многим другим требованиям. Например, технология закалки кинжала (рис. 101), хотя и позволяет получить искомый результат, однако с современной точки зрения выглядит дикой, нелепой и бесчеловечной. Причин тому две: на заре человечества моральные нормы были неразвиты, а главное — указанная технология опирается не на научные знания, а на мифологические объяснительные схемы. Древние “технологи” были искренне убеждены, что сила раба “переходит” в кинжал и улучшает его боевые качества.

В данной главе под *формализацией знаний* будем понимать представление человеческих знаний на формальном или частично формальном языке. Учитывая сказанное ранее (гл. 3), исключим из анализа декларативные профессиональные знания и ограничимся проблемой формализации технологических (императивных) знаний.

План дальнейшего изложения таков. Мы рассмотрим ряд примеров из самых различных, очень непохожих друг на друга областей профессиональной деятельности и покажем, что язык ДРАКОН “повсюду

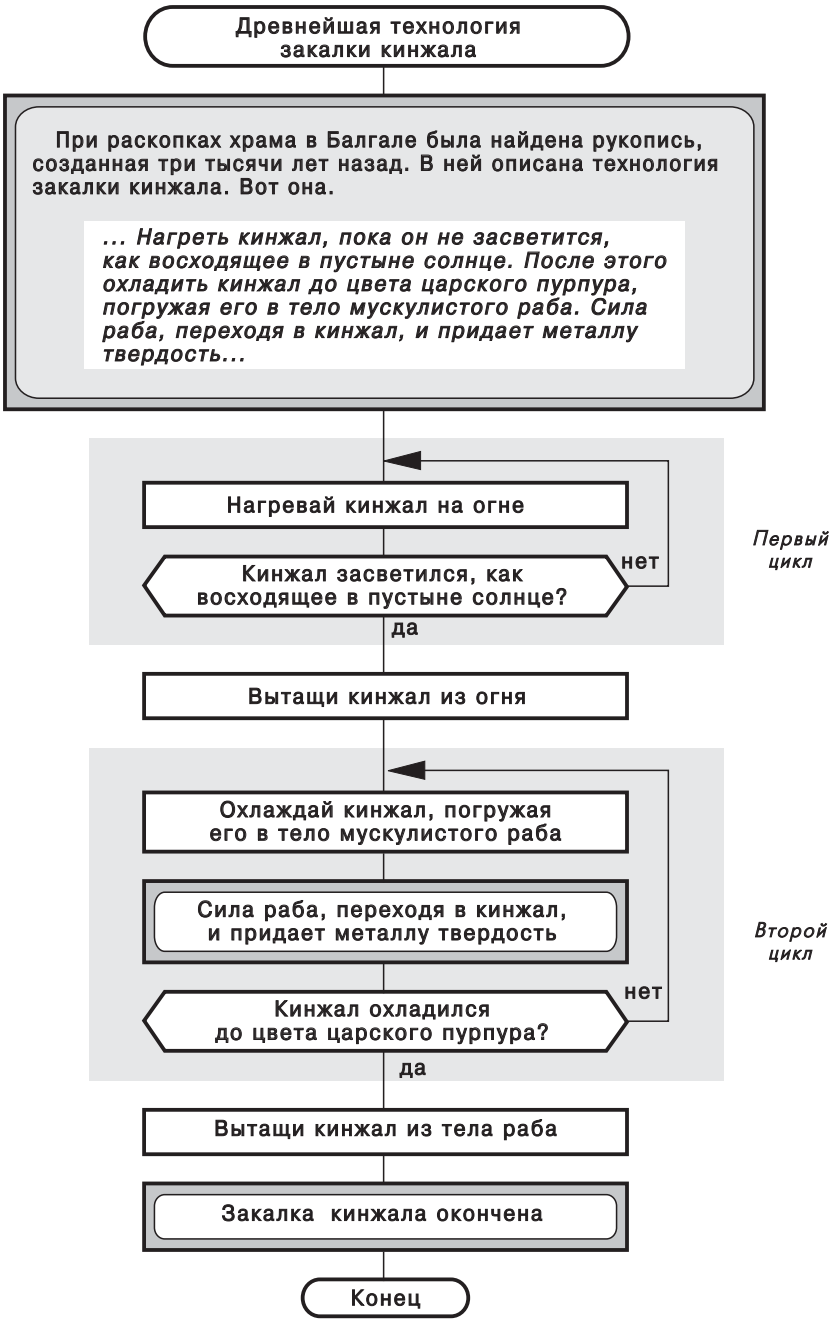


Рис. 101. **Чудовищный древний алгоритм с двумя циклами**

молодец” — он позволяет эффективно формализовать императивные знания во многих, хотя и не во всех ситуациях. Что это дает? Поскольку процесс формализации знаний оказывается чрезвычайно легким, он становится доступным практически для любого человека, который хорошо знает свое дело. Это означает, что с появлением языка ДРАКОН каждый специалист приобретает новые возможности:

- он может формализовать свои знания *сам*, без помощи инженеров по знаниям или программистов, т. е. воспользоваться всеми благами формализации знаний;
- он может выражать свои мысли на своем родном профессиональном языке, но в строгом формализованном виде.

В результате специалист получает мощное средство делового общения, ибо благодаря ДРАКОНУ его родной профессиональный язык каким-то чудесным образом стал очень похож на профессиональные языки других специальностей.

УЧЕБНЫЕ ЭКСПЕРТНЫЕ СИСТЕМЫ

Рассмотрим школьную задачу по химии за 10-й класс. Учитель берет одно из шести химических удобрений и помещает его в колбу. Что в колбе — неизвестно. Это может быть аммиачная селитра, натриевая соль, сульфат аммония, суперфосфат, сильвинит или калийная соль. Нужно выполнить эксперимент, позволяющий узнать, какое именно вещество находится в колбе.

Опыт делают в два этапа. Сначала идут подготовительные действия:

1. Положить удобрение в три сосуда.
2. В первый сосуд добавить серную кислоту H_2SO_4.
3. Во второй сосуд добавить раствор хлорида бария BaCl.
4. В третий сосуд добавить щелочь.

После этого анализируются результаты и определяется неизвестное вещество на основании табл. 3.

В методических указаниях по курсу “информатика” для средних школ рекомендуется использовать этот пример для ознакомления учащихся с принципом работы профессиональных экспертных систем. Для этой цели школьникам предлагается изучить упрощенную учебную экспертную систему в виде программы на языке БЕЙСИК. Функционирование учебной экспертной системы реализуется в диалоге ученика и системы. Экспертная система задает ученику серию вопросов, анализирует ответы и сравнивает с хранящимися в ней фактами. Система производит логический вывод и формирует ответ на интересующий пользователя вопрос — в данном случае сообщает ученику название удобрения.

Таблица 3

Название вещества	Внешний вид	Результат взаимодействия с		
		H_2SO_4	BaCl	щелочью
Аммиачная селитра	Белые кристаллы	Бурый газ	—	Запах аммиака
Натриевая селитра	Бесцветные кристаллы	Бурый газ	Помутнение	—
Сульфат аммония	Светло-серые кристаллы	—	Белый осадок	Запах аммиака
Суперфосфат	Светло-серый порошок	—	Белый осадок	—
Сильвинит	Розовые кристаллы	—	—	—
Калийная соль	Бесцветные кристаллы	—	—	—

Учебная экспертная система (программа на языке БЕЙСИК)

```
10 REM Распознавание удобрений
20 PRINT «При взаимодействии с серной кислотой выделяется бурый газ?»
30 INPUT A$
40 IF A$="да" THEN GOSUB 100 ELSE GOSUB 200
50 PRINT «Данное удобрение — »;X$
60 END
100 REM взаимодействие со щелочью
110 INPUT «При взаимодействии с раствором щелочи ощущается запах аммиака?»;B$
120 IF B$="да" THEN X$="аммиачная селитра" ELSE X$="натриевая селитра"
130 RETURN
200 REM взаимодействие с раствором хлорида бария
210 INPUT «При взаимодействии с раствором хлорида бария и уксусной кислотой выпадает белый осадок?»;C$
220 IF C$="да" THEN GOSUB 300 ELSE GOSUB 400
230 RETURN
300 REM взаимодействие с раствором щелочи
310 INPUT «При взаимодействии с раствором щелочи ощущается запах аммиака?»;D$
320 IF D$="да" THEN X$="сульфат аммония" ELSE X$="суперфосфат"
330 RETURN
400 REM розовые кристаллы
410 NPUT «Розовые кристаллы?»;E$
420 IF E$="да" THEN X$="сильвинит" ELSE X$="калийная соль"
430 RETURN
```

Более удачный вариант решения той же задачи показан на рис. 102. Учебная экспертная дракон-система работает следующим образом. После запуска системы рабочая точка процесса начинает двигаться от иконы “заголовок” к иконе “конец”. По ходу ее движения иконы и соединительные линии вспыхивают и горят на экране более ярким светом, выделяя пройденную часть пути. Когда процесс дошел до иконы-вопрос “При взаимодействии с H_2SO_4 выделяется бурый газ?”, данная икона начинает мерцать, привлекая к себе внимание и требуя ответа. Реагируя на это событие, ученик подводит курсор к нужному ответу (да или нет) и щелкает клавишей мыши. Икона перестает мерцать и (при ответе “да”) загорается путь, ведущий к иконе-вопрос “При взаимодействии со щелочью ощущается запах аммиака?”, которая начинает мерцать. Далее события повторяются, пока на экране не загорится искомое название удобрения.

Таким образом, дракон-система выполняет те же самые функции, что и система на БЕЙСИКЕ. Вместе с тем она обладает рядом преимуществ.

! Программа на БЕЙСИКЕ содержит 790 символов (не считая пробелов), из которых только 488 символов (60%) описывают задачу на естественном языке, а остальные 302 (40%) представляют собой набор иероглифов — загадочный текст на птичьем языке программирования, который все нормальные люди (не программисты) воспринимают с неудовольствием. В дракон-схеме на рис. 102 повод для недовольства исчезает, “птичья абракадабра” полностью отсутствует, а все необходимые функции тем не менее выполняются. Становится очевидным, что “программные иероглифы” являются паразитными, избыточными и даже вредными, поскольку они работают не на пользователя (которому они не нужны), а сами на себя. В данном случае сущность эргономизации состоит в полном отказе от использования “птичьей абракадабры”.

! Другое преимущество заключается в системном подходе к проблеме симультанизации. С точки зрения процесса познания, проблема распознавания удобрения состоит из трех частей:

1) постановка задачи;
2) описание действий с исследуемым веществом и реактивами;
3) логический анализ результатов опыта.

Недостаток программы на БЕЙСИКЕ в том, что она освещает лишь последнюю часть и “прячет от читателя” две первых. Дракон-система свободна от этого дефекта: в первой ветке даны постановка задачи (икона “комментарий”) и описание последовательности ручных манипуляций (четыре иконы “действие”), во второй ветке демонстрируется логический анализ и получение ответа.

! Исключительно важно, что все три части проблемы предъявляются зрителю в одном визуальном поле. Благодаря этому обеспечивается симультанизация восприятия и улучшение работы ума.

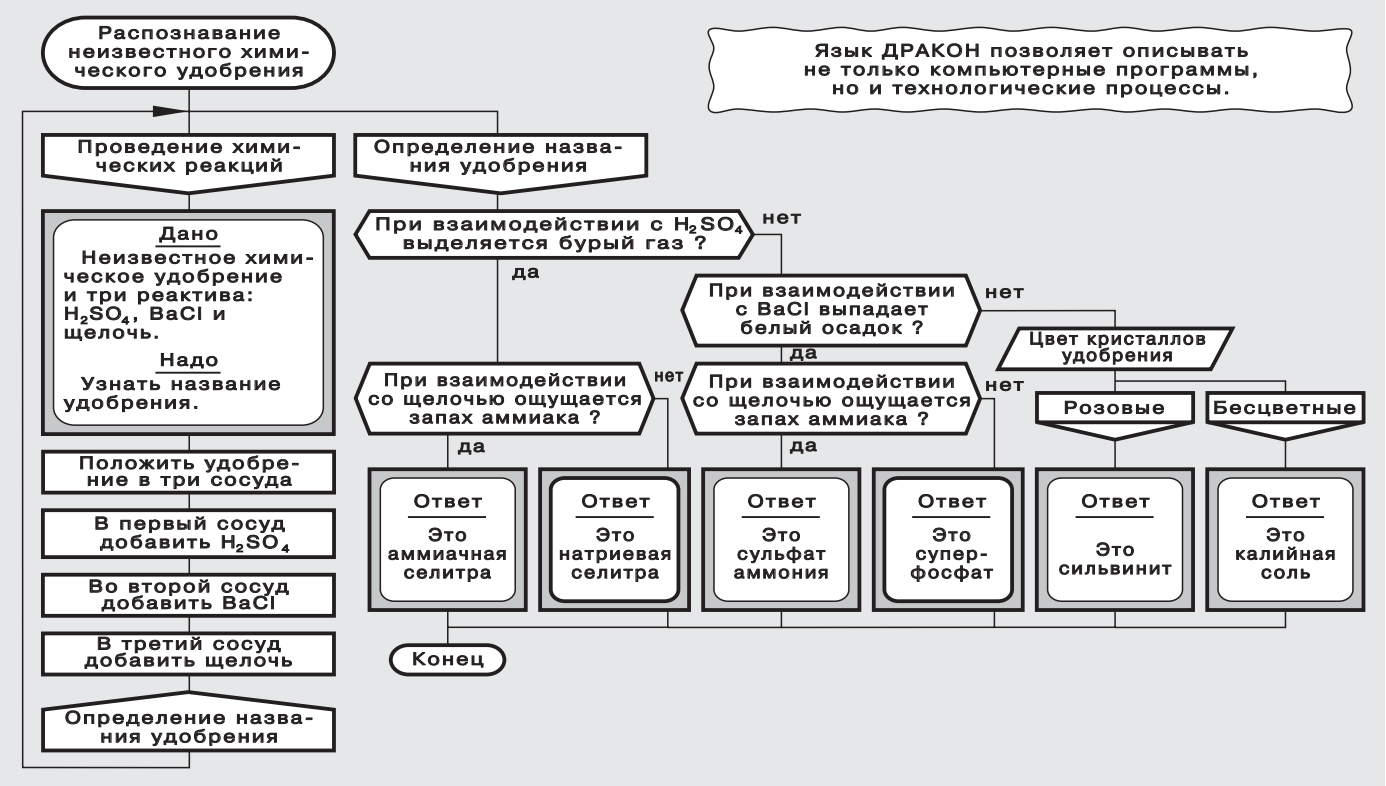


Рис. 102. **Технология: распознавание неизвестного химического удобрения**

- Программа на БЕЙСИКЕ — пример плохой (неэргономичной) экспертной системы, которая общается с пользователем через узкую “замочную скважину”, сквозь которую виден один-единственный вопрос и больше ничего. Тем самым бейсик-система не дает возможности человеку одновременно охватить единым взором и логические детали, и общую картину логического вывода, фактически превращая пользователя в какого-то пасынка или даже дурачка, от которого скрывают все самое интересное.
- Хорошо известно, что пользователю далеко не безразлично, каким образом экспертная система приходит к своему решению. Идя навстречу таким пожеланиям, современные экспертные системы помогают пользователю не только принимать решения, но и позволяют выявить мотивы их принятия через систему объяснения. Более того, специалисты считают, что от наличия или отсутствия в системе объяснительной функции зависит право этой системы называться экспертной. Исходя из этого, многие системы разрешают пользователю задавать вопрос “Почему?”, после чего “раскрывают карты” и в словесной форме показывают пользователю ход своих рассуждений. Это хорошо, но мало. В ряде приложений идеальным решением можно назвать такое, при котором система предъявляет пользователю всю панораму возможных логических выводов, на которой выделяется (яркостью или цветом) маршрут цепочки конкретных умозаключений, ведущих к выбранному ответу. ДРАКОН-система реализует именно этот идеальный подход (рис. 102).

ВИЗУАЛИЗАЦИЯ ОПИСАНИЯ ТЕХНОЛОГИЧЕСКИХ ПРОЦЕССОВ

На рис. 103 представлено упрощенное описание технологического процесса изготовления фруктовых консервов из косточковых плодов (автор технологии Е. Свешникова). Реальный технологический процесс может быть очень сложным. Обычно его описывают как головной процесс, содержащий большое число вставок. В качестве примера в головном процессе на рис. 103 показана вставка “Изготовление сиропа и маринада”, раскрытая на рис. 104.

В реальном техпроцессе часто встречаются одновременно протекающие процессы. Для их изображения на языке ДРАКОН применяется не только икона “параллельный процесс”, но и другие средства (учитывающие специфику технологических процессов), которые в данной книге не рассматриваются.

Дракон-схемы технологических процессов могут найти применение в следующих случаях:

- создание наглядных плакатов, дающих целостное представление о процессе во всей его многосложности и используемых в качестве демонстрационных материалов; при этом в иконе “комментарий” могут помещаться чертежи, фотографии, схемы установок, станков, сетей трубопроводов и другого оборудования;
- выпуск технологической документации;
- проектирование и моделирование технологических процессов;
- создание визуальной базы данных о техпроцессах;

- создание экспертных систем для проектирования технологических процессов, а также тренажеров для эксплуатационников;
- изготовление альбомов и каталогов технологических процессов для обучения или рекламы; можно рекомендовать формат бумажной страницы альбома А3, имея в виду, что оригинал-макеты альбомов готовятся на лазерном принтере формата А3.

ЧТО ТАКОЕ МЕТОДОЛОГИЯ?

Джеймс Мартин подчеркивает необходимость различать два понятия: методика *(technique)* и методология *(methodology)*.

Методика — это способ выполнения одной операции. Например, правила составления схем потоков данных — это методика.

Методология разработки систем охватывает *набор задач* (операций), которые необходимо решить в процессе создания системы. Существует много задач, при решении которых применяется много методик. Выход одной задачи часто является входом в другую. Применяя компьютеры при разработке систем, желательно полностью автоматизировать те задачи, которые поддаются автоматизации, а остальные выполнять автоматизированным способом, когда человек работает вместе с компьютером. Поток знаний, передаваемых от одной задачи к следующей, должен находиться внутри компьютеризованных инструментов всегда, когда это возможно. Методология *RAD* использует компьютеризованные средства и ручные методы, разумно связанные между собой, чтобы достичь две цели: большую скорость и высокое качество разработки. *Методология* определяет, в чем заключается каждая задача, как ее успешно выполнить, какие неприятности и опасности возможны в процессе работы и как их избежать.

ВИЗУАЛИЗАЦИЯ МЕТОДОЛОГИЙ

Иногда высказывают мнение, что язык ДРАКОН хорошо описывает простые задачи и непригоден для изображения сложных проблем. Это неверно. ДРАКОН специально сконструирован, чтобы облегчить формализацию широкого спектра задач, включая самые сложные. Более того, чем сложнее проблема, тем больше выигрыш от использования языка ДРАКОН.

Чтобы убедиться в этом, рассмотрим методологию проектирования атомного реактора. Ясно, что это грандиозная, “запредельная” по сложности проблема. Целостный взгляд на методологию представлен

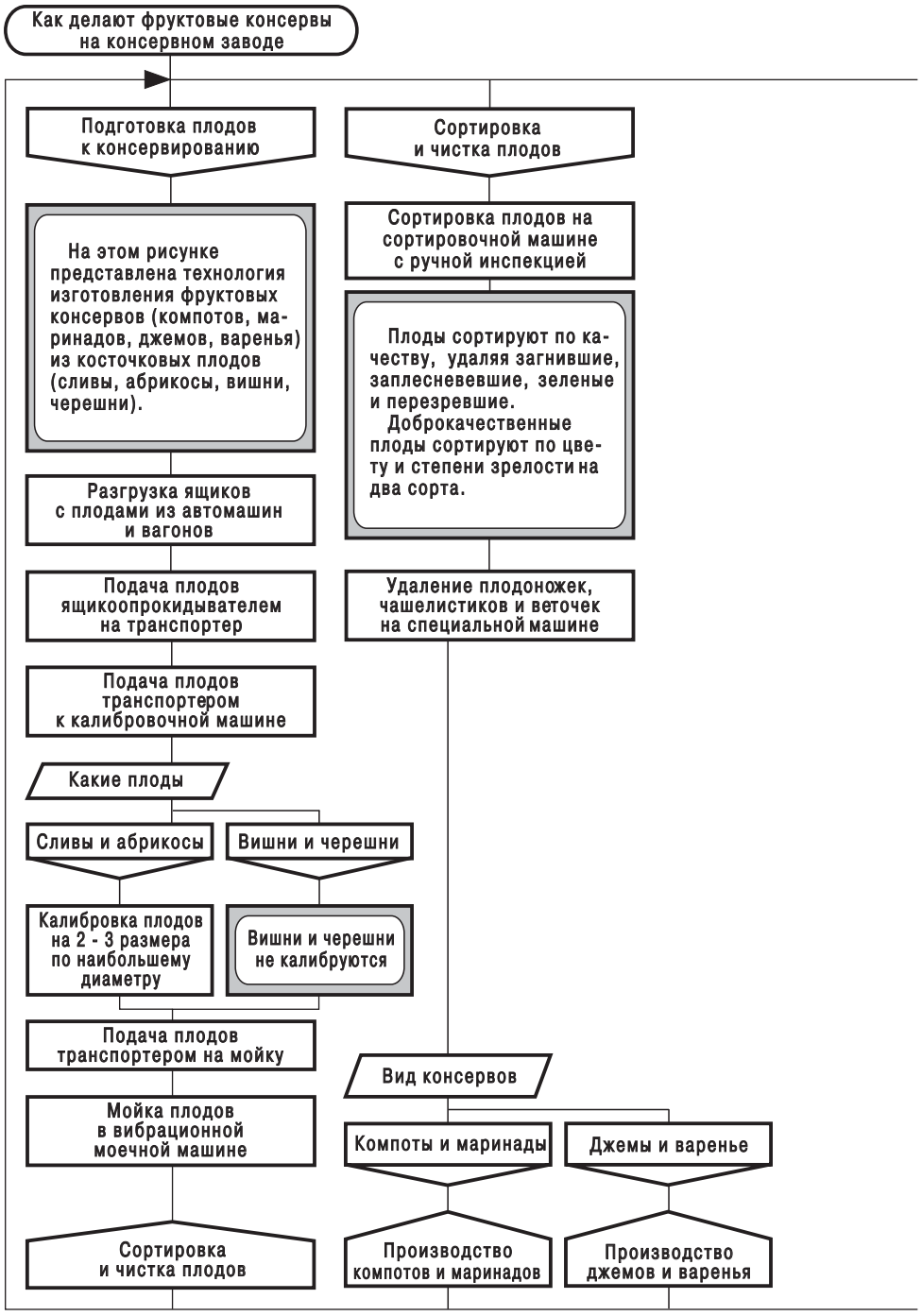
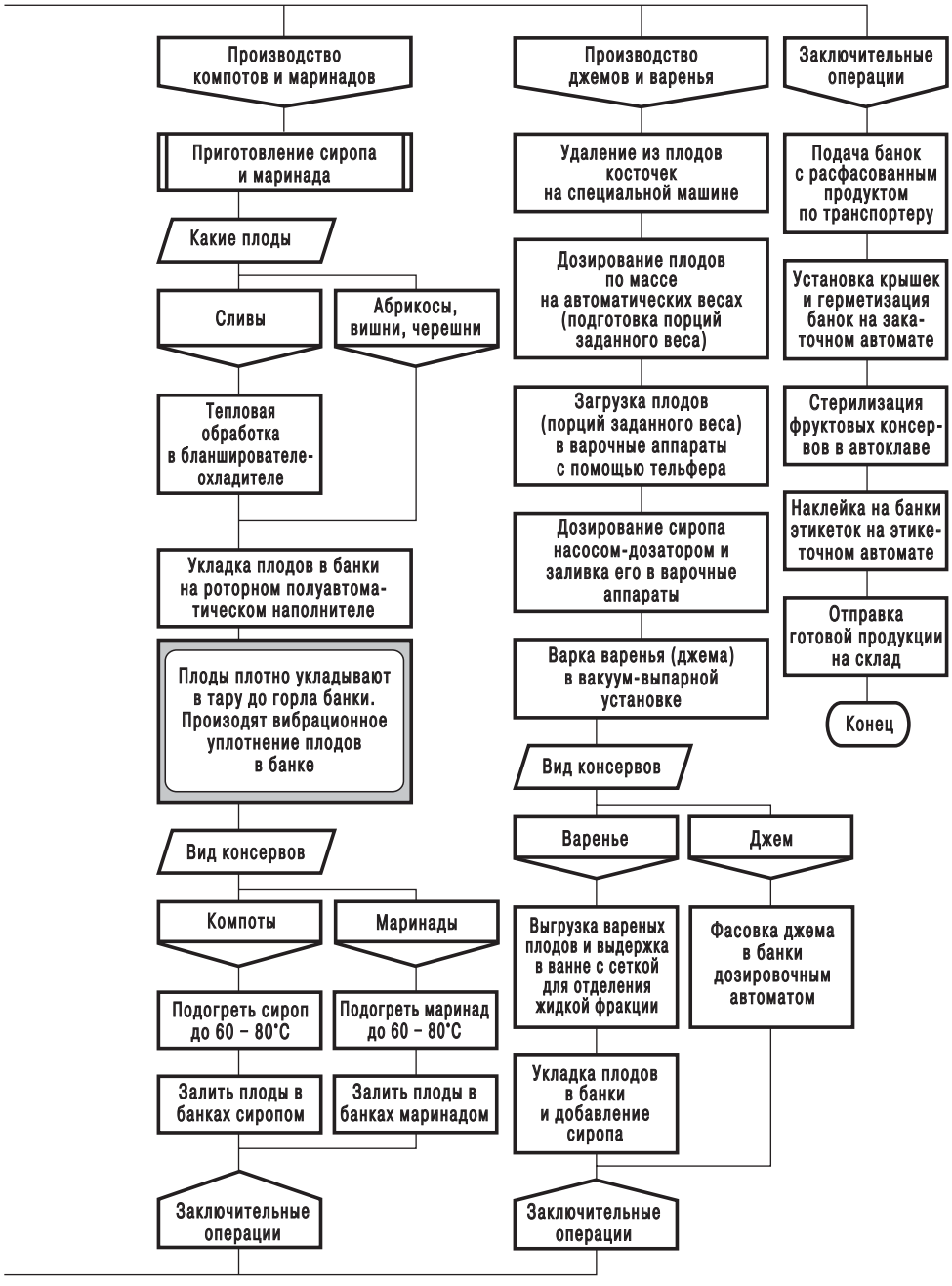

Как делают фруктовые консервы на консервном заводе
Подготовка плодов к консервированию
Сортировка и чистка плодов
На этом рисунке представлена технология изготовления фруктовых консервов (компотов, маринадов, джемов, варенья) из косточковых плодов (сливы, абрикосы, вишни, черешни).
Сортировка плодов на сортировочной машине с ручной инспекцией
Плоды сортируют по качеству, удаляя загнившие, заплесневевшие, зеленые и перезревшие.
Доброкачественные плоды сортируют по цвету и степени зрелости на два сорта.
Разгрузка ящиков с плодами из автомашин и вагонов
Подача плодов ящикоопрокидывателем на транспортер
Удаление плодоножек, чашелистиков и веточек на специальной машине
Подача плодов транспортером к калибровочной машине
Какие плоды
Сливы и абрикосы
Вишни и черешни
Калибровка плодов на 2 - 3 размера по наибольшему диаметру
Вишни и черешни не калибруются
Подача плодов транспортером на мойку
Вид консервов
Мойка плодов в вибрационной моечной машине
Компоты и маринады
Джемы и варенье
Сортировка и чистка плодов
Производство компотов и маринадов
Производство джемов и варенья

Рис. 103. **Технология изготовления фруктовых консервов из косточковых плодов**



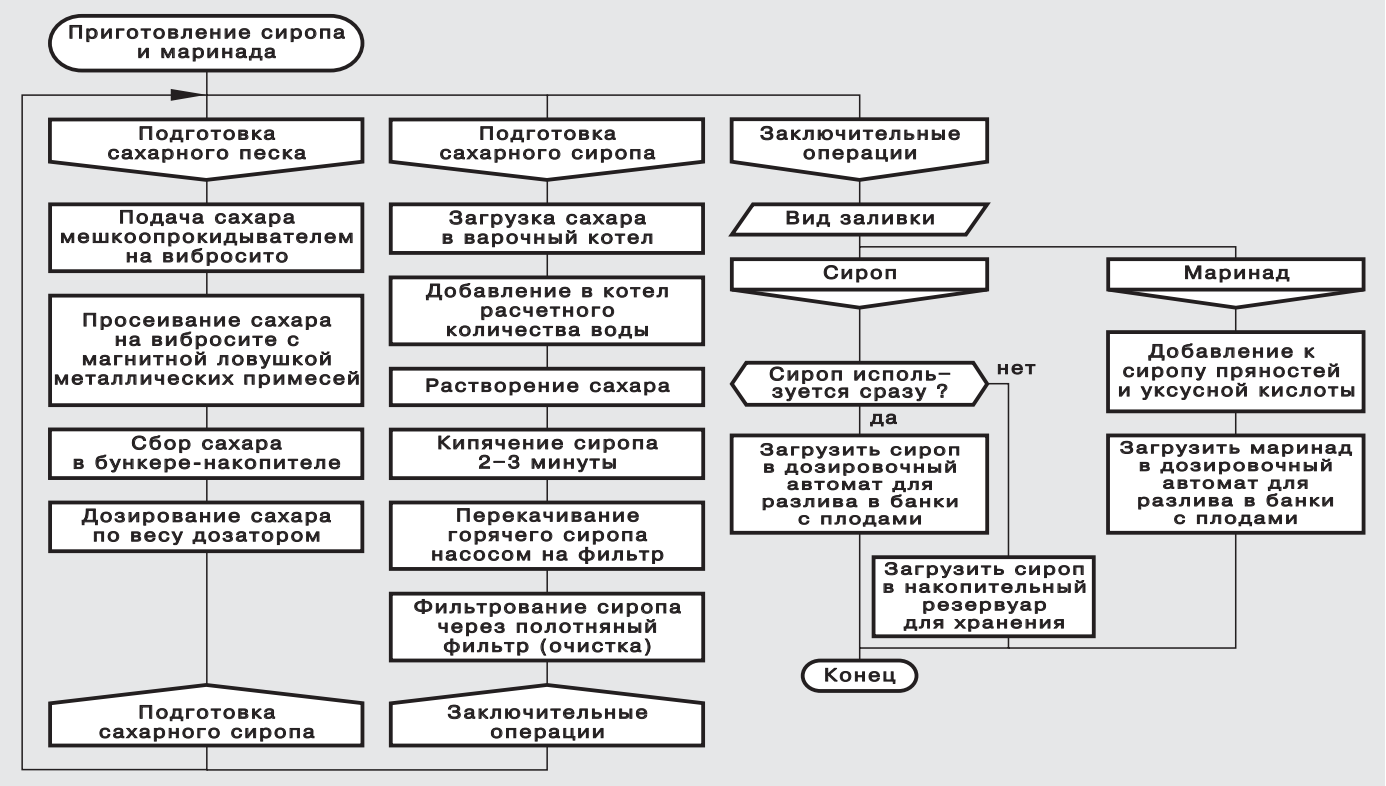


Рис. 104. **Технология изготовления сиропа и маринада**

на рис. 105. Дракон-схема на рис. 105 содержит большое число вставок, для обозначения которых в данном случае целесообразно ввести термин *алгоритм-концепция*. Например, во второй и четвертой ветке на рис. 105 имеются иконы-вставки “Расчет стационарных параметров первого контура атомного реактора” и “Расчет реактивностных аварий атомного реактора”. Соответствующие им алгоритмы-концепции показаны на рис. 106 и 107[1].

Рис. 105—107 убедительно демонстрируют, что любую, сколь угодно сложную методологию можно изобразить с помощью простого и единообразного приема, который можно охарактеризовать как *наглядную декомпозицию*. Верхний уровень иерархии, показанный на рис. 105, можно рассматривать как вершину гигантской пирамиды, откуда открывается взгляд на проблему с высоты птичьего полета. Там же перечисляются все алгоритмы-концепции второго уровня, которые в нашей воображаемой пирамиде расположены на один шаг “ближе к земле”. Рассматривая алгоритм второго уровня (изображенные на рис. 106 и 107), легко заметить, что в них указываются алгоритмы-концепции третьего уровня, которые находятся еще ближе к земле, т. е. дают более детальное знакомство с проблемой. Постепенно спускаясь с вершины пирамиды к ее основанию, мы наблюдаем последовательную декомпозицию сложной проблемы на все более мелкие и подробные детали, которые в конечном итоге (когда мы спустимся “на уровень земли”) дадут исчерпывающее и полное описание методологии как императивной проблемы. При необходимости ее можно дополнить соответствующими декларативными описаниями.

Важным достоинством является тот факт, что язык не зависит от уровня иерархии — он один и тот же и на самом верху и у основания пирамиды. Благодаря этому достигается резкое упрощение описания задач любой сложности; в итоге “уму непостижимая” проблема превращается в относительно простую, ясную и наглядную.

Насколько известно автору, до сих пор практически отсутствовали *эффективные эргономичные* изобразительные средства, позволяющие одновременно решать две задачи: формализацию и визуализацию методологий. По этой причине целостный взгляд на методологию, как на детерминированный многоступенчатый процесс, имеющий начало и конец, по сути дела был недоступен широкому кругу специалистов и учащихся, оставаясь достоянием узкой группы суперспециалистов, которые “все держат в голове”. Из-за этого остальным участникам сложного проекта вынужденно отводилась роль винтиков творческого организма, которые должны знать свой “шесток”, но которым “не положено” иметь целостное панорамное видение процесса во всей его многосложности. Язык ДРАКОН позволяет сделать важный шаг к устра-

[1] Авторами дракон-схем на рис. 105—107 и изображенной на них методологии являются В. Болнов, Д. Шипов и В. Кууль.

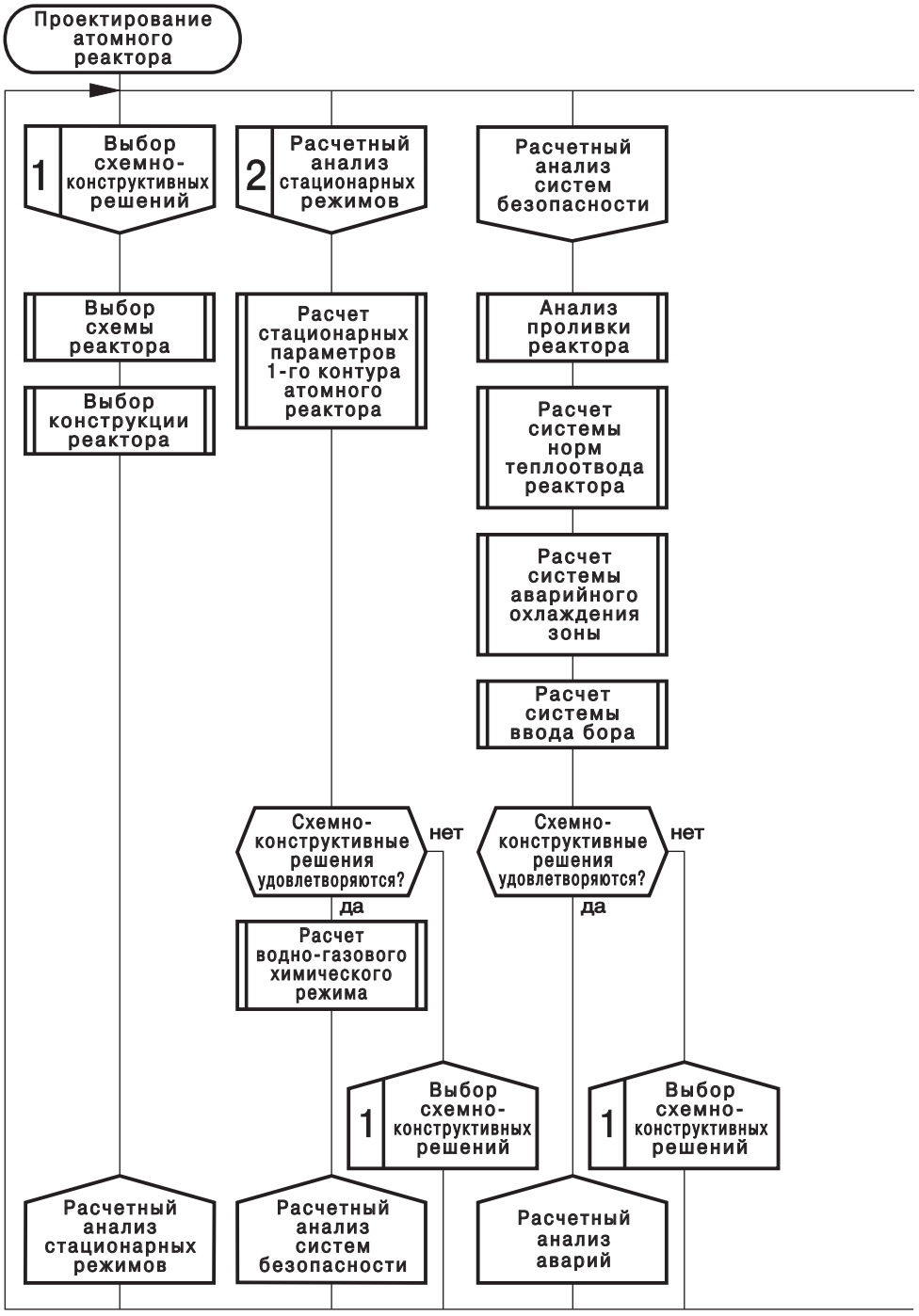
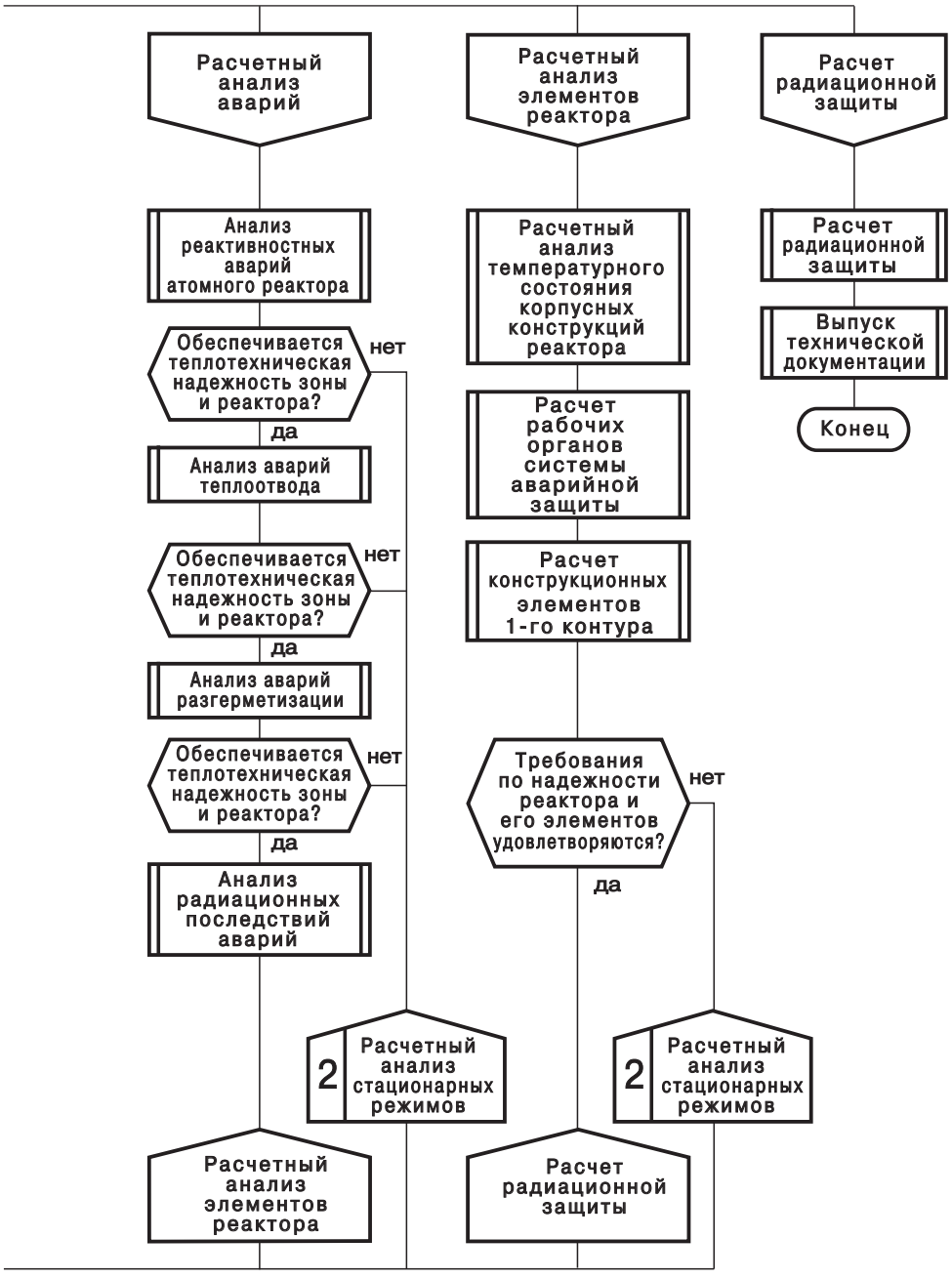

Проектирование атомного реактора
1
Выбор схемно-конструктивных решений
2
Расчетный анализ стационарных режимов
Расчетный анализ систем безопасности
Выбор схемы реактора
Выбор конструкции реактора
Расчет стационарных параметров 1-го контура атомного реактора
Анализ проливки реактора
Расчет системы норм теплоотвода реактора
Расчет системы аварийного охлаждения зоны
Расчет системы ввода бора
Схемно-конструктивные решения удовлетворяются?
нет
да
Схемно-конструктивные решения удовлетворяются?
нет
да
Расчет водно-газового химического режима
1
Выбор схемно-конструктивных решений
1
Выбор схемно-конструктивных решений
Расчетный анализ стационарных режимов
Расчетный анализ систем безопасности
Расчетный анализ аварий

Рис. 105. **Методология “Проектирование атомного реактора”**



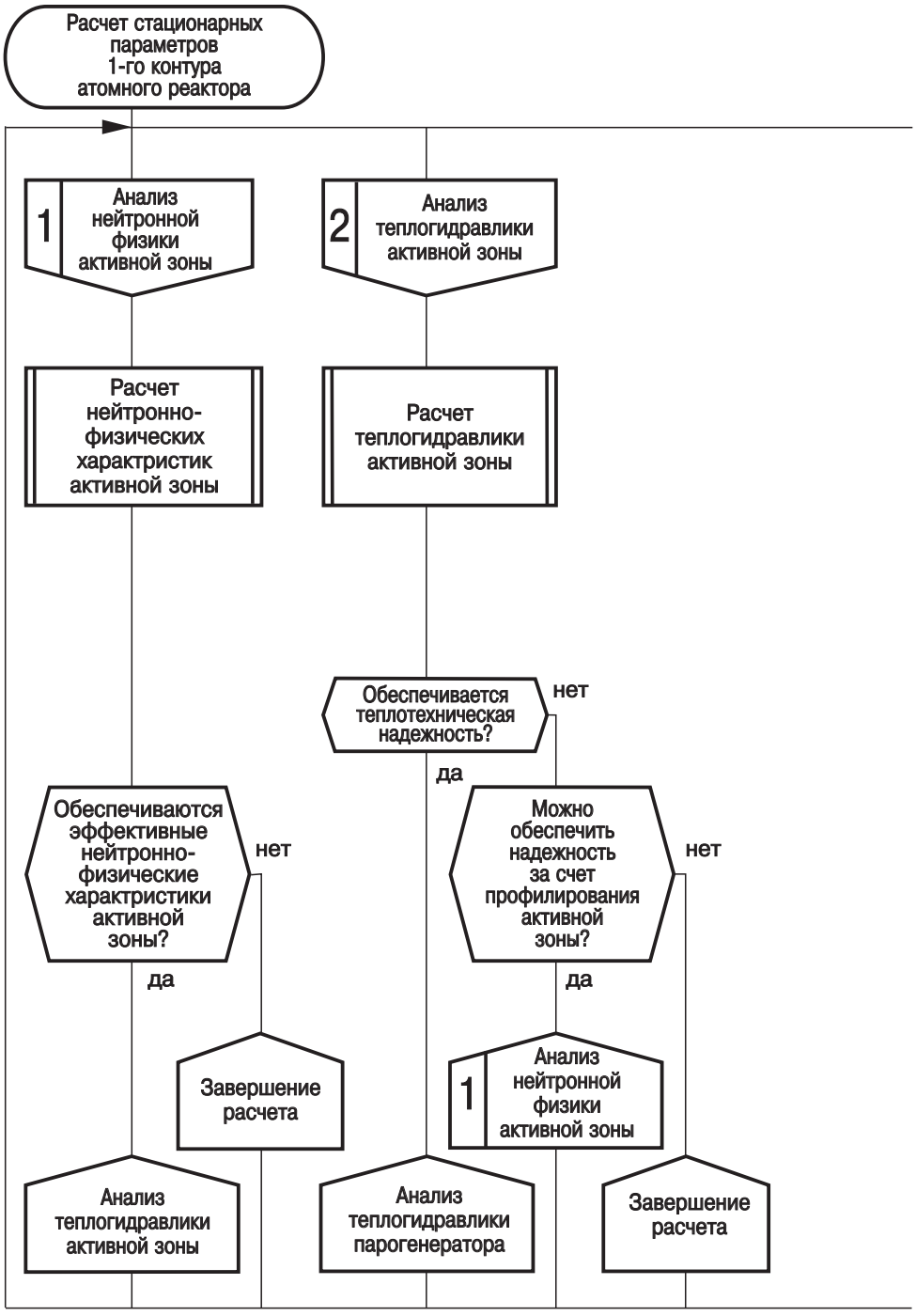
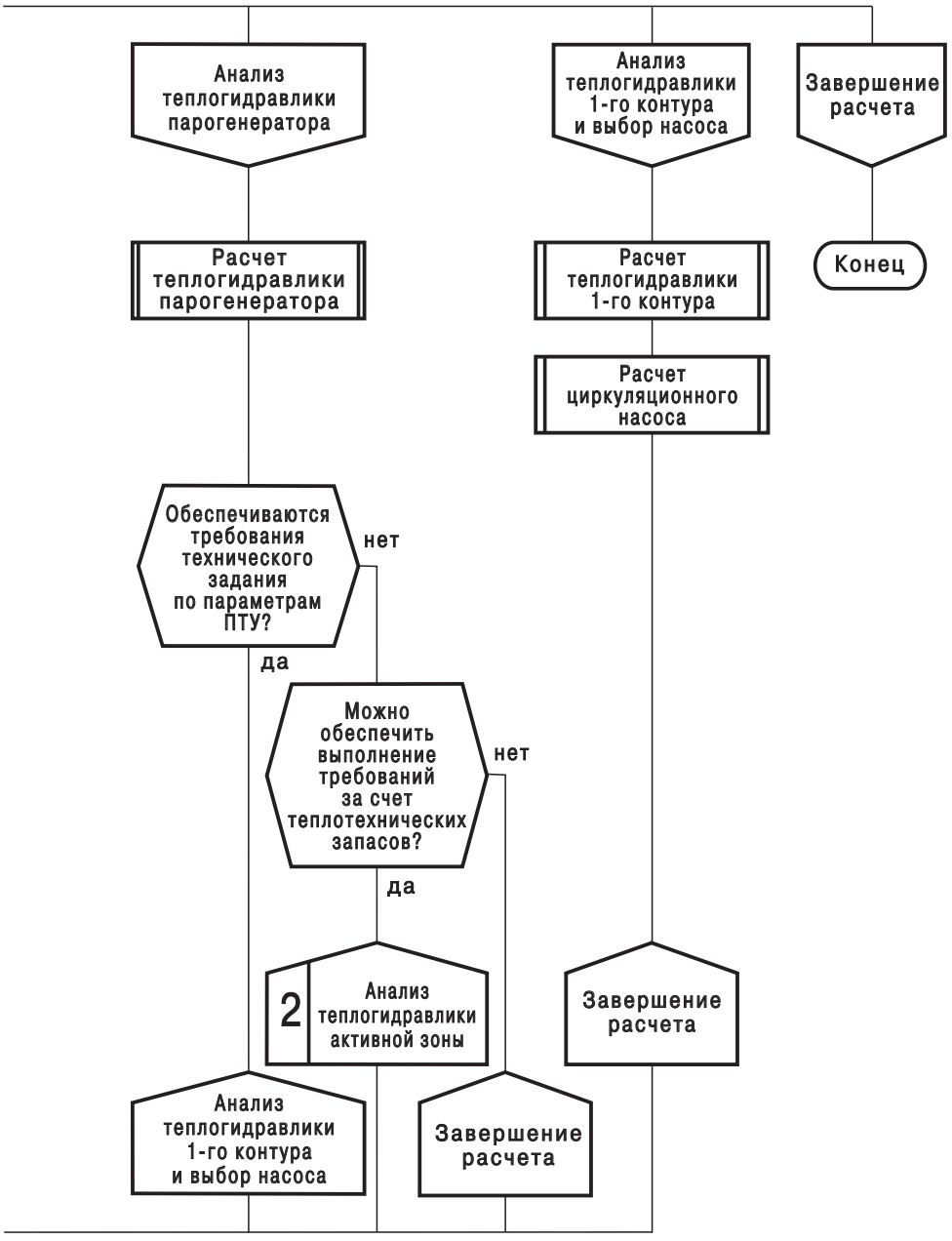

Расчет стационарных параметров 1-го контура атомного реактора
1
Анализ нейтронной физики активной зоны
2
Анализ теплогидравлики активной зоны
Расчет нейтронно-физических характристик активной зоны
Расчет теплогидравлики активной зоны
Обеспечивается теплотехническая надежность?
нет
да
Обеспечиваются эффективные нейтронно-физические характристики активной зоны?
нет
да
Можно обеспечить надежность за счет профилирования активной зоны?
нет
да
Завершение расчета
1
Анализ нейтронной физики активной зоны
Анализ теплогидравлики активной зоны
Анализ теплогидравлики парогенератора
Завершение расчета

Рис. 106. **Алгоритм-концепция “Расчет стационарных параметров 1-го контура атомного реактора”.** ***Данный алгоритм является вставкой во вторую ветку алгоритма на рис. 105***



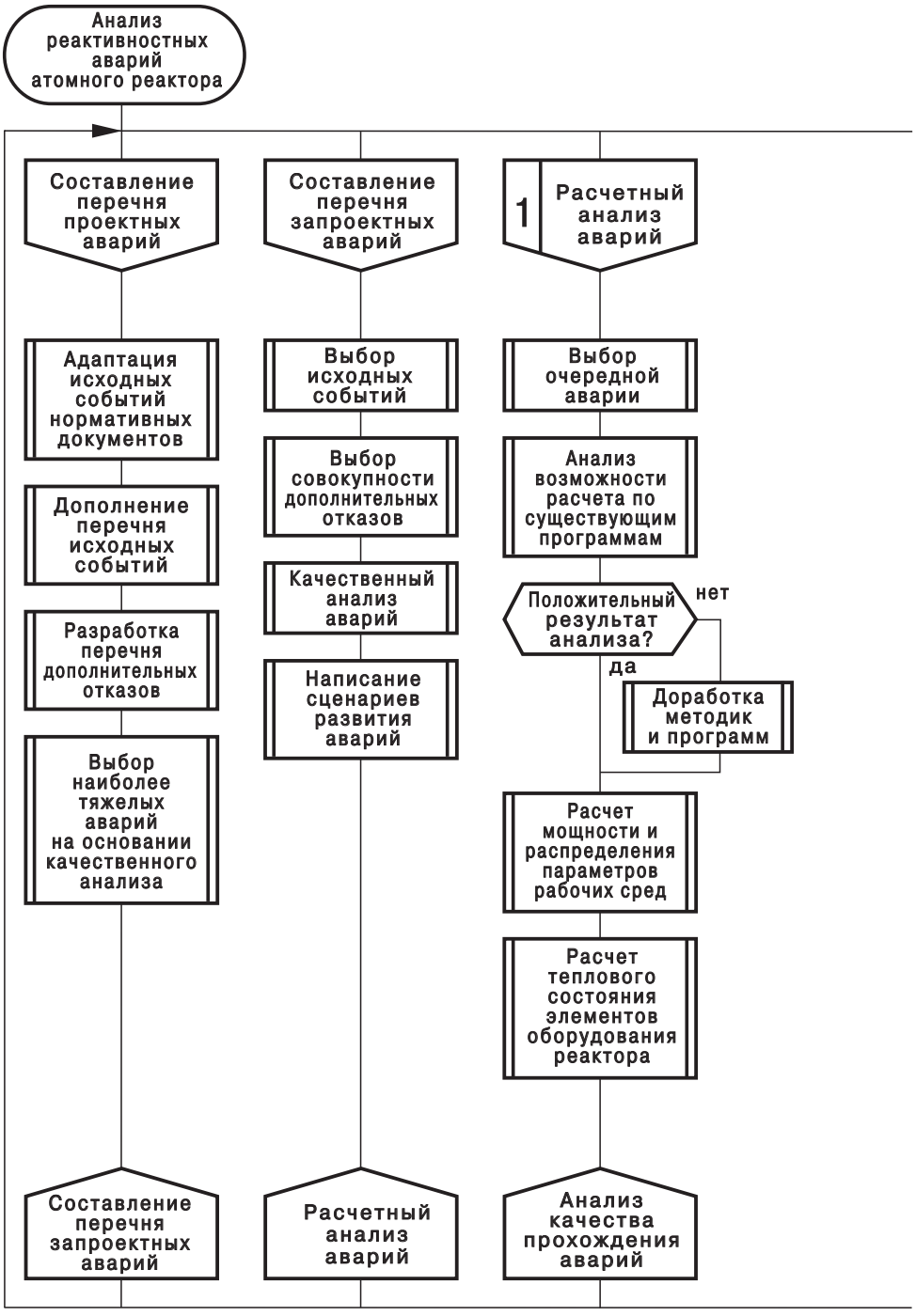
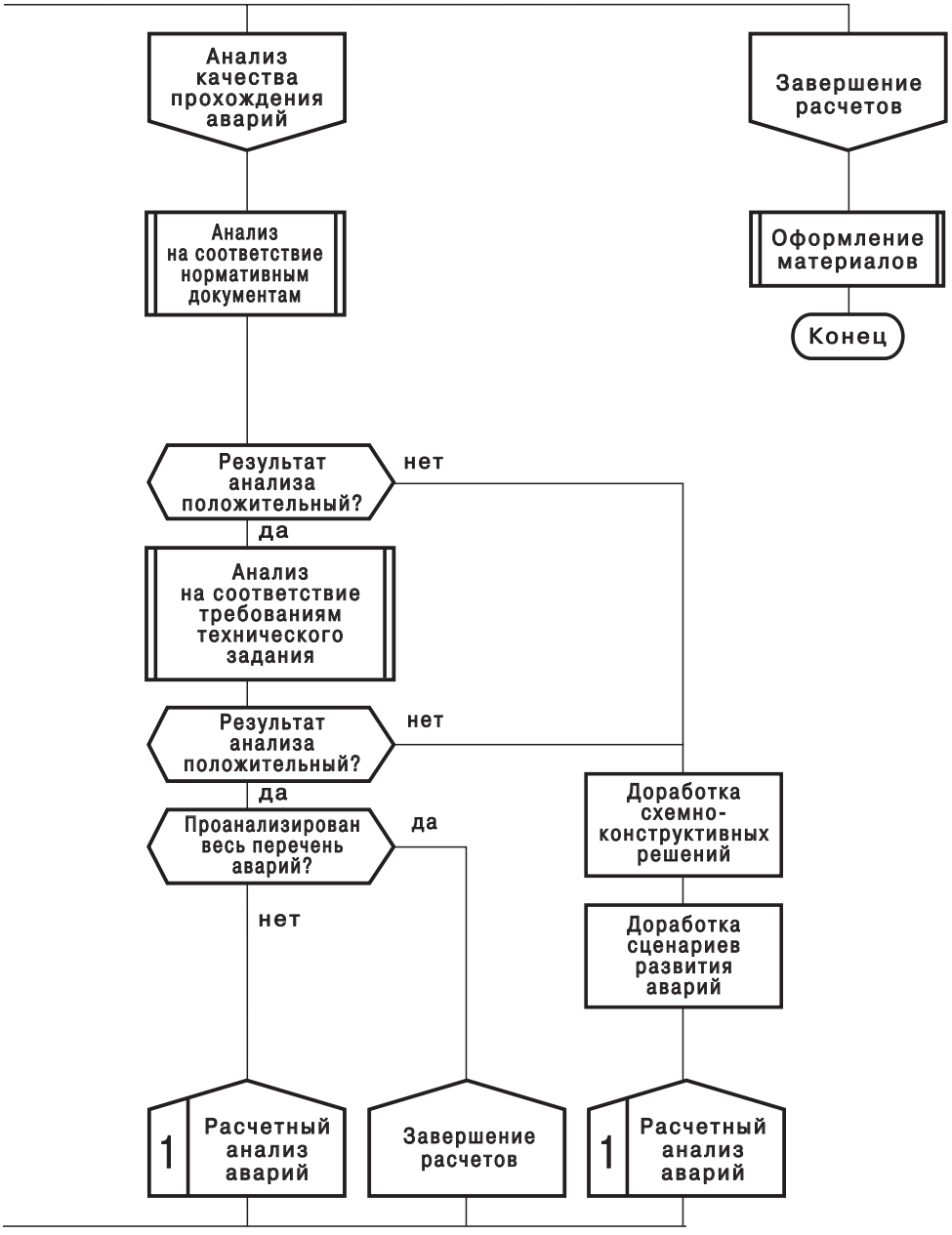

Анализ реактивностных аварий атомного реактора
Составление перечня проектных аварий
Составление перечня запроектных аварий
1
Расчетный анализ аварий
Адаптация исходных событий нормативных документов
Дополнение перечня исходных событий
Разработка перечня дополнительных отказов
Выбор наиболее тяжелых аварий на основании качественного анализа
Выбор исходных событий
Выбор совокупности дополнительных отказов
Качественный анализ аварий
Написание сценариев развития аварий
Выбор очередной аварии
Анализ возможности расчета по существующим программам
Положительный результат анализа?
нет
да
Доработка методик и программ
Расчет мощности и распределения параметров рабочих сред
Расчет теплового состояния элементов оборудования реактора
Составление перечня запроектных аварий
Расчетный анализ аварий
Анализ качества прохождения аварий

Рис. 107. **Алгоритм-концепция “Анализ реактивностных аварий атомного реактора”.** ***Данный алгоритм является вставкой в четвертую ветку алгоритма на рис. 105***



нению этого недостатка, более эффективно организовать совместную работу участников сложного проекта и более разумно использовать интеллектуальные ресурсы их коллективного мозга.

СИСТЕМА “ЧЕЛОВЕК—МАШИНА”

В настоящем параграфе[1] обосновывается целесообразность использования языка ДРАКОН для формализованного описания систем “человек—машина”. Рассмотрим для примера разработку системы “экипаж—вертолет”. Описание этой системы необходимо иметь при проектировании вертолета на этапе эскизного проекта для достижения трех целей.

Во-первых, для проведения эргономического анализа системы “человек—машина” при выборе вариантов:

- распределения функций между человеком и машиной;
- распределения функций между членами экипажа вертолета;
- состава оборудования;
- численности экипажа.

Во-вторых, для обеспечения специалистов информацией о работе различных подсистем системы “человек—машина” в штатных и аварийных условиях работы. В-третьих, для обучения операторов работе с системой.

В настоящее время при проектировании летательных аппаратов описание алгоритма работы системы “человек—машина” делается в текстовой форме в виде документа, который называется “Руководство по летной эксплуатации”, форма которого определена государственным стандартом. Этот документ выпускается слишком поздно — после окончания рабочего проектирования и обычно бывает приурочен к началу летных испытаний. Он представляет собой констатацию сложившихся в ходе проектирования алгоритмов работы системы “человек—машина” и не предназначен для достижения первых двух целей. Кроме того, текстовая форма документа не отвечает принципам симультанного восприятия, что затрудняет его использование для обучения персонала.

В связи с этим была предпринята попытка применить для решения задачи язык ДРАКОН. В частности, была разработана детальная дракон-схема, описывающая работу системы “экипаж—вертолет” при возникновении аварийной ситуации в воздухе — пожаре правого двигателя. Анализ полученных результатов позволяет сделать вывод о полезности такого описания: оно сочетается с требованиями, предъявляемыми к математическому моделированию системы “человек—машина”, определению временной структуры деятельности членов экипажа, использованию микроструктурного анализа деятельности и т. д. Можно утверждать, что создание библиотеки дракон-схем проектируемой системы “человек—машина” — необходимый этап детального эргономического анализа при создании пилотируемых летательных аппаратов.

---

[1] Автором данного параграфа является главный эргономист Московского вертолетного завода, член Международного эргономического общества А. Макаркин.

ВИЗУАЛИЗАЦИЯ БИОЛОГИЧЕСКИХ АЛГОРИТМОВ

Чем глубже человеческий разум проникает в тайны живой материи, тем яснее становится, что живые существа во многих случаях ведут себя, как информационные биомашины, перерабатывающие информацию с помощью биоалгоритмов. Опыт показывает, что биологические алгоритмы очень похожи на самые обычные алгоритмы, с которыми мы постоянно сталкиваемся в технике. А раз так, язык ДРАКОН может стать удобным средством для выражения и накопления знаний об информационных процессах, протекающих в живых организмах. Для иллюстрации приведем цитату из известной биологической книги, описывающую один из таких алгоритмов.

> «Рассмотрим изменения, происходящие в организме жабы в сезон размножения. Глаза жабы воспринимают свет и передают эту информацию в мозг, который определяет, что продолжительность светового дня увеличивается. Гипоталамус направляет в гипофиз соответствующие рилизинг-факторы, и гипофиз начинает выделять в кровь различные гормоны, включая фолликулостимулирующий и лютеинизирующий. Когда семенники и яичники “обнаруживают” их присутствие в крови, они начинают увеличиваться в размерах, продуцировать гаметы, а также выделять собственные гормоны и среди них — половые: тестостерон и эстроген. Реагируя на половые гормоны, мозг посылает нервные импульсы к мышцам — животное начинает поиск места для размножения и брачного партнера. Так, благодаря сложному взаимодействию органов чувств, нервов, мозга, мышц и эндокринных желез животное адекватно реагирует на смену сезона — наступление весны» [1].

Описание этого алгоритма на языке ДРАКОН показано на рис. 108 (чтобы не утомлять читателя громоздкими биологическими терминами, автор несколько упростил текст и снабдил его некоторой долей юмора).

Вместе с тем следует специально подчеркнуть, что реальные биологические алгоритмы исключительно сложны. Традиционная для биологической литературы текстовая форма представления алгоритмических знаний вносит неоправданные трудности для читателей и является устаревшей. По мнению автора, язык ДРАКОН может оказать существенную помощь биологам. Создание бумажных альбомов и компьютерных библиотек биологических дракон-схем даст возможность улучшить

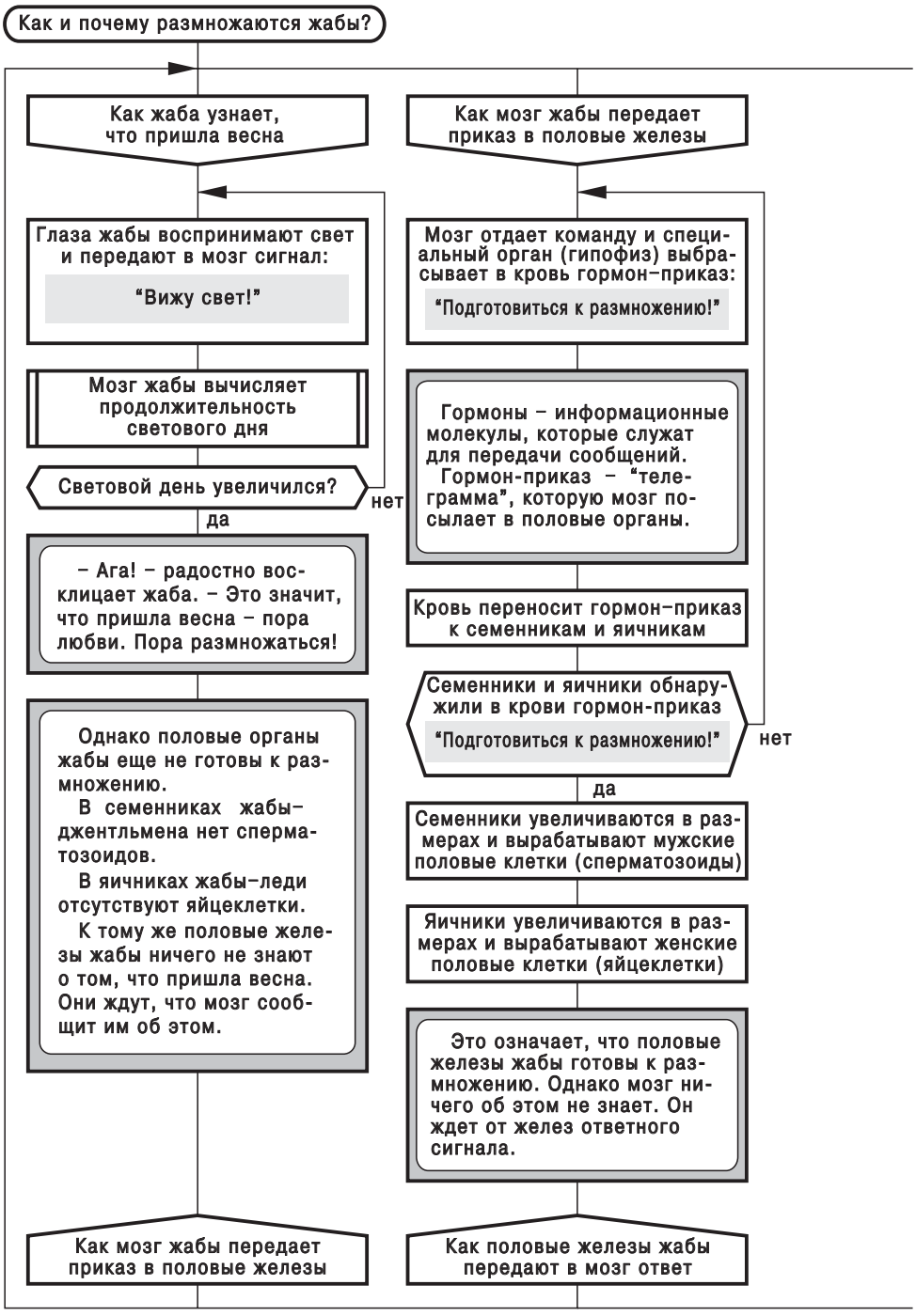
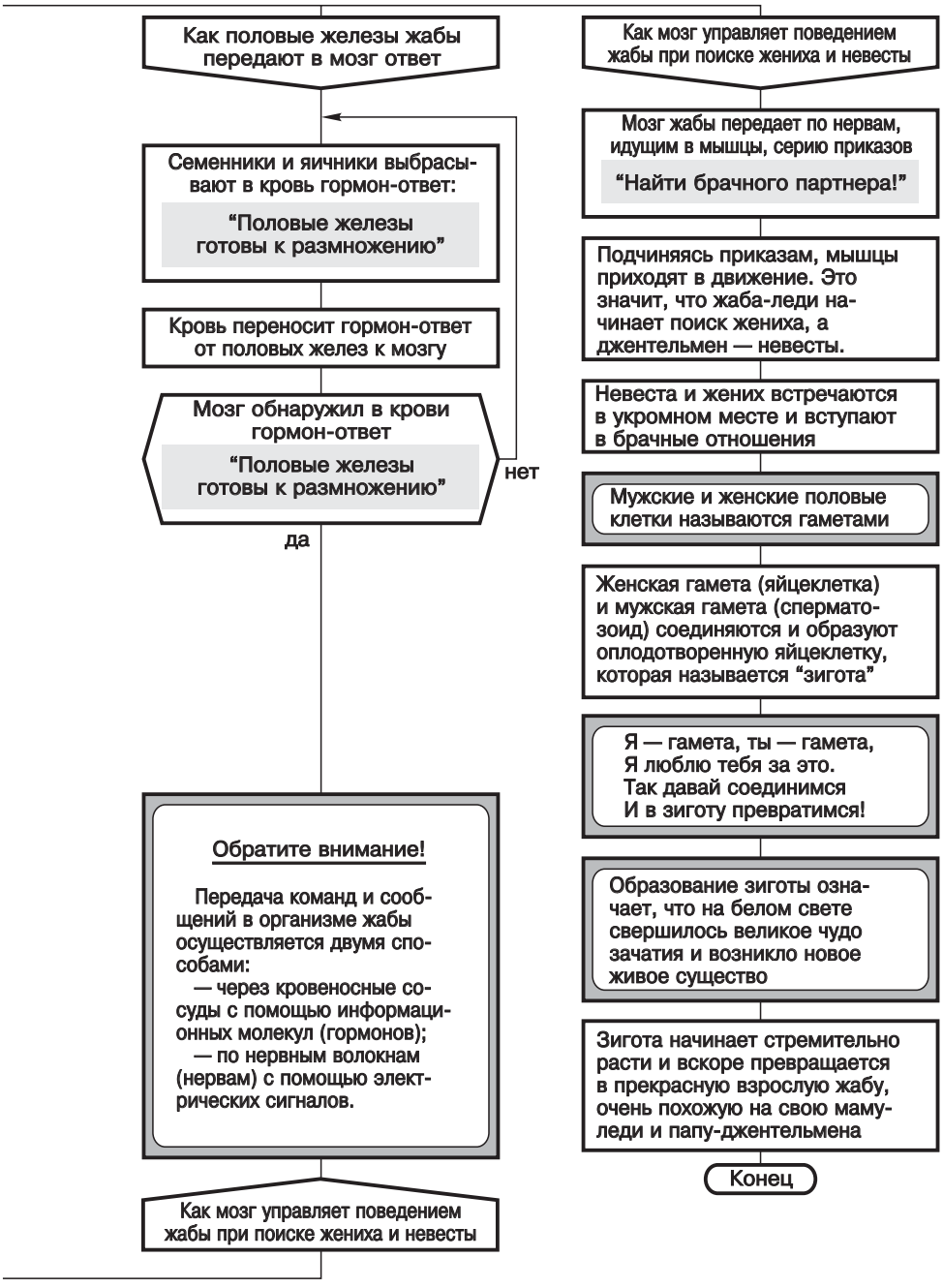

Как и почему размножаются жабы?
Как жаба узнает, что пришла весна
Как мозг жабы передает приказ в половые железы
Глаза жабы воспринимают свет и передают в мозг сигнал:
“Вижу свет!”
Мозг отдает команду и специальный орган (гипофиз) выбрасывает в кровь гормон-приказ:
“Подготовиться к размножению!”
Мозг жабы вычисляет продолжительность светового дня
Гормоны – информационные молекулы, которые служат для передачи сообщений.
Гормон-приказ – “телеграмма”, которую мозг посылает в половые органы.
Световой день увеличился?
нет
да
– Ага! – радостно восклицает жаба. – Это значит, что пришла весна – пора любви. Пора размножаться!
Кровь переносит гормон-приказ к семенникам и яичникам
Семенники и яичники обнаружили в крови гормон-приказ
“Подготовиться к размножению!”
нет
да
Однако половые органы жабы еще не готовы к размножению.
В семенниках жабы-джентльмена нет сперматозоидов.
В яичниках жабы-леди отсутствуют яйцеклетки.
К тому же половые железы жабы ничего не знают о том, что пришла весна. Они ждут, что мозг сообщит им об этом.
Семенники увеличиваются в размерах и вырабатывают мужские половые клетки (сперматозоиды)
Яичники увеличиваются в размерах и вырабатывают женские половые клетки (яйцеклетки)
Это означает, что половые железы жабы готовы к размножению. Однако мозг ничего об этом не знает. Он ждет от желез ответного сигнала.
Как мозг жабы передает приказ в половые железы
Как половые железы жабы передают в мозг ответ

Рис. 108. **Как размножаются жабы? Использование языка ДРАКОН для описания биологических процессов**



форму представления биологических знаний, сделать ее более строгой и наглядной, позволит выявить и устранить алгоритмические пробелы в знаниях, поможет укреплению позиций информационной биологии и облегчит дальнейшие исследования таинственного механизма функционирования живых организмов.

ВИЗУАЛИЗАЦИЯ МЕДИЦИНСКИХ АЛГОРИТМОВ

Медики редко произносят слово “алгоритм”. А жаль! — ведь алгоритмы составляют значительную часть медицинских знаний. На рис. 109 представлен знакомый почти каждому пример — измерение кровяного давления (которое медики обозначают торжественным словом “сфигмоманометрия”). Этот алгоритм получен путем точного воспроизведения инструкции по выполнению сфигмоманометрии из руководства по клинической профилактике, подготовленного комитетом США по профилактической медицине [2]. Алгоритмическими описаниями полны многие медицинские руководства, например, описания иммунологических методов, клиническая лабораторная диагностика, микробиологические инструкции по идентификации микроорганизмов и многое другое. Вообще говоря, процесс обследования и лечения всегда представляет собой некоторую последовательность действий, следовательно, его можно рассматривать как технологический процесс или алгоритм.

По мнению автора, учебные альбомы и компьютерные библиотеки медицинских дракон-схем могли бы принести ощутимую пользу в медицинских научных исследованиях, врачебной практике и системе медицинского образования, не говоря уже об облегчении взаимопонимания медиков между собой и с медицинскими программистами.

ДРУГИЕ ПРИМЕРЫ ВИЗУАЛИЗАЦИИ

Приведем еще несколько примеров, подтверждающих универсальность и “всеядность” языка ДРАКОН и демонстрирующих возможность его применения в различных сферах человеческой деятельности. Рисунок 110 иллюстрирует использование языка для изображения грамматических правил. На рис. 111 показан пример формализации простейших правил анализа стихотворений.

Следующий пример родился из маленького школьного “приключения”. Присутствуя на уроке в одной из московских школ, автор наблюдал за мучениями мальчика, решавшего задачу, показанную в иконе-комментарий на рис. 112. Чувствовалось, что он знает все формулы и догадывается об общем ходе решения, тем не менее у него никак не складывалась общая картина. Он не мог четко разбить задачу на отдельные этапы и выстроить из них упорядоченную последовательность, ведущую к победе. Почему? Что ему мешало? Заглянув через плечо, автор увидел в тетради “живописную мазню”. Правильные формулы прыгали

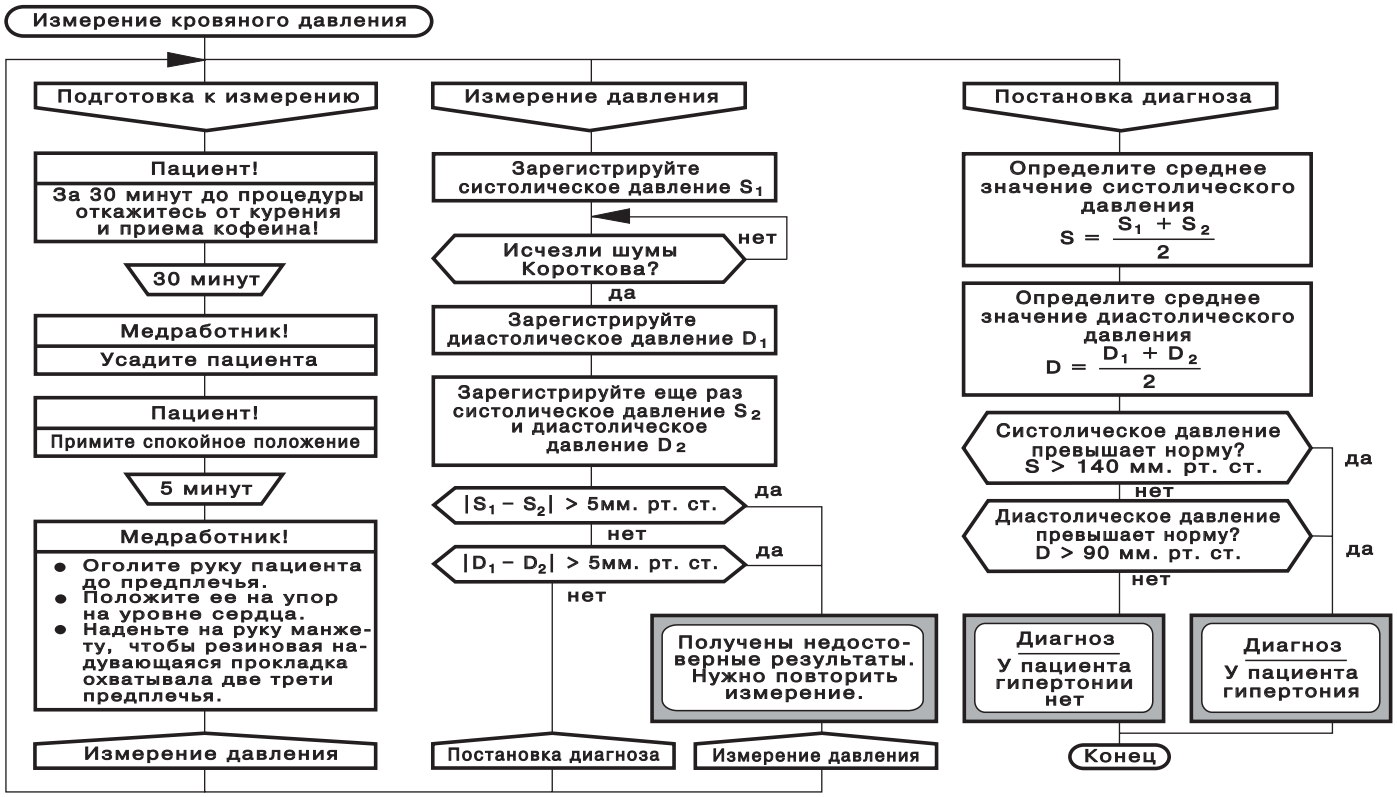


Рис. 109. **Измерение кровяного давления**

Рис. 110. **Правила написания отрицательных местоимений**

Рис. 111. **Изучаем правила поэзии: как определить стихотворный размер**

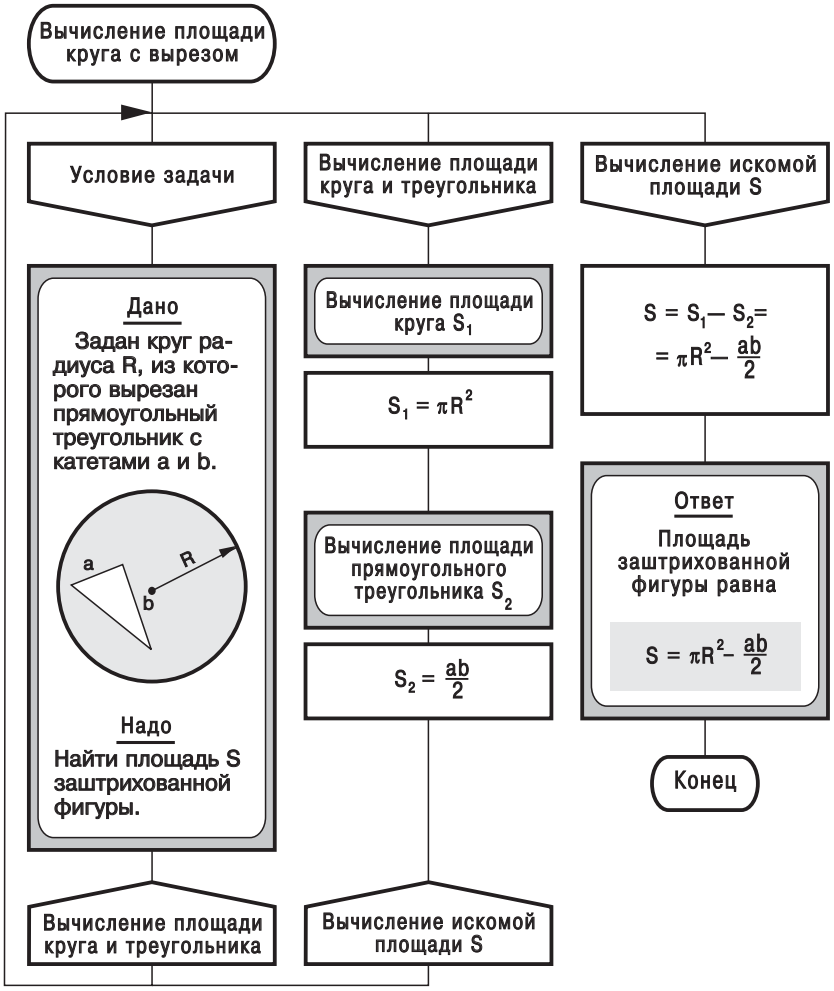


Рис. 112. **Вычисление площади круга с вырезом**

по странице, играя в чехарду: все было перечеркнуто и перевернуто вверх дном. Короче, это была плохая, неэргономичная, неупорядоченная зрительная сцена. Немудрено, что парень вконец запутался.

Обдумывая ситуацию, автор пришел к следующим предположениям. Во-первых, нужно эргономизировать зрительную сцену, придав ей форму дракон-схемы. Во-вторых, школьника нужно специально обучить, чтобы он запомнил визуальный образ дракон-схемы. В-третьих, решение задачи должно сводиться к заполнению пустых клеточек дракон-схемы; тогда запись решения волей-неволей окажется упорядоченной, а “чехарда” станет попросту невозможной. В-четвертых, нужно учить школьников не только разбивать ход решения на отдельные этапы (действия), но и придумывать для них краткие и точные заголовки. В-пятых, следует объяснить, что эти заголовки следует записывать в иконах “имя ветки” и “комментарий”.

На основании этих предположений автор по согласованию с учительницей изобразил решение, как показано на рис. 112. Чем же отличается решение на рис. 112 от традиционной записи? Выделим три наиболее важных отличия.

- Зрительная сцена имеет предельно четкую структуру. Она упорядочена и по вертикали, и по горизонтали.
- Все без исключения этапы решения и формулы имеют разъясняющие словесные заголовки. Последние записываются не где угодно, а в специальных рамочках, каждая из которых "знает свое место".
- Обеспечена симультанность восприятия: в одном визуальном поле находятся: 1) условие задачи; 2) решение; 3) ответ.

Дракон-схема на рис. 113 появилась на свет при сходных обстоятельствах.

Большинство примеров на рис. 101—113 являются адаптированными, описывающими крайне простые, даже примитивные алгоритмы. Использование "игрушечных" примеров связано с тем, что реальные ситуации слишком сложны и "не влезают" в книгу.

В качестве иллюстрации приведем названия реальных задач, для описания и решения которых целесообразно использовать техноязык.

- Расчет угла визирования по курсу и угла визирования по тангажу для определения углового положения линии визирования при полете ракеты.
- Последовательность работ и этапов, выполняемых при создании объектов вертолетной техники.
- Создание единой системы сейсмологических наблюдений и прогноза землетрясений.
- Алгоритм работы системы управления орбитального корабля "БУРАН" в режиме довыведения.
- Описание функций интеллектуальной автоматизированной системы ведения и анализа проектной документации.
- Термообработка тороидальных сердечников из железо-никелевых сплавов.
- Установка изделия (ракетоносителя с закрепленным на нем космическим кораблем) на пусковое устройство стартовой площадки.
- Расчет траектории электронов в статических электрическом и магнитном полях.
- Обработка информации, поступающей из прибора регистрации затмения солнца.
- Управление спринклерной системой реакторного отделения атомной электростанции.
- Проверка многослойных печатных плат.

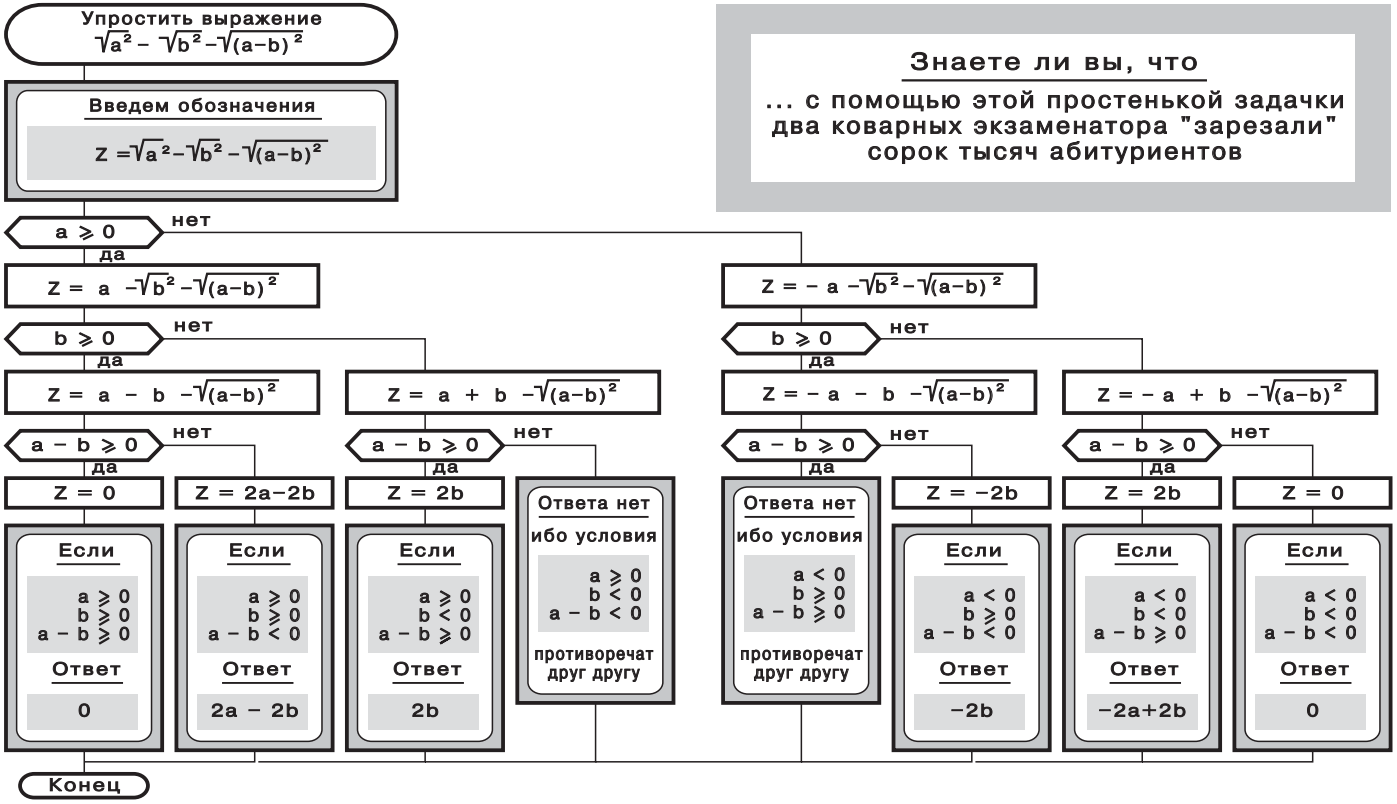


Рис. 113. **Как упростить алгебраическое выражение**

! Изучение влияния радиации и высокочастотных воздействий на организм кролика.
! Художественная вышивка на швейной машинке.
! Малоотходная технология переработки свеклы.

Наибольшие для человека трудности возникают в сложных видах деятельности, когда человеческий ум, сталкиваясь с большими и разнообразными задачами, начинает давать сбои, следствием чего являются разного рода ошибки, недоработки, дефекты и упущения, взаимное непонимание, путаница, затяжки выполнения работ и срывы плановых сроков. Техноязык ДРАКОН дает возможность ослабить или устранить подобные неприятности. Позволяя значительно упростить форму представления задачи, делая ее обозримой и ясной, ДРАКОН приносит существенную пользу, заметно повышает производительность труда, ощутимо снижает издержки.

ОПИСАНИЕ СТРУКТУРЫ ДЕЯТЕЛЬНОСТИ

Попытаемся взглянуть на рис. 101—113 под другим углом зрения. Почти все эти рисунки представляют собой конкретные примеры описания структуры деятельности в различных областях: металлообработке, химии, атомной энергетике, медицине и т. д.

Рисунки 101—113 описывают различные процессы, действия и события, которые на первый взгляд не имеют ничего общего. Если содержание этих рисунков представить в виде текстового описания, общая, *инвариантная часть* указанных процессов и событий как бы исчезает, становится скрытой, неявной, неуловимой. Образно говоря, язык ДРАКОН срывает шапку-невидимку с этого инварианта, делает его зримым, бьющим в глаза. В данном случае инвариантом является *структура деятельности*. Уточним сказанное. Любую деятельность можно описать с помощью дракон-схемы. При этом абстрактная дракон-схема является *логическим инвариантом деятельности*.

Способность абстракции, возможность увидеть единое в различном — важная способность человеческого ума. Инструменты, развивающие эту способность, содействуют увеличению “силы ума”.

Известно, что процесс обучения и образовательная среда учат человека извлекать знания из собственной деятельности, постигать принципы собственных действий и руководствоваться ими в новых ситуациях. Или, как говорят педагоги, осуществлять *перенос знаний*, занимающий огромное место в образовательном процессе и практической жизни. Язык ДРАКОН, формализуя структуру деятельности, позволяет легко выполнять перенос знаний и навыков, выявляя логические инварианты деятельности, закрепляя их в сознании и стимулируя более глубокое постижение принципов и структуры человеческих действий.

Внимательно анализируя рис. 101—113, абстрагируясь от содержания деятельности и концентрируя внимание на ее структуре и формальных аспектах описания, мы обнаруживаем “сходство в различном” и на конкретных примерах убеждаемся в том, что техноязык ДРАКОН действительно пригоден для описания структуры деятельности в самых разнообразных, непохожих друг на друга сферах деятельности. Преимущество состоит в использовании единой формы для представ-

ления технологических (императивных) знаний и описания структуры деятельности.

Для полноты картины заметим, что техноязык ДРАКОН позволяет описывать два класса процессов:

- деятельность (рис. 101—107, 109—113);
- естественные процессы, протекающие в живых организмах (рис. 108).

НУЖЕН ЛИ СТАНДАРТ ДЛЯ ОПИСАНИЯ ДЕЯТЕЛЬНОСТИ?

И в нашей стране, и за рубежом сегодня отсутствует единый стандарт для описания деятельности (структуры деятельности), что во многих случаях создает путаницу и неразбериху. Разномастные и разнокалиберные описания, принятые в различных отраслях, часто имеют многочисленные недостатки, содержат пробелы и двусмысленности. Они неформализованы, ненаглядны, неудобны в работе и трудны для понимания. Зачастую описания вообще отсутствуют, а те, что есть, — устарели и не соответствуют действительности. Несмотря на низкое качество большинства описаний, трудоемкость их создания весьма велика. Все это вносит значительные трудности в работу, заметно снижает производительность труда, создает ненужные препятствия для общения и взаимопонимания между специалистами разных профессий.

Было бы желательно создать единый стандарт для описания структуры деятельности, снабженный компьютерной поддержкой и рассчитанный на постепенное внедрение во всех отраслях и предметных областях, где его применение может дать положительный эффект. На наш взгляд, при разработке стандарта целесообразно взять за основу техноязык ДРАКОН.

Неприятность в том, что традиционные понятия “деятельность” [3, 4] и “алгоритм” [5], весьма полезные сами по себе, к сожалению, плохо приспособлены для решения поставленной задачи. Стремясь поправить дело, дадим три новых определения, которые, не претендуя на строгость (в данном случае она не нужна), позволяют выявить глубинную связь понятий “деятельность”, “алгоритм” и “техноязык”.

Деятельность — последовательность физических и информационных действий, учитывающая необходимые ограничения (условия) и позволяющая получать нужный результат за конечное число шагов (действий).

Алгоритм — описание структуры деятельности.

Техноязык — язык для описания структуры деятельности.

ВЫВОДЫ

1. Чтобы вскрыть основополагающую структуру человеческих знаний, нужно расчленить их на технологические (императивные) и декларативные. Подобное расчленение мы склонны рассматривать как генеральное деление знаний.

2. Ценность технологических (императивных) знаний состоит в том, что они теснейшим образом связаны с одним из наиболее фундаментальных понятий социально-гуманитарных наук — деятельностью.

3. Технологические (императивные) знания выявляют, закрепляют в сознании и объективируют структуру деятельности.

4. Важным свойством деятельности является существование глубинных логических инвариантов (структурных конструкций), выражаемых с помощью понятия “абстрактная дракон-схема”.

5. Традиционные способы описания деятельности не позволяют выявить глубинные инварианты; последние оказываются замаскированными, скрытыми, спрятанными от читателя. Это обстоятельство затрудняет перенос знаний и навыков, играющий важную роль в образовании и практической жизни.

6. Техноязык как особое средство, специально созданное для описания структуры деятельности, позволяет выявить и обнажить логические инварианты деятельности, сделать их явными, зримыми, доступными для всех людей.

7. Формализация знаний — труд, производительность которого играет важную роль (см. гл. 3). Если этот труд слишком сложен (производительность труда мала), то формализацию могут выполнять только специально подготовленные элитные специалисты; при таких условиях автоформализация практически невозможна. И наоборот, если данный труд удается облегчить, формализация становится посильной почти для каждого — только в этом случае создаются необходимые предпосылки для автоформализации.

8. Техноязык ДРАКОН кардинальным образом облегчает труд формализации и повышает его производительность. Следовательно, техноязык пригоден для эффективной автоформализации технологических знаний.

9. Целесообразно создать единый межотраслевой стандарт для описания структуры деятельности. Стандарт должен опираться на техноязык ДРАКОН, рассматриваемый как единое средство для описания структуры деятельности методом автоформализации технологических знаний.

10. Описание структуры деятельности и результаты формализации технологических знаний целесообразно хранить в компьютерной форме (в виде библиотек визуального гипертекста), а также в виде бумажных визуальных альбомов подходящего формата (например, формата А3).

Глава 14
ВИЗУАЛЬНЫЙ ДРАКОН-РЕДАКТОР

Овладение техникой визуализации научно-технической информации, умение представить ее в виде ясного и простого рисунка имеют большое значение.

Валерий Венда

ЗАЧЕМ НУЖЕН ДРАКОН-РЕДАКТОР?

Разумеется, в случае крайней нужды дракон-схему можно нарисовать и вручную. Однако это не лучший способ. Гораздо удобнее воспользоваться специальной программой, которая называется “ДРАКОН-редактор”. Если же нужно создать ДРАКОН-программу, ручное рисование вообще исключается. Без ДРАКОН-редактора ввести ДРАКОН-программу в компьютер невозможно.

В состав редактора входит меню графоэлементов (рис. 114). Чтобы нарисовать дракон-схему, пользователь сначала вызывает меню на экран персонального компьютера, а затем с его помощью рисует или, как говорят, конструирует дракон-схему. При этом важную роль играют так называемые заготовки.

ЗАГОТОВКА-ПРИМИТИВ И ЗАГОТОВКА-СИЛУЭТ

Чтобы вырастить огромное дерево, нужно бросить в землю маленькое семечко. Любая сколько угодно сложная дракон-схема тоже вырастает из “семечка”, которое называется заготовкой. Заготовки бывают двух сортов: одна используется для построения дракон-схемы “примитив”, из другой получается силуэт (рис. 115).

Вообще говоря, обе заготовки следовало бы поместить в меню. Однако в целях экономии применяется следующий прием: в меню находится только одна из них — заготовка-силуэт (рис. 114, вверху слева), а заготовка-примитив получается из нее в результате простой операции, описание которой мы для краткости опускаем.

Построение любой дракон-схемы выполняется за конечное число шагов путем соответствующих преобразований выбранной заготовки.

ЧТО ТАКОЕ АТОМ?

Элемент меню на рис. 114 называется *атомом*, если он имеет два вертикальных отростка.

ДРАКОН-редактор может выполнять несколько операций, среди которых важную роль играет команда “ввод атома” (рис. 116). Операция

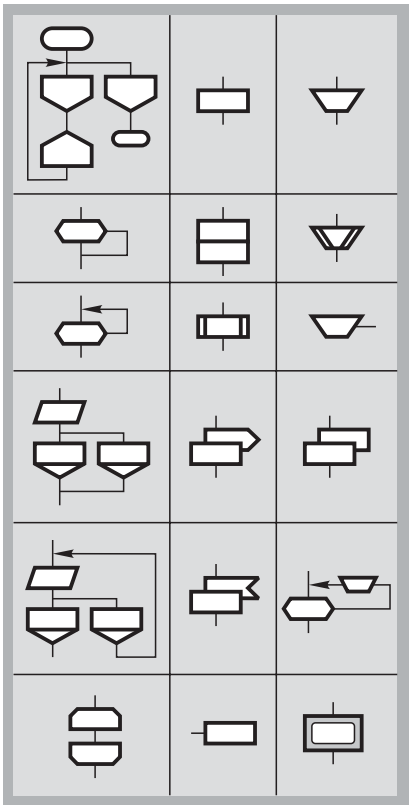


Рис. 114. **Меню графоэлементов ДРАКОН-редактора**

Рис. 115. **Преобразуя заготовки с помощью фиксированного набора формальных визуальных операций, можно построить любую дракон-схему**

выполняется в два этапа: сначала пользователь выбирает нужный атом из меню, затем обращается к дракон-схеме и указывает точку, в которую нужно его ввести.

Атомы вставляются не куда попало, а только в разрешенные места, которые называются *валентными точками* дракон-схемы. Перечень точек включает:

- валентные точки заготовок (отмечены на рис. 115);
- валентные точки макроикон (отмечены на рис. 2);
- входы и выходы атомов.

Ввод атома производится так: происходит разрыв соединительной линии в выбранной пользователем валентной точке, после чего в место разрыва вставляется атом, как показано на рис. 116.

В реальных дракон-схемах валентные точки не изображаются, а подразумеваются.

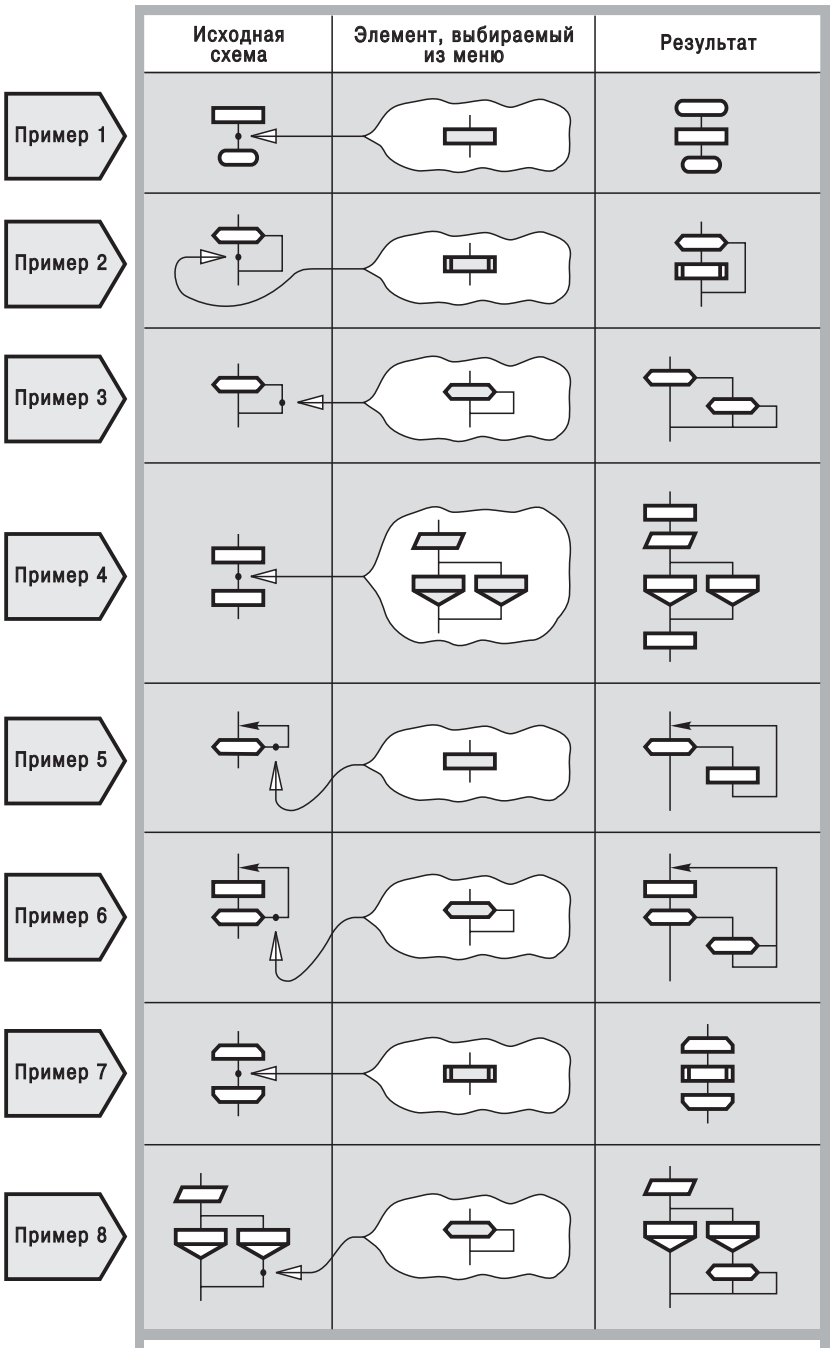


Рис. 116. **Примеры выполнения операции “ввод атома”**

ПРИМЕР ПОСТРОЕНИЯ ДРАКОН-СХЕМЫ “ПРИМИТИВ”

Дракон-схема строится на экране компьютера методом “сборки из кубиков”. Чтобы посмотреть, как это делается, нарисуем схему, изображенную на рис. 117. В начале работы пользователь вызывает на экран визуальное меню (рис. 114) и размещает его в удобном для себя месте, например в правом верхнем углу экрана. Остальная часть экрана используется как рабочее поле для построения дракон-схемы.

Схема, которую мы хотим построить (рис. 117), — это примитив. Поэтому прежде всего нужно поместить в рабочее поле заготовку-примитив. Для этого на первом шаге графического конструирования пользователь обращается к меню, подводит курсор к макроиконе “заготовка-силуэт”, преобразует ее в заготовку-примитив и копирует последнюю в рабочее поле экрана (рис. 118, шаг 1). На втором шаге пользователь вызывает из меню икону “действие”. Но куда ее поместить? Пользователь переводит курсор в рабочее поле и указывает точку в заготовке-примитив, в которой следует разорвать соединительную линию, чтобы в образовавшийся разрыв вставить выбранную икону. Результат операции виден на рисунке (рис. 118, шаг 2).

Два следующих шага выполняются аналогично: в дракон-схему вводятся еще одна икона “действие” и макроикона “переключатель” (рис. 118, шаги 3 и 4).

Далее следует операция “добавление варианта”, выполняемая без обращения к меню. С ее помощью модифицируется макроикона “переключатель”, к которой добавляется еще одна икона “вариант” (рис. 118, шаг 5).

Последующий ход строительства ясен из рис. 118 (шаги 6—8). После того как графический узор (слепыш) дракон-схемы построен, производится заполнение его текстом. Окончательный результат работы редактора показан на рис. 117.

ОПЕРАЦИЯ “ПЕРЕСАДКА ЛИАНЫ”

Обезьяна, сидевшая на дереве, поймала свисавшую сверху лиану. Однако нижняя часть лианы приросла к стволу и не поддавалась. Обезьяна перегрызла ее зубами, уцепилась за конец и мигом перелетела на соседнее дерево, где намертво привязала лиану к ветке.

Нечто подобное умеет делать и ДРАКОН-редактор. Роль верхнего конца лианы играет выход иконы “вопрос” или “вариант”. *Лиана* — это присоединенная к нему последовательность шампур-блоков или просто соединительная линия. При некоторых условиях (подробно описанных в следующей главе) нижний конец лианы можно оторвать от своего места и присоединить в другую точку дракон-схемы. Такая операция называется *пересадка лианы*.

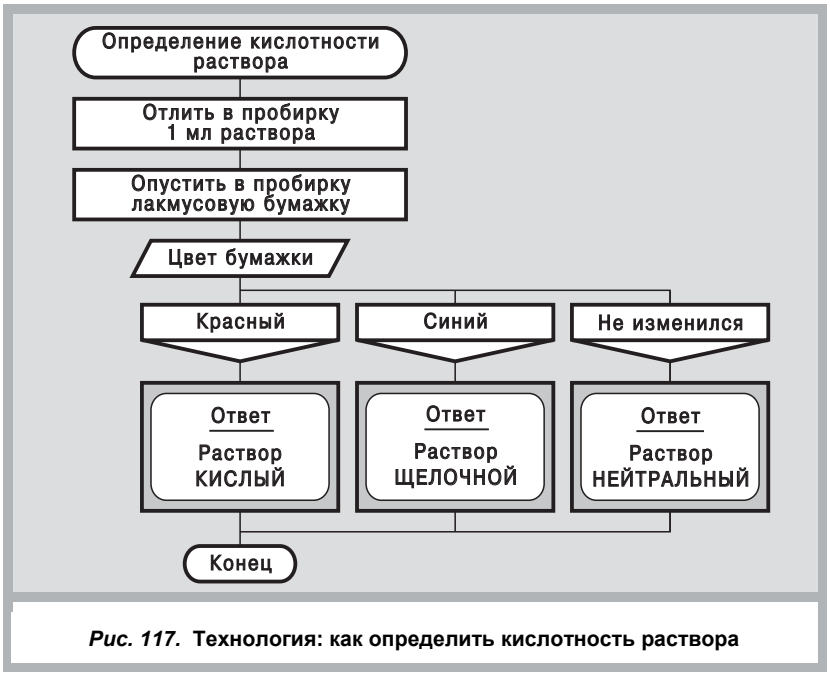


Рис. 117. **Технология: как определить кислотность раствора**

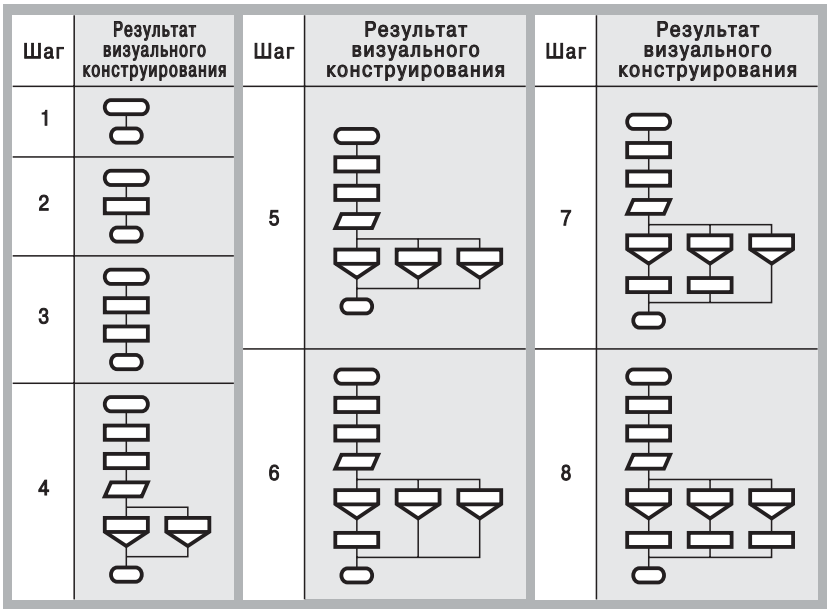


Рис. 118. **Конструирование дракон-схемы “примитив” с помощью ДРАКОН-редактора**

Выполнение этой операции не требует обращения к меню и осуществляется в два этапа. Сначала курсор подводится к нижнему концу лианы, который пользователь желает освободить (рис. 119, левая графа). Но куда его присоединить? Пользователь выбирает желаемую точку и отмечает ее курсором (рис. 119, средняя графа). Результат операции "пересадка лианы" показан на том же рисунке в правой графе.

Многие дракон-схемы, представленные в этой книге, построены с помощью пересадки лианы. Укажем некоторые из них: рис. 26*б*, 27*б*, 40, 41, 42, 57*в*, 60*в*, 67—74.

ОПЕРАЦИЯ "ЗАЗЕМЛЕНИЕ ЛИАНЫ"

Функция "пересадка лианы" универсальна в том смысле, что она применима и к примитиву, и к силуэту. В отличие от нее операция *заземление лианы* относится только к силуэту. Она служит для построения веток, имеющих несколько выходов (многоадресных веток). Для этого необходимо организовать в ветке разветвление (с помощью макроикон "развилка" или "переключатель"), затем оторвать присоединенную к ним лиану от прежнего места и присоединить ее через икону "адрес" к нижней горизонтальной линии силуэта, "заземлить" ее.

Операция "заземление лианы" проводится в два этапа без обращения к меню. Первый этап (отрыв нижнего конца лианы от своего места) осуществляется точно так же, как при пересадке лианы (рис. 120, левая графа). На втором этапе пользователь подводит курсор к нижней линии силуэта, указывая точку, куда лиана может дотянуться, не пересекая других линий (рис. 120, средняя графа). Это действие порождает автоматическое появление в нужном месте иконы "адрес", через которую лиана и присоединяется к нижней линии (рис. 120, правая графа).

Заземление лианы использовалось при построении силуэтов с многоадресными ветками, например, на рис. 4, 6*а*, 51–54, 84, 88, 89.

ПРИМЕР ПОСТРОЕНИЯ ДРАКОН-ПРОГРАММЫ "СИЛУЭТ"

Построим визуальную программу, изображенную на рис. 96. На первом шаге конструирования пользователь обращается к меню и вызывает в рабочее поле заготовку-силуэт (рис. 121, шаг 1). На следующем шаге с помощью операции "добавление ветки" (которая выполняется без обращения к меню) он модифицирует заготовку, вставляя в силуэт еще одну ветку (рис. 121, шаг 2). Дальнейший ход строительства ясен из рисунка: в шагах 3—8, 10—13 реализуется операция "ввод атома", в шаге 9 — "заземление лианы". Графическое конструирование заканчивается в момент получения желаемого слепыша (рис. 121, шаг 13).

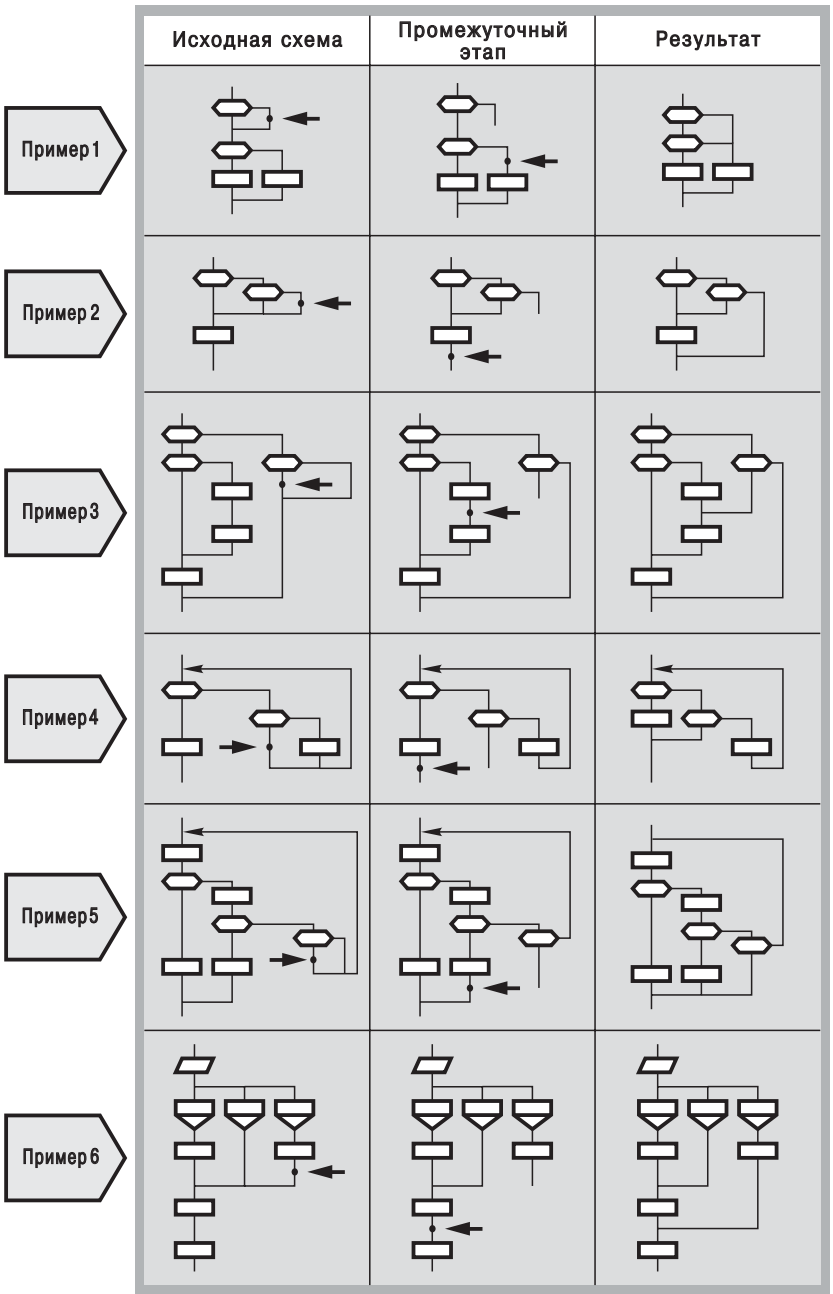


Рис. 119. **Примеры выполнения операции “пересадка лианы”**

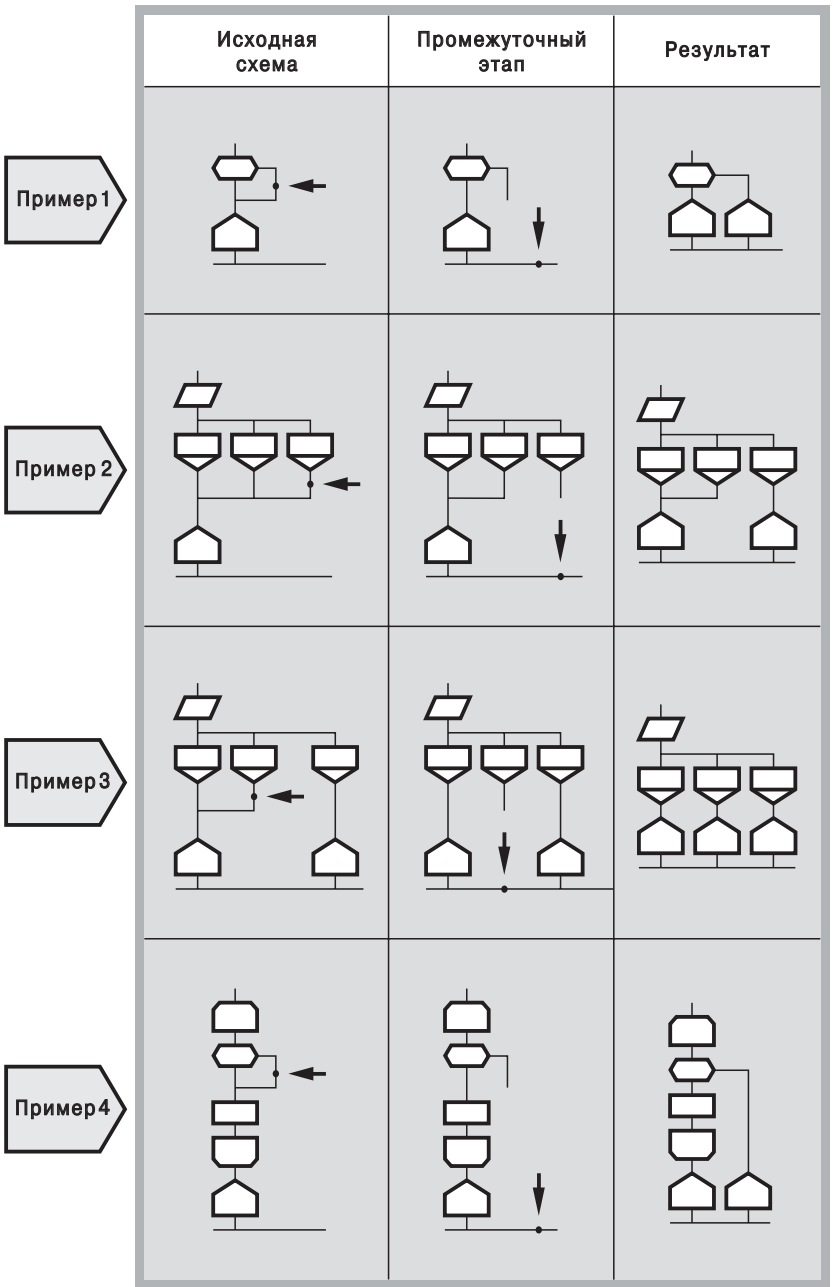


Рис. 120. **Примеры выполнения операции “заземление лианы”**

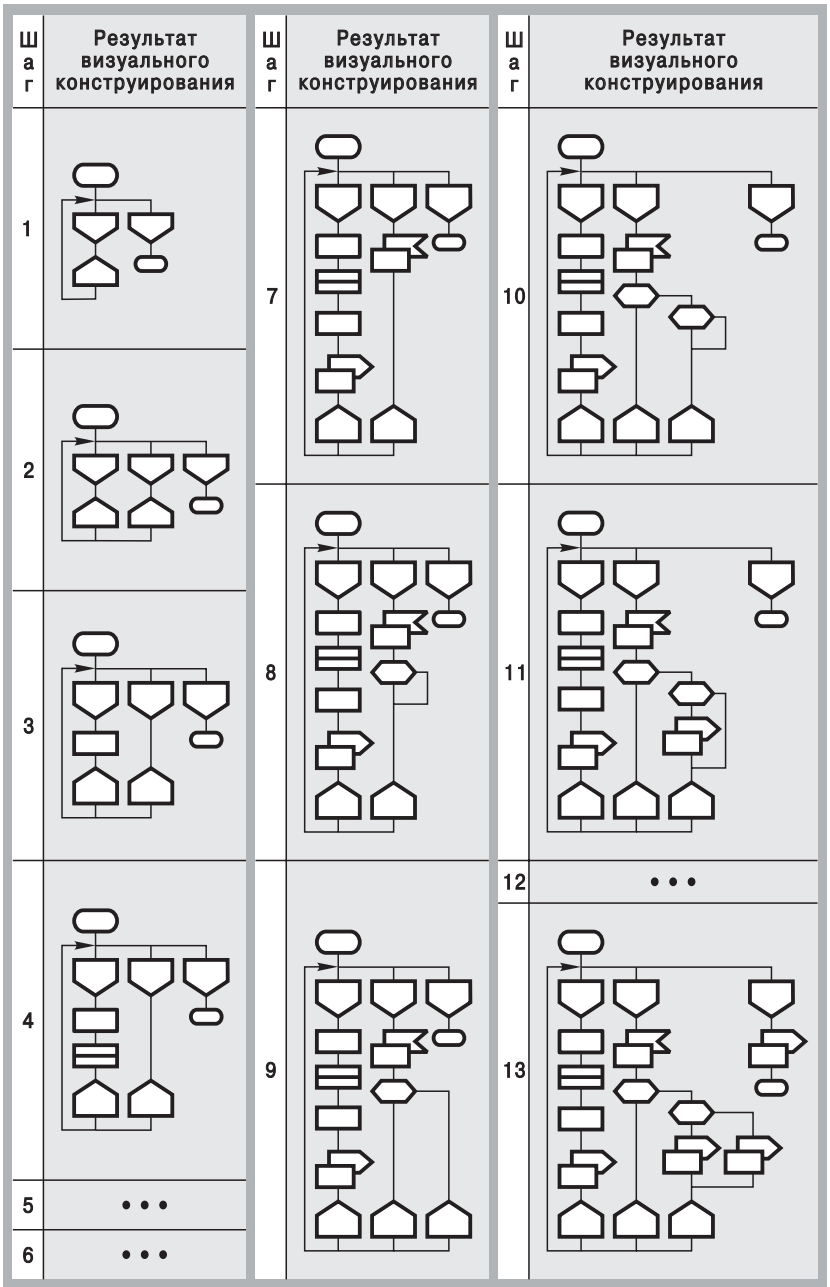


Рис. 121. **Конструирование дракон-схемы “силуэт” с помощью ДРАКОН-редактора**

Затем в иконах записываются текстовые операторы, после чего визуальная программа приобретает окончательный вид, показанный на рис. 96. Данная программа может выполняться методом интерпретации или же транслироваться в объектный код.

ФОРМИРОВАНИЕ НАДПИСЕЙ “ДА” И “НЕТ”

Вернемся к обсуждению рис. 121 и подробнее остановимся на шаге 13, на котором завершается графическое построение дракон-схемы и в качестве результата на экране возникает слепыш. Дело в том, что законченный чертеж-слепыш, сформированный при реальной работе ДРАКОН-редактора, имеет отличие от абстрактной дракон-схемы, изображенной на рис. 97. Последняя полностью лишена текста (не содержит ни одной буквы или цифры), а построенный редактором слепыш возле каждой иконы “вопрос” обязательно имеет надписи “да” и “нет”. Эти надписи появляются на дракон-схеме всякий раз, когда из меню вызывается элемент с иконой “вопрос” в ходе операции “ввод атома”. Согласно алгоритму этой операции редактор пишет “да” у нижнего выхода иконы “вопрос” и “нет” — у правого. Чтобы пользователь в случае необходимости мог поменять их местами, в редакторе предусмотрена операция “да-нет”. Применение последней к конкретной иконе “вопрос” приводит к тому, что слова “да” и “нет” у ее выходов меняются местами (при этом все остальные элементы дракон-схемы остаются на своих местах).

ВЫВОДЫ

1. Хотя общее число икон и макроикон языка ДРАКОН равно 45, для построения любой дракон-схемы достаточно иметь небольшое меню, содержащее всего 18 графоэлементов.
2. Визуальное меню существенно облегчает работу пользователя — оно дает возможность конструировать дракон-схему методом “сборки из кубиков”. Для этого служит операция “ввод атома”, позволяющая доставать “кубики” из меню и укладывать их в проектируемую дракон-схему.
3. Другие операции (“пересадка лианы”, “заземление лианы” и т. д.) разрешают вносить в схему логические детали и “украшения”, расширяющие ее функциональные возможности и улучшающие эргономическое качество.
4. В алгоритмах ДРАКОН-редактора реализован полный набор правил визуального синтаксиса, что освобождает пользователя от необходимости подробно запоминать синтаксические правила.
5. ДРАКОН-редактор создает только правильно построенные схемы и исключает возможность появления запрещенных схем. Отсюда вытекает, что при работе с ДРАКОН-редактором ошибки визуального синтаксиса принципиально невозможны.

Глава 15

ОПИСАНИЕ ВИЗУАЛЬНОГО СИНТАКСИСА ЯЗЫКА ДРАКОН

> ...наглядность, говоря обыденным языком, в один день научает нас с большей легкостью и прочностью тому, чему не могут научить правила, повторяемые хотя бы тысячу раз, так как собственное наблюдение ... идет здесь рука об руку с теоретическим определением.
>
> *Галилео Галилей*

ОБЩИЕ ПОНЯТИЯ

Тезис 1. Иконы — визуальные буквы, образующие визуальный алфавит языка ДРАКОН, представленные на рис. 1.

Тезис 2. Заготовка-примитив и заготовка-силуэт — фигуры, показанные на рис. 115.

Предварительный тезис 3. Примитив — фигура, полученная путем преобразования заготовки-примитив за конечное число шагов с помощью фиксированного набора операций (перечисленных ниже в тезисе 36).

Предварительный тезис 4. Силуэт — фигура, полученная путем преобразования заготовки-силуэт за конечное число шагов с помощью фиксированного набора операций (перечисленных ниже в тезисе 37).

Тезис 5. Дракон-схема — общее понятие для обозначения примитива и силуэта.

ШАМПУР-БЛОК

Тезис 6. Шампур-блок — часть дракон-схемы, имеющая один вход сверху и один выход снизу, содержащая одну или несколько икон, причем:

- ! вход и выход лежат на одной вертикали, через которую проходит путь от входа к выходу;
- ! через каждую икону, входящую в состав шампур-блока, проходит хотя бы один путь от входа к выходу;
- ! в состав шампур-блока могут входить любые иконы за исключением следующих: заголовок, конец, формальные параметры, петля силуэта.

Тезис 7. Главная вертикаль шампур-блока — вертикаль, соединяющая его вход и выход.

Остальные вертикали, если они есть, находятся правее главной. Все вертикали шампур-блока ориентированы сверху вниз, кроме цепей, используемых для организации петли цикла.

ОПЕРАЦИЯ “ВВОД АТОМА”

Тезис 8. Атомы — фигуры, изображенные на рис. 122. Эти фигуры используются в операции “ввод атома”. Любой атом является шампур-блоком.

Тезис 9. Валентная точка — точка, принадлежащая заготовке или дракон-схеме, в которой разрешается произвести разрыв соединительной линии, чтобы в место разрыв вставить атом с помощью операции “ввод атома”.

Тезис 10. Ввод атома — преобразование заготовки или дракон-схемы, выполняемое следующим образом: производится разрыв соединительной линии в валентной точке и в это место вставляется атом, как показано на рис. 116.

Дополнительные сведения об атомах

Тезис 11. Атомы делятся на простые и составные. Простой атом состоит из одной иконы, составной содержит не менее двух (рис. 122).

Тезис 12. Функциональный атом — простой атом, не являющийся пустым оператором. Таковы все простые атомы, кроме комментария.

Тезис 13. Составные атомы бывают пустые и непустые. В непустом есть хотя бы один функциональный атом. В пустом нет ни одного.

Тезис 14. В полностью законченной дракон-схеме не должно быть ни одного пустого атома (так как последний эквивалентен пустому оператору). Пустые атомы разрешается использовать на всех этапах построения дракон-схемы, кроме заключительного.

Критические и нейтральные точки

Тезис 15. Валентные точки делятся на нейтральные и критические.

Тезис 16. Точка называется нейтральной, если применение операции “ввод атома” к данной точке является возможным, но не обязательным. В отличие от нее критическая точка требует обязательного ввода атома.

Тезис 17. Валентные точки находятся в заготовках и атомах. Они показаны на рис. 115 и 122, где нейтральные точки обозначены светлыми кружками, критические — жирными точками.

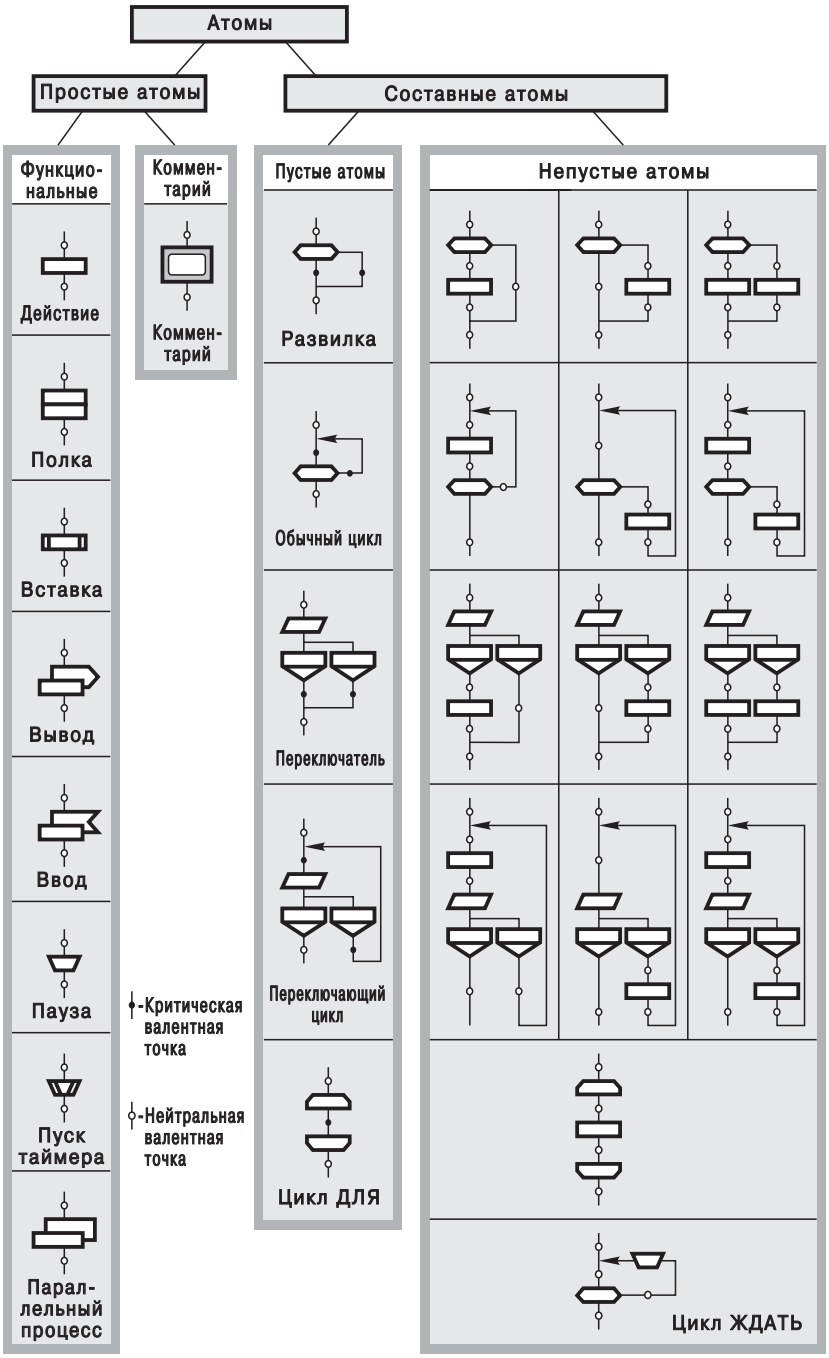


Рис. 122. **Атомы и валентные точки**

Тезис 18. Если в фигуре (заготовке или атоме) одна критическая точка, ввод атома обязательно производится именно в нее; при этом критическая точка уничтожается. Если фигура имеет две критические точки, обязательный ввод атома делается только в одну из них; при этом критическая точка, в которую произведен ввод, уничтожается, а другая критическая точка нейтрализуется, т. е. становится нейтральной.

Тезис 19. Полная совокупность критических точек охватывает:

- критические точки в пустых атомах;
- одну критическую точку в заготовке-примитив;
- одну критическую точку в заготовке-силуэт.

Тезис 20. Полная совокупность нейтральных точек охватывает:

- входные и выходные точки атомов;
- две внутренние точки в атоме “цикл ЖДАТЬ”;
- одну точку в заготовке-силуэт;
- точки, полученные в результате нейтрализации критических точек.

Правила использования операции “ввод атома” при построении дракон-схемы

Тезис 21. Операция “ввод атома” применяется для ввода только простых и пустых атомов, а также цикла ЖДАТЬ. Ввод непустого атома осуществляется в два этапа; сначала вводится пустой атом, затем в его критическую точку вводится функциональный атом.

П о я с н е н и е. Ввод пустого атома — очень удобный строительный прием. Он позволяет обеспечить богатство и разнообразие создаваемых дракон-схем и используемых в них конфигураций. Среди последних особую роль играет так называемая “матрешка”.

Тезис 22. *Матрешка* — фигура, полученная путем ввода пустого атома в критическую точку пустого атома, а также путем многократного вложения пустых и непустых атомов друг в друга (рис. 123).

Тезис 23. Матрешка бывает пустой (если все содержащиеся в ней атомы пустые), частично пустой (если в ней есть как пустые, так и непустые атомы) и непустой (если все ее атомы непустые). См. рис. 124—126.

П о я с н е н и е. После того как пользователь эффективно использовал пустые атомы для придания дракон-схеме желаемой конфигурации, он должен убрать их из схемы.

Тезис 24. Чтобы устранить пустые атомы из дракон-схемы, есть два способа:

- превратить пустой атом в непустой;
- преобразовать пустой атом в пустую матрешку, затем превратить ее в непустую.

Тезис 25. Устранение из дракон-схемы пустых атомов автоматически приводит к уничтожению всех критических точек.

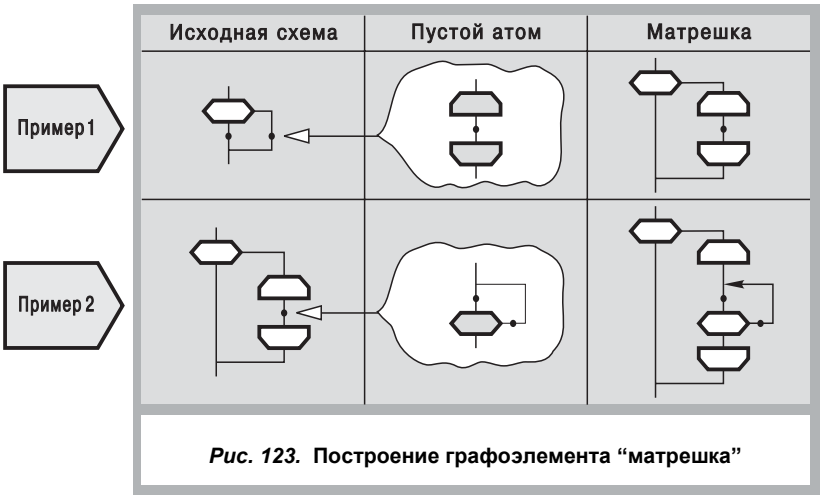


Рис. 123. **Построение графоэлемента “матрешка”**

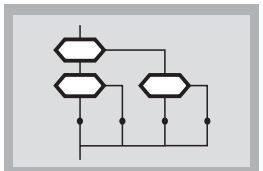


Рис. 124. **Пустая матрешка**

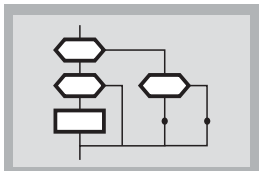


Рис. 125. **Частично пустая матрешка**

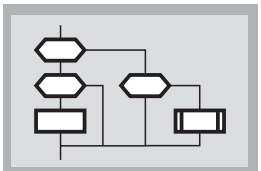


Рис. 126. **Непустая матрешка**

Рис. 127. **Ошибка “сиамские близнецы” — это запрещенная связь между ветками**

ОПЕРАЦИИ С ЛИАНОЙ

Тезис 26. Лиана — часть дракон-схемы, имеющая один вход и один выход, именуемые “началом лианы” и “концом лианы” соответственно. Началом лианы может быть любой выход икон “вопрос” и “вариант”, если он (выход) не является петлей цикла. Концом лианы считается точка слияния, в которой нижняя часть лианы соединяется с другой линией (концом лианы не может быть неразветвленный вход иконы).

Тезис 27. Лиана может быть нагруженной (если она содержит иконы) и ненагруженной (если это просто линия).

Пересадка лианы

Тезис 28. Пересадка лианы — преобразование дракон-схемы, выполняемое за четыре шага.

Шаг 1. Производится отрыв конца лианы от точки присоединения (рис. 119).

Шаг 2. Конец лианы с помощью вертикальных и горизонтальных линий присоединяется к любой валентной точке, куда лиана может дотянуться без пересечения с другими линиями (рис. 119). При этом запрещается:

- ! формировать второй вход в ветку (ошибка “сиамские близнецы” — см. рис. 127);
- ! образовывать новый цикл;
- ! создавать второй вход в цикл.

Однако разрешается строить новый путь из середины обычного цикла к единственному входу в этот цикл, создавая визуальный эквивалент оператора *continue* языка СИ (см. рис. 90, пример 7, а также рис. 41).

Шаг 3. Производится эквивалентное преобразование топологии дракон-схемы, чтобы лиане не пришлось загибаться наверх (рис. 128) и соблюдались правила построения шампур-блока (рис. 129).

Шаг 4. Устраняются неоправданные изломы линий (рис. 130).

Заземление лианы

Тезис 29. Заземление лианы — преобразование дракон-схемы, выполняемое за четыре шага.

Шаг 1. Производится отрыв конца лианы от точки присоединения (рис. 120).

Шаг 2. Конец лианы с помощью вертикальной линии присоединяется к любой точке нижней горизонтальной линии силуэта, куда он может дотянуться, не пересекая другие линии.

Шаг 3. Производится разрыв линии в нижней части лианы и в место разрыва вставляется икона “адрес” (рис. 120).

Шаг 4. Устраняются неоправданные изломы линий.

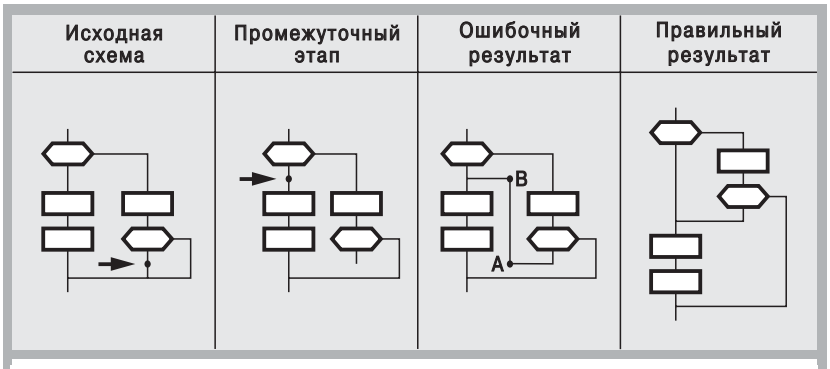


Рис. 128. **Ошибка при пересадке лианы. Вертикаль AB ориентирована снизу вверх, что запрещено**

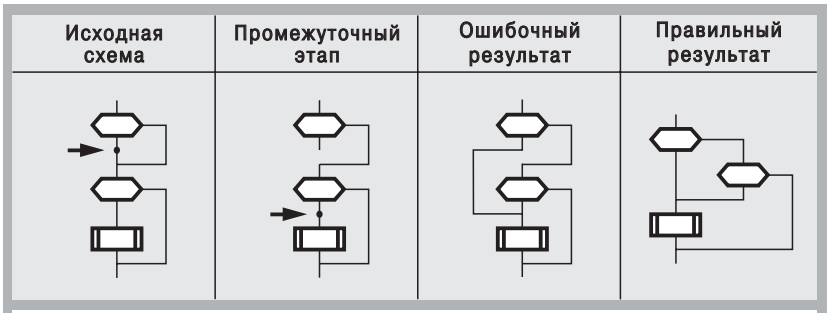


Рис. 129. **Ошибка при пересадке лианы. Нарушено правило "вход и выход шампур-блока лежат на одной вертикали, через которую проходит путь от входа к выходу"**

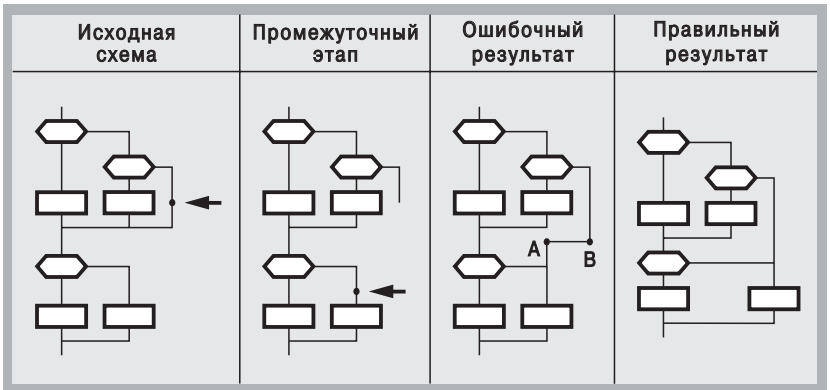


Рис. 130. **Ошибка при пересадке лианы. В точках A и B имеются неоправданные изломы линий, которые можно устранить**

ПРОЧИЕ ОПЕРАЦИИ

Тезис 30. Боковое присоединение — преобразование дракон-схемы, с помощью которого в схему добавляются иконы “синхронизатор” или “формальные параметры”.

Икона “синхронизатор” размещается слева от другой иконы и соединяется с ней горизонтальным отростком. Перечень икон, к которым осуществляется боковое присоединение синхронизатора, показан на рис. 2 (п. 8—20).

Икона “формальные параметры” размещается справа от иконы “заголовок” и соединяется с ней горизонтальным отростком, как показано на рис. 2 (п. 1).

Тезис 31. Добавление варианта — преобразование дракон-схемы, с помощью которого в атом “переключатель” добавляется еще одна икона “вариант”. Число добавлений не более 14, так что максимальное число вариантов в переключателе равно 16.

Тезис 32. Добавление ветки — преобразование силуэта, в который добавляется еще одна ветка. Число добавлений не более 14, так что максимальное число веток в силуэте равно 16.

Тезис 33. Удаление последней ветки — преобразование силуэта, при котором удаляется крайняя правая ветка. Этот прием используется при описании бесконечного параллельного процесса, как показано в примерах на рис. 88 и 89.

Тезис 34. Удаление конца примитива — преобразование примитива, при котором удаляется икона “конец”. Это необходимо для описания бесконечного параллельного процесса.

Тезис 35. Дополнительный вход — преобразование силуэта, с помощью которого добавляется еще одна икона “заголовок”, которая размещается над любой иконой “имя ветки” (кроме левой) и соединяется с ней вертикальным отростком. При этом на верхней горизонтальной линии силуэта рисуют направленную вправо стрелку, как показано в примере на рис. 84 справа.

О г р а н и ч е н и е. При наличии веточного цикла запрещается присоединять дополнительный заголовок к середине веточного цикла.

ОСНОВНЫЕ РЕЗУЛЬТАТЫ

Тезис 36. Любая правильно построенная дракон-схема “примитив” является результатом преобразования заготовки-примитив с помощью конечного числа операций: ввод атома, пересадка лианы, добавление варианта, боковое присоединение, удаление конца примитива.

Тезис 37. Любая правильно построенная дракон-схема “силуэт” является результатом преобразования заготовки-силуэт с помощью конечного числа операций: ввод атома, добавление ветки, пересадка лианы, заземление лианы, добавление варианта, боковое присоединение, удаление последней ветки, дополнительный вход.

П о я с н е н и е. Тезисы 36 и 37 могут рассматриваться как окончательные определения понятий “примитив” и “силуэт”.

В Ы В О Д Ы

1. Изложенные выше 37 тезисов (вместе с рисунками, на которые они ссылаются) дают однозначное описание визуального синтаксиса, которое при желании можно изложить строгим математическим языком.
2. Это описание является достаточным для построения ДРАКОН-редактора, способного решить две задачи. Во-первых, нарисовать (в соответствии с указаниями пользователя) любую абстрактную дракон-схему, принадлежащую множеству правильно построенных (удовлетворяющих требованиям визуального синтаксиса) дракон-схем. Во-вторых, создать в памяти компьютера формальное описание построенной схемы, пригодное (после заполнения икон надлежащими текстовыми операторами) для трансляции в объектные коды или для выполнения программы в режиме интерпретации.

Глава 16

ВИЗУАЛЬНОЕ СТРУКТУРНОЕ ПРОГРАММИРОВАНИЕ

Наше мышление основано в первую очередь на зрительном восприятии.

Вадим Глезер

ПОСТАНОВКА ПРОБЛЕМЫ

Попробуем включить воображаемый “боковой прожектор” и взглянуть на проблему под другим углом зрения. Существует некоторое, причем весьма глубокое, хотя и не всегда очевидное сходство между изложенными выше идеями и концепцией структурного программирования. Исходя из этого, введем термин “визуальное структурное программирование” и определим его как набор правил, совпадающий с визуальным синтаксисом языка ДРАКОН. В концентрированном виде эти правила изложены в гл. 15.

Чтобы отграничить теоретические аспекты визуального структурного программирования от второстепенных деталей, нам понадобится термин “шампур-метод”. Впрочем, иногда выражения “шампур-метод” и “визуальное структурное программирование” будут использоваться как синонимы.

Размышляя над проблемой, автор пришел к следующим предварительным выводам или, лучше сказать, предположениям.

- Несмотря на наличие целого ряда общих признаков, текстовое и визуальное структурное программирование — существенно разные вещи.
- Традиционное (текстовое) структурное программирование является, по-видимому, наилучшим решением соответствующей задачи для традиционного (текстового) программирования.
- Для визуального программирования подобное утверждение неправомерно. Можно, конечно, тупо перенести правила текстового структурного программирования на визуальный случай. Но это не будет хорошим решением.
- Чтобы разработать эффективный метод структуризации для визуального варианта, необходимо, взяв за основу правила текстового структурного программирования, значительно модифицировать их.
- Принципы структуризации, положенные в основу визуального синтаксиса языка ДРАКОН, являются искомым решением.

В данной главе сделана попытка обосновать заявленные выводы.

ИСТОРИЧЕСКАЯ СПРАВКА

Согласно классической теореме Бома и Джакопини, всякая реальная программа может быть построена из функциональных блоков (действий) и двух конструкций: цикла и дихотомического выбора (развилки) [1]. Эдсгер Дейкстра обогатил и усилил эту идею, предложив отказаться от оператора безусловного перехода *goto* и ограничиться тремя управляющими конструкциями: последовательность, цикл, выбор [2].

Дональд Кнут подверг критике тезис Дейкстры о полном исключении *goto*, продемонстрировав случаи, где *goto* полезен. В итоге возникла плодотворная дискуссия, строго говоря, не завершенная до сих пор, в ходе которой выявились четыре варианта мнений (табл. 4).

Таблица 4

Позиция участников дискуссии	**Используются три структурные конструкции?**	**Используются заменители *goto*?**	**Используются *goto*?**
Вариант 1	Да	Нет	Нет
Вариант 2	Да	Нет	Да
Вариант 3	Да	Да	Да
Вариант 4	Да	Да	Нет

Вариант 1 описывает ортодоксальную позицию Дейкстры, согласно которой оператор *goto* имеет “гибельные последствия” и поэтому “должен быть исключен из всех языков программирования высокого уровня”. Исходя из этого были разработаны языки без *goto*: PDL [3], BLISS и др.

Вариант 2 отражает мнение ранних критиков Дейкстры, позиция которых выражается словами: “использование оператора *goto* может оказаться уместным в лучших структурированных программах”; “всегда были примеры программ, которые не содержат *goto* и аккуратно расположены лесенкой в соответствии с уровнем вложенности операторов, но совершенно непонятны, и были другие программы, содержащие *goto* и все же совершенно понятные”. Поэтому нужно “избегать использования *goto* всюду, где это возможно, но не ценой ясности программы” [4].

Как известно, полемика по *goto* “растревожила осиное гнездо” и всколыхнула “весь программистский мир”. Варианты 1 и 2 выражают крайние позиции участников дискуссии, между которыми, как казалось вначале, компромисс невозможен. Однако ситуация изменилась с изобретением и широким распространением *заменителей goto*, примерами которых являются: в языке СИ — операторы *break*, *continue*, *return* и функция *exit* ( ), в языке МОДУЛА-2 — операторы RETURN, EXIT, процедура HALT и т. д.

Заменители — особый инструмент, который существенно отличается как от трех структурных управляющих конструкций, так и от *goto*. Они обладают двумя важными преимуществами по сравнению с *goto*:

1) не требуя меток, заменители исключают ошибки, вызванные путаницей с метками;

2) каждый заменитель может передать управление не куда угодно (как *goto*), а в одну-единственную точку, однозначно определяемую ключевым словом заменителя и его местом в тексте. В результате множество точек, куда разрешен переход, сокращается на порядок, что также предотвращает ошибки.

Вариант 3 описывает языки СИ, АДА и др., где имеются заменители и на всякий случай сохраняется *goto*.

Вариант 4 соответствует языку МОДУЛА-2, где также есть заменители, однако *goto* исключен. Следует подчеркнуть, что при наличии заменителей сфера применения *goto* резко сужается, так что удельный вес ошибок, связанных с его применением, заметно уменьшается; поэтому вопрос об исключении *goto* теряет прежнюю остроту.

Идея структуризации оказала большое влияние на практику. Практически во все императивные языки высокого уровня на этапе их разработки или позже были введены структурные конструкции цикла и ветвления, а также, что очень важно, различные заменители *goto*.

Отживающий метод?

Вместе с тем структурное программирование породило преувеличенные надежды, которые сменились разочарованием и упреками в невыполнении обещаний. В самом деле, на начальном этапе развития структурной технологии не было недостатка в оптимистических прогнозах. Структурный подход даже называли “революцией в программировании”. В 1972 г. Дейкстра писал: “Мы научились столь многому, что в течение ближайших лет программирование может превратиться в деятельность, во многом отличающуюся от того, что имеется сегодня, — настолько отличающуюся, что мы должны очень хорошо подготовить себя к ожидающему нас шоку... Семидесятые годы завершатся тем, что мы окажемся способны проектировать и реализовывать такие системы, которые в настоящее время требуют напряжения всех наших способностей, причем расходы на них будут составлять лишь небольшой процент в человекогодах от их сегодняшней стоимости, и, кроме того, эти системы будут фактически свободны от ошибок” [5].

Оправдались ли эти прогнозы? Вот что пишет Н. Брусенцов в 1985 г. (т. е. спустя пять лет после обозначенного Дейкстрой “контрольного срока”): “Ожидавшегося эффекта эти мероприятия пока не дали. Трудозатраты на разработку и сопровождение программ, если и уменьшились, то незначительно. Надежность по-прежнему остается острейшей проблемой. Даже рьяные поборники идеи структурирования признают, что революция не удалась” [6]. В этих условиях некоторые авторы поспешили объявить структурное программирование “отживающим методом” [7].

ПРАВ ЛИ ИГОРЬ ВЕЛЬБИЦКИЙ?

Размышляя о причинах неудачи и стремясь поправить дело, И. Вельбицкий предлагает радикальным образом пересмотреть само понятие “структура программы”. По его мнению, “структура — понятие многомерное. Поэтому отображение этого понятия с помощью линейных текстов (последовательности операторов) сводит практически на нет пре-

имущества структурного подхода. Огромные ассоциативные возможности зрительного аппарата и аппарата мышления человека используются практически вхолостую — для распознавания структурных образов в виде единообразной последовательности символов" [8].

Развивая мысль, Вельбицкий противопоставляет текстовое и визуальное программирование, приходит к заключению, что "на рельсах текстового представления программ" резервы повышения производительности труда в программировании исчерпаны, и делает вывод о необходимости изменить "базис" программирования, т. е. перейти от текстового программирования к визуальному.

В настоящее время визуальное программирование бурно развивается, число его сторонников растет. Тем не менее, уместно спросить: в какой мере предлагаемый Вельбицким пересмотр понятия "структура программы" согласуется с пионерскими взглядами Дейкстры?

ЧЕТЫРЕ ПРИНЦИПА СТРУКТУРИЗАЦИИ БЛОК-СХЕМ, ПРЕДЛОЖЕННЫЕ Э. ДЕЙКСТРОЙ

Попытаемся еще раз заглянуть в темные переулки истории и внимательно перечитаем классический труд Дейкстры "Заметки по структурному программированию". К немалому удивлению, мы обнаружим, что основной тезис о структурных управляющих конструкциях (для обозначения которых названный автор вводит термины "сочленение", "выбор", "повторение" [2]) излагается с прямой апелляцией к визуальному языку блок-схем! Непосредственный анализ первоисточника со всей очевидностью подтверждает: дейкстрианская "структурная революция" началась с того, что Дейкстра, использовав блок-схемы как инструмент анализа структуры программ, предложил наряду с другими важными идеями четыре принципа структуризации блок-схем, которые в дальнейшем были преданы забвению или получили иное, по нашему мнению, слишком вольное толкование. Эти принципы таковы.

1. *Принцип ограничения топологии блок-схем*. Структурная программа должна приводить "к ограничению топологии блок-схем по сравнению с различными блок-схемами, которые могут быть получены, если разрешить проведение стрелок из любого блока в любой другой блок. Отказавшись от большого разнообразия блок-схем и ограничившись ...тремя типами операторов управления, мы следуем тем самым некоей последовательностной дисциплине" [2].
2. *Принцип вертикальной ориентации входов и выходов блок-схемы*. Имея в виду шесть типовых блок-схем *(if-do, if-then-else, case-of, while-do, repeat-until*, а также "действие"), Дейкстра пишет: "Общее свойство всех этих блок-схем состоит в том, что у каждой из них один вход сверху и один выход снизу" [2].
3. *Принцип единой вертикали*. Вход и выход каждой типовой блок-схемы должны лежать на одной вертикали.
4. *Принцип нанизывания типовых блок-схем на единую вертикаль*. При последовательном соединении типовые блок-схемы следует соединять, не допуская изломов соединительных линий, чтобы выход верхней и вход нижней блок-схемы лежали на одной вертикали.

Хотя Дейкстра не дает словесной формулировки третьего и четвертого принципов, они однозначно вытекают из имеющихся в его работе иллюстраций [2]. Чтобы у читателя не осталось сомнений, мы приводим точные копии подлинных рисунков Дейкстры (рис. 131, средняя и левая графа)[1].

Таким образом, можно со всей определенностью утверждать, что две идеи (текстовое и визуальное структурное программирование), подобно близнецам, появились на божий свет одновременно. Однако этих близнецов ожидала разная судьба — судьба принца и нищего.

ПОЧЕМУ НАУЧНОЕ СООБЩЕСТВО НЕ ПРИНЯЛО ВИДЕОСТРУКТУРНУЮ КОНЦЕПЦИЮ Э. ДЕЙКСТРЫ?

Далее события развивались довольно загадочным образом, поскольку вокруг видеоструктурной[2] концепции Дейкстры образовался многолетний заговор молчания.

Неприятность в том, что изложенные выше рекомендации Дейкстры не были приняты во внимание разработчиками национальных и международных стандартов на блок-схемы (ГОСТ 19.701–90, стандарт *ISO* 5807–85 и т. д.). В итоге метод стандартизации, единственный метод, который мог бы улучшить массовую практику вычерчивания блок-схем, не был использован, а идея Дейкстры оказалась наглухо изолированной от

---

[1] Как видно из рис. 131, предложенная Дейкстрой форма иконы “вопрос” (?) и конструкции “переключатель” *(case)*, а также топология соединительных линий при переходе к языку ДРАКОН подверглись модернизации и эргономическим улучшениям. В частности, стремясь реализовать третий принцип (принцип единой вертикали), Дейкстра использует “внутренние” наклонные линии и избыточные изломы, что нарушает эргономическую гармонию рисунка. Чтобы устранить эти погрешности, в язык ДРАКОН внесены необходимые уточнения. Например, в блок-схеме конструкции *repeat-until* Дейкстра использует пять изломов и одну наклонную линию; в дракон-схеме всего два излома, а наклонных отрезков нет вовсе — см. нижний ряд на рис. 131.

[2] В дальнейшем мы будем нередко использовать приставку “видео”, трактуя ее как “относящийся к визуальному программированию” или “относящийся к визуальному представлению знаний”.

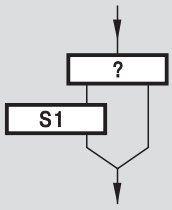
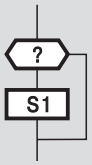
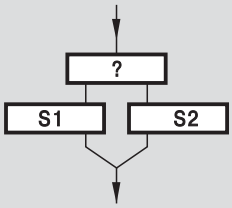
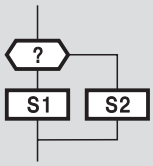
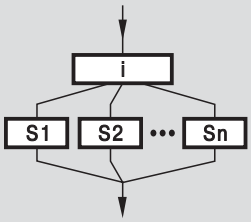
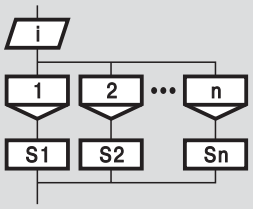
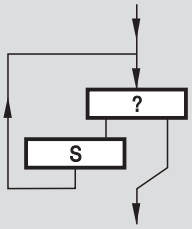
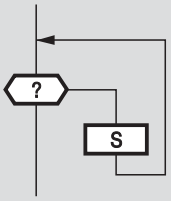
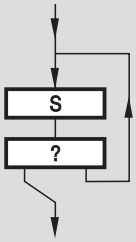
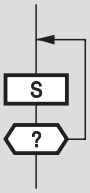
Структурные операторы Дейкстры	Структурные блок-схемы Дейкстры	Дракон-схемы
if ? do S1		
if ? then S1 else S2		
case i of (S1;S2;...Sn)		
while ? do S		
repeat S until ?		

Рис. 131. **Структурные блок-схемы Дейкстры и эквивалентные им визуальные операторы языка ДРАКОН**

реальных блок-схем, используемых в реальной практике программирования. В чем причина этой более чем странной ситуации?

Здесь уместно сделать отступление. Американский историк и философ Томас Кун в книге “Структура научных революций” говорит о том, что в истории науки время от времени появляются особые периоды, когда выдвигаются “несоизмеримые” с прежними взглядами научные идеи. Последние он, следуя Р. Бергману, называет *парадигмами*. История науки — история смены парадигм. Разные парадигмы — это разные образцы мышления ученых. Поэтому сторонникам старой парадигмы зачастую бывает сложно или даже невозможно понять сторонников новой парадигмы (новой системы идей), которая приходит на смену устоявшимся стереотипам научного мышления. По нашему мнению, текстовое и визуальное программирование — это две парадигмы, причем нынешний этап развития программирования есть болезненный процесс ломки прежних взглядов, в ходе которого прежняя *текстовая парадигма* постепенно уступает место новой *визуальной парадигме*. При этом — в соответствии с теорией Куна — многие, хотя и не все видные представители прежней, отживающей парадигмы проявляют своеобразную слепоту при восприятии новой парадигмы, которая в ходе неустанной борьбы идей в конечном итоге утверждает свое господство.

Этот небольшой экскурс в область истории и методологии науки позволяет лучше понять причины поразительного невнимания научного сообщества к изложенным Дейкстрой принципам структуризации блок-схем. Начнем по порядку. Формальная блок-схема — это двумерный чертеж, следовательно, она является инструментом визуального программирования. Отсюда следует, что предложенные Дейкстрой принципы структуризации блок-схем есть не что иное, как исторически первая попытка сформулировать основные (пусть далеко не полные и, возможно, нуждающиеся в улучшении) принципы видеоструктурного программирования.

Однако в 1972 г., в момент публикации работы Дейкстры [2], визуальное программирование как термин, понятие и концепция фактически еще не существовало. Поэтому — вполне естественно — суть концепции Дейкстры была воспринята сквозь призму господствовавших тогда догматов текстового программирования со всеми вытекающими последствиями. Так родилась приписываемая Дейкстре и по праву принадлежащая ему концепция текстового структурного программирования. При этом (что также вполне естественно) в означенное время никто не обратил внимания на тот чрезвычайно важный для нашего исследования факт, что исходная формулировка концепции Дейкстры имела явно выраженную визуальную компоненту — она представляла собой метод структуризации блок-схем, т. е. была описана в терминах видеоструктурного программирования.

Подобное невнимание привело к тому, что авторы стандартов на блок-схемы посчитали, что данная идея их не касается, ибо относится исключительно к тексту программ, и дружно проигнорировали или, скажем мягче, упустили из виду визуальную компоненту структурного метода Дейкстры.

Справедливости ради добавим, что и сам отец-основатель (Э. Дейкстра), обычно весьма настойчивый в продвижении и популяризации своих идей, отнесся к своему видеоструктурному детищу с удивительным безразличием и ни разу не выступил с предложением о закреплении структурной идеи в стандартах на блок-схемы.

ПАРАДОКС СТРУКТУРНОГО ПРОГРАММИРОВАНИЯ

Мы подошли к наиболее интригующему пункту в истории структурного программирования. Чтобы выявить главное звено проблемы, зададим вопрос: являются ли блок-схемы и структурное программирование взаимно исключающими, несовместимыми решениями? В литературе по этому вопросу наблюдается редкое единодушие: да, они несовместимы. Вот несколько отзывов. Блок-схемы "не согласуются со структурным программированием, поскольку в значительной степени ориентированы на использование *goto*" [4]. Они "затемняют особенности программ, созданных по правилам структурного программирования" [9]. "С появлением языков, отвечающих принципам структурного программирования,... блок-схемы стали отмирать" [10].

Парадокс в том, что приведенные высказывания основываются на недоразумении. Чтобы логический дефект стал очевидным, сопоставим две цитаты по методу "очной ставки" (табл. 5). Сравнивая мнение современных авторов с позицией Дейкстры, нетрудно убедиться, что описываемый критиками изъян действительно имеет место, но лишь в том случае, если правила вычерчивания блок-схем игнорируют предложенный Дейкстрой принцип ограничения топологии блок-схем. И наоборот, соблюдение указанного принципа сразу же ликвидирует недостаток.

Таблица 5

Мнение критиков, убежденных в невозможности структуризации блок-схем	**Предложение Э. Дейкстры о структуризации блок-схем**
"Основной недостаток блок-схем заключается в том, что они не приучают к аккуратности при разработке алгоритма: ромб можно поставить в любом месте блок-схемы, а от него повести выходы на какие угодно участки. Так можно быстро превратить программу в запутанный лабиринт, разобраться в котором через некоторое время не сможет даже сам ее автор" [10]	Структуризация блок-схем с неизбежностью приводит "к ограничению топологии блок-схем по сравнению с различными блок-схемами, которые могут быть получены, если разрешить проведение стрелок из любого блока в любой другой блок. Отказавшись от большого разнообразия блок-схем и ограничившись... тремя операторами управления, мы следуем тем самым некоей последовательностной дисциплине" [2]

ПЛОХИЕ БЛОК-СХЕМЫ ИЛИ ПЛОХИЕ СТАНДАРТЫ?

Проведенный анализ позволяет сделать несколько важных замечаний.

! Обвинения, выдвигаемые противниками блок-схем, неправомерны, потому что ставят проблему с ног на голову. Дело не в том, что блок-схемы по своей природе противоречат принципам структуризации, а в том, что при разработке стандартов на блок-схемы указанные принципы не были учтены. На них просто не обратили внимания, поскольку в ту пору — именно в силу парадигмальной

"слепоты" — считалось, что на практике структурное программирование следует применять к текстам программ, а отнюдь не к блок-схемам.

! Чтобы сделать блок-схемы пригодными для структуризации, необходимо ограничить и регламентировать их топологию с учетом видеоструктурных принципов Дейкстры.

! Видеоструктурная концепция Дейкстры — это фундаментальная идея, высказанная более двадцати лет назад и оказавшаяся невостребованной, поскольку она значительно опередила свое время.

! Сегодня созрели благоприятные условия для ее развития. Этому способствуют два обстоятельства. Во-первых, бурное развитие компьютерной графики и визуального программирования. Во-вторых, широкое применение *CASE*-технологий, в которых используется общий для всех участников проекта набор визуальных (графических) языков.

! Предложенные Дейкстрой принципы структуризации блок-схем правильно указывают общее направление развития, однако они нуждаются в доработке, развитии и детализации с учетом последних достижений современной науки, а также опыта (в том числе отрицательного), накопленного при практическом внедрении текстового структурного программирования. В частности, современные блок-схемы должны удовлетворять не только критерию структуризации, но и критериям формализации и эргономизации.

! Именно эту задачу решает шампур-метод, который, с одной стороны, развивает видеоструктурную концепцию Дейкстры, а с другой — учитывает реалии сегодняшнего дня. С помощью шампур-метода разработана новая топология блок-схем (дракон-схемы), регламентация которой произведена на основе принципа когнитивной формализации знаний.

! Современные стандарты на блок-схемы (международный стандарт *ISO* 5807–85, отечественный ГОСТ 19.701–90 и другие национальные стандарты, в том числе американский стандарт *ANSI)*, а также инструкции по их применению следует признать устаревшими, так как они игнорируют принципы структуризации, формализации и эргономизации и объективно содействуют снижению качества соответствующей интеллектуальной продукции.

БЛОК-СХЕМЫ И ТЕОРЕТИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

Наш интерес к структуризации и формализации блок-схем имеет и другую, не менее серьезную причину. Дело в том, что теоретическое программирование, в частности, теория схем программ (схематология) [11] и теория оптимизирующих преобразований программ [12] тесно связаны с теорией графов. По словам А. Ершова, "графы являются основной конструкцией для программиста", так как они "обладают огромной, неисчерпаемой изобразительной силой, соразмерной масштабу задачи программирования" [13].

Для наших целей особенно важным является тот факт, что "графовая форма схем программ" [11] или, что одно и то же, "графовая схема" [14] (короче, граф-схема) аналогична блок-схеме [11, 12]. Более того, как убе-

дительно показал А. Ершов, использовавший блок-схемы в качестве граф-схем [15], разграничение этих понятий является в некотором смысле условным. Различия в начертании (топологии) блок-схем и граф-схем несущественны, а наличие двух терминов оправдывается скорее разными сферами применения, так как термин “блок-схема” используется преимущественно в практическом, а “граф-схема” — в теоретическом программировании.

НОВЫЕ ЦЕЛИ СТАНДАРТИЗАЦИИ БЛОК-СХЕМ

Изложенная выше постановка проблемы нуждается в обобщении и уточнении, что мы и сделаем в форме шести тезисов.

! Существуют три сферы применения блок-схем:
   1) практическое программирование;
   2) теоретическое программирование;
   3) учебная и научно-техническая литература[1].

   Во всех трех сферах используется различная топология блок-схем, унификация отсутствует. Имеющиеся стандарты на блок-схемы относятся только к первой сфере и не касаются двух других. Но главный недостаток состоит в том, что во всех случаях доминируют неформальные блок-схемы.

! Дальнейшее использование неформальных блок-схем во всех случаях следует признать нецелесообразным. С появлением языка SDL и его модификаций началась эра формальных блок-схем. Однако графический синтаксис известных попыток формализации блок-схем не согласуется с видеоструктурными принципами Дейкстры, а также развитым на их основе шампур-методом и по этой причине нуждается в улучшении.

! Желательно разработать единый стандарт, регламентирующий визуальный (но не текстовый) синтаксис формальных блок-схем, который должен применяться во всех трех сферах использования блок-схем, обеспечивая тем самым унификацию блок-схем, граф-схем и органиграмм как полезного в познавательном отношении социокультурного феномена, являющегося универсальным инструментом для визуального представления императивных и технологических знаний любой природы, для решения проблемы автоформализации знаний, описания структуры деятельности и т. д.

! Теоретическое обоснование предполагаемого стандарта должно подразумевать два момента. Во-первых, в основу концепции формализации блок-схем следует положить математический (теоретико-графовый, алгебраический и логический) взгляд на проблему блок-схем. Во-вторых, следует учесть, что в терминах инженерной психологии блок-схемы относятся к классу абстрактных систем отображения информации, которые “служат сенсорной опорой для выполнения человеком логических или математических операций” [16]. Поэтому синтаксис формальных блок-схем следует проектировать с учетом рекомендаций когнитивной эргономики. Поскольку формальные блок-схемы предназначены не только для автоматической обработки в

---

[1] В последнем случае для обозначения блок-схем иногда используют термин “органиграммы”.

компьютере, но и для зрительного восприятия человеком, постольку их синтаксис следует строить с учетом характеристик зрительного анализатора, учитывать системную организацию человеческого зрения (принцип эргономизации).

! В эпоху массовой компьютеризации блок-схемы должны создаваться автоматизированным методом с помощью специального графического редактора, в алгоритмах которого будет “спрятан” упомянутый выше стандарт и который, таким образом, станет де-факто гарантом принудительного соблюдения стандарта всеми пользователями. Ручное вычерчивание блок-схем может использоваться лишь как исключение (для набросков, черновиков и эскизов).

! Правила визуального структурного программирования (или, что одно и то же, визуальный синтаксис языка ДРАКОН) удовлетворяют перечисленным требованиям и, следовательно, могут быть положены в основу проекта упомянутого стандарта, а гарантом его выполнения и единого понимания может стать визуальный ДРАКОН-редактор.

ЧЕМ ОТЛИЧАЮТСЯ БЛОК-СХЕМЫ ОТ ДРАКОН-СХЕМ?

Перейдем теперь к анализу конкретных примеров, которые позволят в наглядной форме выявить различие блок-схем и дракон-схем.

С точки зрения правил языка ДРАКОН, первая блок-схема на рис. 132 (заимствованная из [5]) имеет следующие недостатки.

! Неоправданно большое число изломов линий (в блок-схеме 16 изломов, в дракон-схеме только 5).

! Большое число “паразитных” элементов: 18 стрелок и 4 кружка, которые в дракон-схеме отсутствуют.

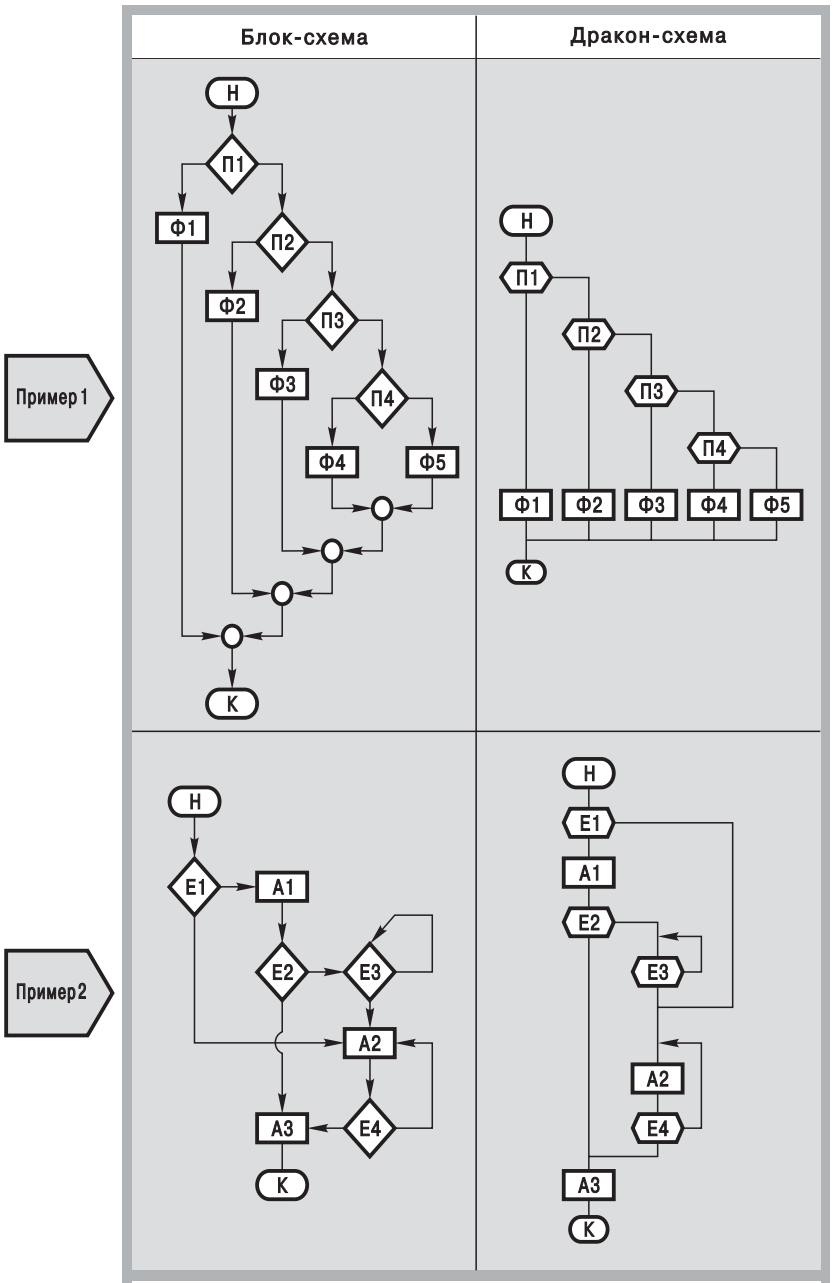


Рис. 132. **Преобразование блок-схем в эквивалентные дракон-схемы**

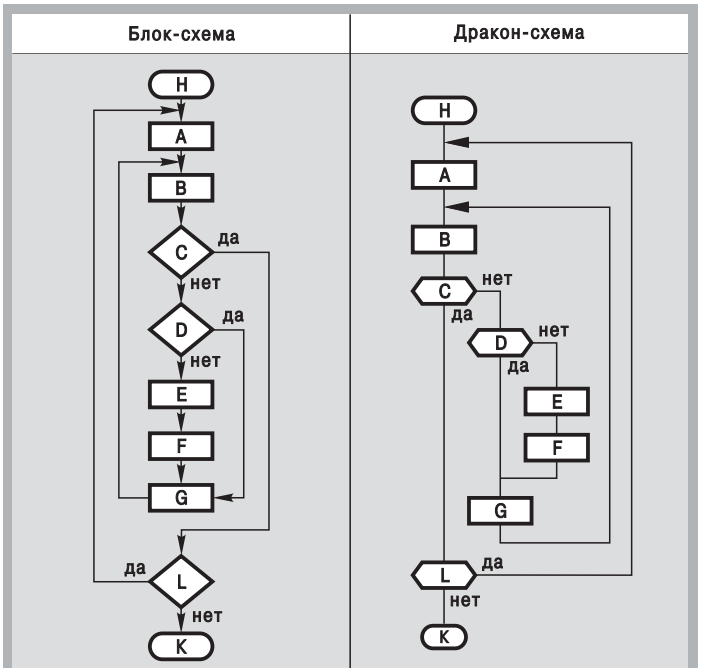


Рис. 132 (окончание)

- Для обозначения развилки используется ромб, который занимает слишком много места и не позволяет поместить внутри необходимое количество удобочитаемого текста, состоящего из строк равной длины.
- Функционально однородные иконы Ф1 — Ф5 в блок-схеме разбросаны по всей площади чертежа, занимая четыре разных горизонтальных уровня; в дракон-схеме они расположены на одном уровне, что служит для читателя наглядной подсказкой об их функциональной однородности.
- Ромбы имеют выход влево, что в дракон-схеме не допускается.
- Икона Ф1 и ее вертикаль расположены слева от шампура (в дракон-схеме это запрещено).
- Ниже икон Ф4, Ф5 имеется четыре уровня горизонтальных линий, которые имеют “паразитный” характер; в дракон-схеме четыре уровня сведены в одну линию.

Вторая блок-схема на рис. 132 (взятая из [17]) имеет следующие изъяны.

- Слева от иконы А2 имеется пересечение линий (в дракон-схеме пересечения запрещены).
- Возле иконы Е3 имеется линия под углом 45º (в дракон-схеме наклонные линии не допускаются).
- Иконы А2, А3 и Е3 имеют более одного входа (в дракон-схеме это запрещено).
- Иконы А1, А2, А3, Е3 имеют входы сбоку (в дракон-схеме вход разрешается только сверху).

! Отсутствует шампур, так как выход иконы “заголовок” и вход иконы “конец” не лежат на одной вертикали.

Предыдущие два примера “плохих” блок-схем были случайным образом взяты из технической литературы. Следующий (третий) пример (см. рис. 132) скопирован из источника [18], где он характеризуется как “стандартная блок-схема ANSI” (Американский национальный институт стандартов). Блок-схема, выполненная по американскому стандарту, также имеет многочисленные дефекты:

! Ниже иконы G имеет место разрыв шампура (нарушено правило, согласно которому один из путей, идущих от входа к выходу, должен проходить по главной вертикали).
! Икона G имеет два входа (в дракон-схеме разрешается только один вход).
! Икона G имеет вход сбоку (в дракон-схеме это запрещено).
! У иконы G выход находится слева (в дракон-схеме он должен быть снизу).
! Две петли обратной связи обычного цикла находятся слева от шампура и закручены по часовой стрелке (в дракон-схеме они расположены справа от шампура и закручены против часовой стрелки).
! Используются неудобные ромбы (в дракон-схеме их заменяют эргономичные иконы “вопрос”).
! Ромб L имеет выход слева (в дракон-схеме он должен быть справа).
! Используются 12 стрелок, из которых 10 — паразитные (в дракон-схеме всего 2 стрелки).
! Имеется один избыточный излом линии (в блок-схеме 9 изломов, в дракон-схеме только 8).

Таким образом, американская блок-схема, как и предыдущие примеры, по всем параметрам проигрывает дракон-схеме.

В ЧЕМ СХОДСТВО ВИЗУАЛЬНОГО И ТЕКСТОВОГО СТРУКТУРНОГО ПРОГРАММИРОВАНИЯ?

Можно предложить девять правил, устанавливающих соответствие между понятиями шампур-метода и классического структурного кодирования.

1. Понятие “функциональный атом” (рис. 122) эквивалентно понятиям “функциональный узел”, “функция” [3] и “узел обработки” [5], используемым в литературе по структурному программированию.
2. Макроикона “развилка” (рис. 122), в которую произведен ввод функционального атома в левую или обе критические точки, эквивалентна конструкциям *if-then u if-then-else* соответственно [3] — см. также две верхние графы на рис. 131.
3. Макроикона “обычный цикл” (рис. 122), в которую произведен ввод функционального атома в одну или обе критические точки, эквивалентна конструкциям *do-until*, *while-do*, *do-while-do* соответственно [3] — см. две нижние графы на рис. 131.
4. Непустая макроикона “переключатель” (рис. 122), в которую введено нужное число икон “вариант” с помощью операции “добавление варианта”, эквивалентна конструкции *case* [2] — см. рис. 131.
5. Непустая макроикона “цикл ДЛЯ” (рис. 122) эквивалентна циклу *for* языка СИ.

6. Непустая макроикона “переключающий цикл” (рис. 122), в которую введено нужное число икон “вариант”, эквивалентна циклу с вложенным оператором *case*, используемым, например, при построении рекурсивных структурированных программ по методу Ашкрофта—Манны [3].
7. Шампур-блок — видеоструктурный эквивалент понятия “простая программа” [3].
8. Атом — общее понятие для основных составляющих блоков, необходимых для построения программы согласно структурной теореме Бома и Джакопини [1]. Оно охватывает также все элементарные структуры структурного кодирования, кроме последовательности [3] (последняя в шампур-методе считается неэлементарной структурой).
9. Операция “ввод атома” позволяет смоделировать две операции: во-первых, построить последовательность из двух и более атомов, во-вторых, методом заполнения критических точек осуществить многократное вложение составных операторов друг в друга, благодаря чему единственный функциональный блок “постепенно раскрывается в сложную структуру основных элементов”.

Воспользуемся термином “базовые операции” для обозначения операций “ввод атома”, “добавление варианта” и “боковое присоединение”. Применяя к заготовке-примитиву базовые операции любое число раз, мы всегда получим структурную дракон-схему в традиционном смысле слова.

В ЧЕМ РАЗЛИЧИЕ ВИЗУАЛЬНОГО И ТЕКСТОВОГО СТРУКТУРНОГО ПРОГРАММИРОВАНИЯ?

Структурные, лианные и адресные блоки

Шампур-метод позволяет строить как структурные, так и метаструктурные (лианные) программы. Выше мы выяснили, что базовые операции моделируют классическое структурное кодирование. Чтобы смоделировать полезные функции оператора *goto*, в шампур-методе предусмотрены операции “пересадка лианы” и “заземление лианы”.

Введем три понятия.

Структурный блок — шампур-блок, построенный только с помощью базовых операций, без использования действий с лианой.

Лианный блок — шампур-блок, полученный из структурного блока с помощью операции “пересадка лианы”.

Адресный блок — результат преобразования структурного блока с помощью операции “заземление лианы” и, возможно, “пересадка лианы”. Адресный блок используется только в силуэте и представляет собой многоадресную ветку (или ее нижнюю часть). Он имеет один вход и не менее двух выходов, присоединенных к нижней горизонтальной линии силуэта через иконы “адрес”.

В примитиве могут использоваться только структурные и лианные блоки (рис. 133, 134), в силуэте — все три типа блоков: структурные, лианные и адресные (рис. 135, 136)[1].

Операции с лианой и оператор goto

Операции с лианой моделируют все без исключения функции заменителей *goto* (например, дополнительный выход из цикла), а также некоторые функции *goto*, которые невозможно реализовать с помощью заменителей. Однако они не приводят к хаосу, вызванному бесконтрольным использованием *goto*. С эргономической точки зрения, действия с лианой на порядок эффективнее и удобнее, чем *goto* и заменители; с другой стороны, они весьма эффективно корректируют недостатки классического структурного программирования.

Чтобы убедиться в этом на примере, вернемся к анализу рис. 27. В гл. 7 мы рассмотрели эргономические преимущества схемы на рис. 27*б* по сравнению с рис. 27*а*. Было показано, что улучшение эргономичности достигнуто за счет использования равносильных преобразований алгоритмов: вертикального и горизонтального объединения. При этом за кадром осталась важная проблема — проблема синтаксиса: как построить указанные схемы? Теперь мы имеем возможность осветить этот вопрос. Схема на рис. 27*а* представляет собой структурный блок, полученный с помощью операции “ввод атома”. В отличие от нее схема на рис. 27*б* — это лианный блок, построенный методом пересадки лианы.

Уместно вспомнить предостережение Г. Майерса: “Правила структурного программирования часто предписывают повторять одинаковые фрагменты программы в разных участках модуля, чтобы избавиться от употребления операторов *goto*. В этом случае лекарство хуже болезни; дублирование резко увеличивает возможность внесения ошибок при изменении модуля в будущем” [4]. Как видно из рис. 26 и 27, пересадка лианы позволяет элегантно и без потерь решить эту непростую проблему, одновременно улучшая наглядность и понимаемость программы, обеспечивая более эффективное топологическое упорядочивание маршрутов.

---

[1] Заметим, что силуэт на рис. 136 можно интерпретировать как детерминированный конечный автомат [19], показанный на рис. 137 (входной алфавит и переходная функция автомата не показаны).

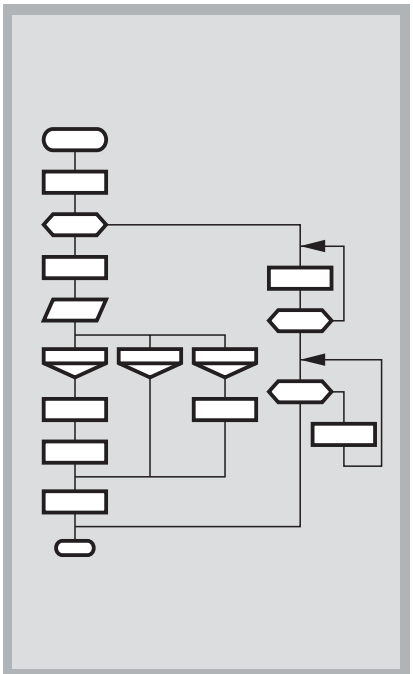


Рис. 133. **Примитив, построенный из структурных блоков**

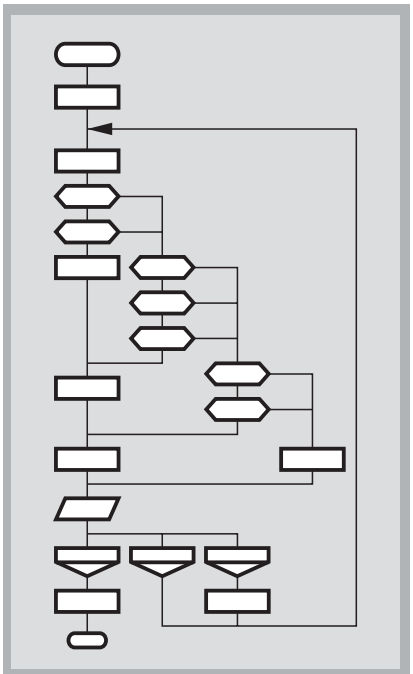


Рис. 134. **Примитив, построенный из структурных и лианных блоков**

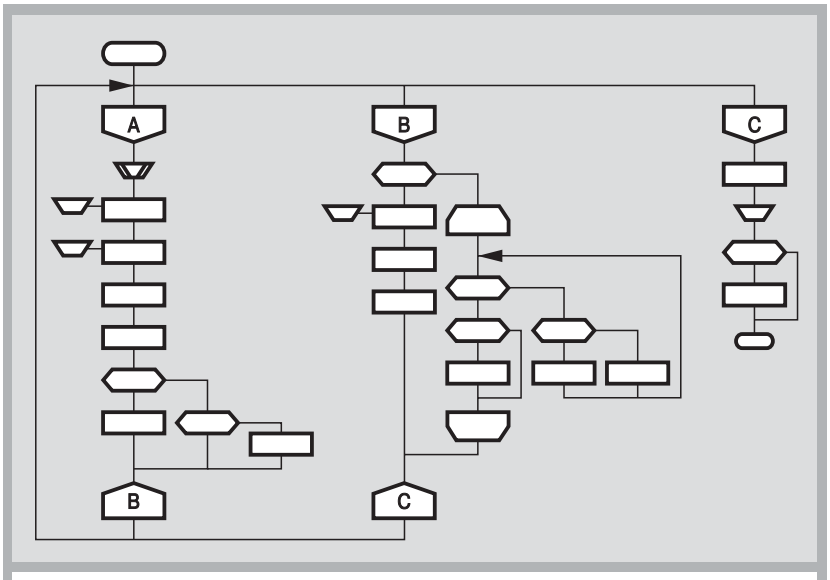


Рис. 135. **Силуэт, построенный из структурных блоков**

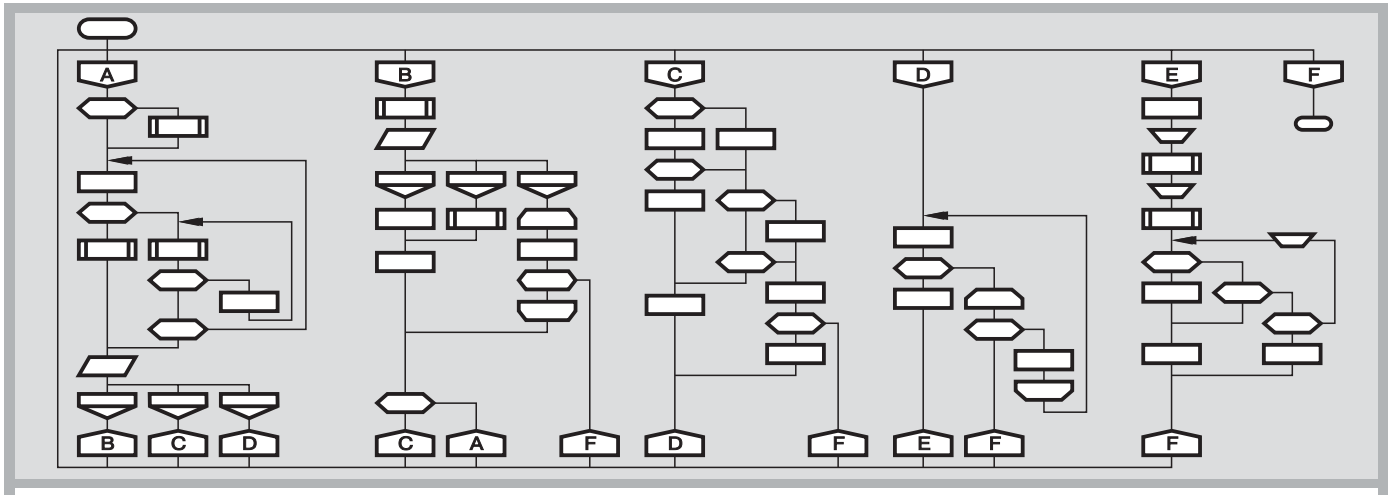


Рис. 136. **Силуэт, построенный из структурных, лианных и адресных блоков**

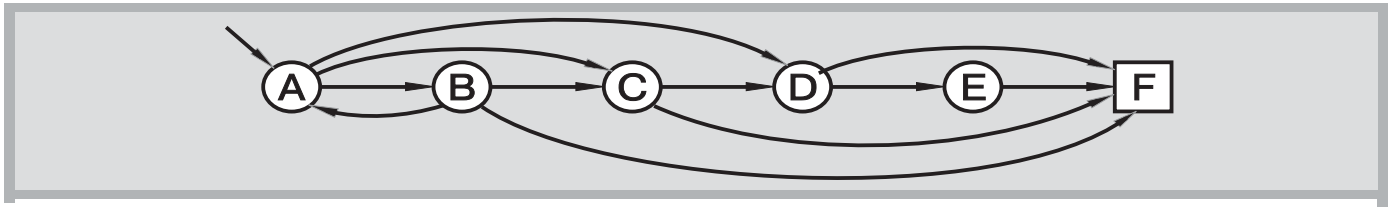


Рис. 137. **Детерминированный конечный автомат, соответствующий силуэту на рис. 136**

Пересадка лианы узаконивает лишь некоторые, отнюдь не любые передачи управления, поскольку определение данной операции (см. гл. 15, тезис 28) содержит ряд ограничений. Запрет на образование нового цикла вызван тем, что переход на оператор, расположенный выше (раньше) в тексте программы, считается "наихудшим применением оператора *goto*" [4]. Указанный запрет вводится, чтобы выполнить требование: использовать *goto* только для передачи управления вперед по программе, "которое некоторыми организациями принимается в качестве компромиссной версии структурного программирования".

Запрет на образование второго входа в цикл соответствует требованию структурного кодирования, согласно которому цикл, как и любая простая программа, должен иметь не более одного входа. Лишь третий запрет является оригинальной особенностью шампур-метода: он запрещает передачи управления, изображение которых с помощью лианы ведет к пересечению линий. Таким образом, пересадка лианы разрешает только те переходы вниз по ДРАКОН-программе, которые образуют связи с валентными точками и изображаются легко прослеживаемыми маршрутами, т. е. непересекающимися линиями.

Является ли текстовое структурное программирование формальным методом?

Ортодоксальный метод Дейкстры, полностью запрещающий *goto* и заменители (см. с. 238, табл. 4, вариант 1), безусловно является строгим формальным методом. К сожалению, он полезен лишь как интересная теоретическая идея, которая, как показал всемирный опыт, в чистом виде оказалась непригодной для массового использования.

По мнению специалистов, "правила структурного программирования верны в 95%. Но остаются злополучные 5%" [20]. Чтобы поправить дело, решить проблему пяти процентов и создать метод, пригодный для широкой практики, пришлось пойти на компромисс и дать добро на использование заменителей и так называемое "ограниченное применение *goto*" (см. табл. 4, варианты 2—4). Благодаря этому проблема массовой практики программирования была решена. Но какой ценой? Ценой отказа от строгого формализма.

Это нетрудно показать. Например, авторы учебника языка СИ советуют осторожно и редко применять заменители *break* и *continue*, "поскольку слишком частое их использование ухудшает читаемость программы, увеличивает вероятность ошибок и затрудняет ее модификацию" [21]. Далее они пишут: избегайте использовать *goto*, ибо это "чрезвычайно плохое" средство, которое следует применять "как можно реже или не применять совсем" [21].

Отсюда вытекает, что решение о списке случаев, когда нужно или не нужно употреблять перечисленные операторы, предоставляется программисту, который будет действовать, возможно, из лучших побуждений, но в соответствии со своим личным опытом и пристрастиями. Таким образом, о строгой формализации в данном случае речь идти не может.

Следовательно, мы вправе сделать существенное замечание. Структурное кодирование, используемое в широкой практике программирования, не является формальным методом, так как к формальным прави-

лам Дейкстры пришлось добавить неформальные правила, касающиеся *goto* и заменителей.

В шампур-методе аналогом *goto* и заменителей служат формальные операции “пересадка лианы” и “заземление лианы”, на использование которых не накладывается никаких неформальных ограничений.

Тем самым мы приходим к важному выводу. В отличие от текстового структурного программирования, обладающего лишь частичной формализацией, правила визуального структурного программирования формализованы на 100%, что подтверждается тем фактом, что они реализованы в алгоритмах ДРАКОН-редактора.

ПОЧЕМУ САМОЛЕТ НЕ МАШЕТ КРЫЛЬЯМИ?

Говоря о будущем структурного программирования, необходимо осознать, что текстовое и визуальное программирование опирается на разные системы понятий, которые по-разному расчленяют действительность. Поэтому видеоструктурное программирование нельзя рассматривать как результат механического перевода устоявшихся понятий классического структурного программирования на новый язык.

Поясним сказанное. При визуальном структурном программировании программист работает только с чертежом программы, не обращаясь к ее тексту, подобно тому, как программист, работающий, скажем, на Паскале, не обращается к ассемблеру и машинному коду — они для него просто не существуют! Это значит, что столь тщательно обоснованная Дейкстрой коллекция ключевых слов структурного кодирования *(if, then, else, case, of, while, do, repeat, until, begin, end* [2]) при переходе к визуальному программированию становится ненужной — для программиста она просто перестает существовать! В равной степени становятся лишними и отмирают и другие ключевые слова, используемые оппонентами Дейкстры в различных вариантах расширения структурного кодирования: *goto*, *continue*, *break*, *exit* и т. д.

Следует специально подчеркнуть: поскольку в визуальном варианте структурного программирования ключевое слово *goto* не используется, теряют смысл и все споры относительно законности или незаконности, опасности или безопасности его применения, а также обширная литература, посвященная обсуждению этого, некогда казавшегося столь актуальным вопроса.

Предвижу возражения: хотя названные ключевые слова действительно исчезают, однако выражаемые ими понятия сохраняют силу и для визуального программирования. Например, две видеопрограммы на рис. 27 можно построить с помощью понятий *if-then-else* и *goto*. Данное возражение нельзя принять, поскольку произошла подмена предмета обсуждения. С помощью указанных понятий можно построить не видеопрограммы, а текстовые программы, эквивалентные видеопрограммам на рис. 27. Что касается собственно видеопрограмм, то они, будучи “выполнимой графикой”, строятся из “выполняемых” икон и “выполняемых” соединительных линий. Причем — подчеркнем еще раз — в процессе построения (которое реализуется с помощью ДРАКОН-редактора) пользователь не применяет понятия *if-then-else* и *goto*, ибо в этом нет никакой необходимости.

Нельзя путать задачу и систему понятий, на которую опирается метод ее решения. В обоих случаях — и при текстовом, и при визуальном структурном кодировании — ставится одна и та же задача: улучшить понимаемость программ и обеспечить более эффективный интеллектуальный контроль за передачами управления. Однако система понятий коренным образом меняется: ту функцию, которую в текстовой программе выполняют ключевые слова, в видеопрограмме реализуют совершенно другие понятия: иконы, макроиконы, соединительные линии, шампур, главная вертикаль шампур-блока, лиана, атом, пересадка лианы, запрет пересечения линий и т. д.

Очевидно, что использование понятий, выражаемых ключевыми словами классического структурного программирования, не является самоцелью, а служит лишь для того, чтобы "делать наши программы разумными, понятными и разумно управляемыми" [22]. Указанные понятия решают эту задачу не во всех случаях, а только в рамках текстового программирования. При переходе к визуальному программированию задача решается по-другому, с помощью другого набора понятий. Именно отказ от старого набора понятий и замена его на новый и позволяет добиться новой постановки задачи и более эффективного ее решения. Поэтому высказываемое иногда критическое замечание: "недостаток шампур-метода в том, что он не удовлетворяет требованиям структурного программирования" является логически неправомерным. Нельзя упрекать самолет за то, что он не машет крыльями.

ВЫВОДЫ

1. Визуальное структурное программирование (шампур-метод) и текстовое структурное программирование несоизмеримы (в куновском смысле слова), они опираются на разные системы понятий, которые по-разному расчленяют действительность.
2. Текстовое структурное программирование решило стоявшие перед ним исторические задачи, исчерпало свои эвристические возможности и, выполнив свою миссию, потеряло актуальность. В настоящее время точкой роста научного знания является визуальное структурное программирование.
3. При использовании шампур-метода набор ключевых слов классического структурного программирования становится ненужным. Благодаря этому создаются предпосылки, которые в будущем, возможно, позволят исключить ключевые слова и тем самым устранить путающий всех разнобой ключевых слов и структурных конструкций в разных языках программирования.
4. В отличие от текстового структурного программирования шампур-метод является полностью формальным.
5. По эргономическим показателям визуальное структурное программирование существенно превосходит свой текстовый аналог.
6. Широко распространенное мнение о несовместимости блок-схем и структурного программирования является мифом, нелепой ошибкой, основанной на невнимательном прочтении классической работы Э. Дейкстры "Заметки по структурному программированию".

7. Дальнейшее использование неструктурных, неформальных и неэргономичных блок-схем во всех случаях следует признать нецелесообразным.
8. Существующая литература по блок-схемам, включая международные и национальные стандарты, на 99% устарела.
9. Современные стандарты на блок-схемы объективно содействуют снижению качества соответствующей интеллектуальной продукции. Указанные стандарты игнорируют три важнейших принципа: структуризации, формализации и эргономизации.
10. Актуальной задачей является разработка новой системы международных и национальных стандартов на блок-схемы, свободных от перечисленных недостатков. В основу проекта новых стандартов целесообразно положить изложенные в этой книге правила визуального структурного программирования. Дракон-схемы наследуют все или почти все достоинства блок-схем и устраняют их недостатки.

Глава 17

ИСЧИСЛЕНИЕ ИКОН И ПОПЫТКА ПРЕДСКАЗАТЬ БУДУЩЕЕ

Зрительные образы — плоть и кровь самого мышления.

Рудольф Арнхейм

ВИЗУАЛЬНОЕ ЛОГИЧЕСКОЕ ИСЧИСЛЕНИЕ

Включим еще один “боковой прожектор” и попытаемся взглянуть на визуальный синтаксис языка ДРАКОН с позиций математической логики. Нашему взору откроется необычная картина. Оказывается, любая абстрактная дракон-схема представляет собой теорему, которая строго выводится (доказывается) из двух аксиом, каковыми являются заготовка-примитив и заготовка-силуэт.

— Какие же это аксиомы? — вправе удивиться читатель. — Ведь это просто картинки-слепыши! А дракон-схемы вовсе не похожи на теоремы! Кто и зачем их должен доказывать? Наверно, это шутка или метафора.

— Вовсе нет, отнюдь не метафора. Ниже будет показано, что визуальный синтаксис ДРАКОНА построен как логическое исчисление (назовем его “исчисление икон”). Данное исчисление можно рассматривать как раздел *визуальной математической логики*. Последнее понятие не является традиционным. Математическая логика и ее основные понятия (исчисление, логический вывод и т. д.) сформировались в рамках текстовой парадигмы. В данной главе, по-видимому, впервые вводятся визуальные аналоги этих понятий и на их основе строится исчисление икон.

ОБЩЕИЗВЕСТНЫЕ СВЕДЕНИЯ О МАТЕМАТИЧЕСКОЙ ЛОГИКЕ

Принципиальным достижением математической логики является разработка современного аксиоматического метода, который характеризуется тремя чертами:

- явной формулировкой исходных положений (аксиом) развиваемой теории (формальной системы);
- явной формулировкой правил логического вывода, с помощью которых из аксиом выводятся теоремы теории;
- использованием формальных языков для изложения теорем рассматриваемой теории [1].

Основным объектом изучения в математической логике являются логические исчисления. В понятие исчисления входят такие основные компоненты, как:

а) формальный язык, который задается с помощью алфавита и синтаксиса,
б) аксиомы,
в) правила вывода [1].

Таким образом, исчисление позволяет, зная аксиомы и правила вывода, получить (т. е. вывести, доказать) все теоремы теории, причем теоремы, как и аксиомы, записываются только на формальном языке.

Напомним, что в рамках математической логики три термина: логическое исчисление, формальная система и теория можно рассматривать как синонимы. Следовательно, теоремы исчисления, теоремы формальной системы и теоремы теории — это одно и то же.

ОБ ОДНОМ РАСПРОСТРАНЕННОМ ЗАБЛУЖДЕНИИ

Существуют два подхода к формализации человеческих знаний: визуальный (графический, изобразительный) и текстовый. С этой проблемой связано любопытное противоречие. С одной стороны, преимущество графики перед текстом для человеческого восприятия считается общепризнанным, так как человеческий мозг в основном ориентирован на визуальное восприятие и люди получают информацию при рассмотрении графических образов быстрее, чем при чтении текста. И. Вельбицкий совершенно справедливо указывает: “Текст — наиболее общая и наименее информативная в смысле наглядности и скорости восприятия форма представления информации”, а чертеж — “наиболее развитая интегрированная форма представления знаний”.

С другой стороны, теоретическая разработка принципов визуальной формализации знаний все еще не развернута в должной мере. Причину отставания следует искать в истории науки, в частности в особенностях развития математики и логики.

В этих дисциплинах с давних пор (иногда явно, чаще неявно) предполагалось, что результаты математической и логической формализации знаний в подавляющем большинстве случаев должны иметь форму текста (а не изображения). Например, Стефен Клини пишет: «Будучи формализованной, теория по своей структуре является уже не системой осмысленных предложений, а системой фраз, рассматриваемых как последовательность слов, которые, в свою очередь, являются последовательностями букв... В символическом языке символы будут обычно соответствовать целым словам, а не буквам, а последовательности символов, соответствующие фразам, будут называться “формулами”... Теория доказательств... предполагает... построение произвольно длинных последовательностей символов» [2].

Из этих рассуждений видно, что Клини (как и многие другие авторы) ставит в центр исследования проблему *текстовой* формализации и полностью упускает из виду всю совокупность проблем, связанных с *визуальной* формализацией.

Анализ литературы, посвященной данной теме, показывает, что большинство ученых исходит из неявного предположения, что научное знание — это прежде всего “текстовое” знание, что наиболее адекватной (или даже единственно возможной) формой для представле-

ния результатов научного исследования является последовательность формализованных и неформализованных фраз, т. е. текст (а отнюдь не визуальные образы). В основе этого предположения лежит ошибочная точка зрения, которую можно охарактеризовать как “принцип абсолютизации текста”.

Принцип абсолютизации текста

Суть его можно выразить, например, в форме следующих рассуждений.

> Прогресс науки обеспечивается успехами логико-математической формализации и разработкой новых научных понятий и принципов, а не усовершенствованием рисунков. Формулы и слова выражают сущность научной мысли, рисунки — это всего лишь иллюстрации к научному тексту, они облегчают понимание уже готовой, сформированной научной мысли, но не участвуют в ее формировании. Короче говоря, язык науки — это формулы и предложения, но никак не картинки. В науке есть суть, сердцевина, от которой зависит успех научного творчества и получение новых научных результатов (она выражается логико-математическими формализмами, научными понятиями и суждениями, выраженными в словах). И есть вспомогательные задачи (обучение новичков, обмен информацией между учеными) — здесь-то и помогают картинки, облегчая взаимопонимание. Кроме того, рисунки имеют необязательную, свободную и нестрогую форму, их невозможно формализовать. Поэтому формализация научного знания несовместима с использованием рисунков. Таким образом, рисунки есть нечто внешнее по отношению к науке. Совершенствование языка рисунков и научный прогресс — разные вещи, они не связаны между собой.

Существует ряд работ, косвенным образом доказывающих, что принцип абсолютизации текста является ошибочным и вредным [3, 4 и др.]. Сегодня все больше ученых приходит к выводу, что визуальную формализацию знаний нельзя рассматривать как нечто второстепенное для научного познания, поскольку она входит в саму ткань мысленного процесса ученого и “может опосредовать самые глубинные, творческие шаги научного познания” [3].

Вместе с тем в математической логике визуальные методы, насколько нам известно, пока еще не нашли широкого применения. Иными словами, математическая логика по сей день остается оплотом текстового мышления и текстовых методов формализации знаний. Это обстоятельство играет отрицательную роль, мешая поставить последнюю точку в доказательстве ошибочности “принципа абсолютизации текста”.

Далее мы попытаемся на частном примере исчисления икон продемонстрировать принципиальную возможность визуализации по крайней мере некоторых разделов или, скажем аккуратнее, вопросов математической логики.

ВИЗУАЛИЗАЦИЯ ПОНЯТИЙ МАТЕМАТИЧЕСКОЙ ЛОГИКИ

Нам понадобится определение двух понятий: *визуальный логический вывод* (видеовывод) и *визуальное логическое исчисление* (видеоисчисление). Чтобы облегчить изучение материала, используем уже знакомый читателю метод очной ставки, помещая в левой графе табл. 6 хорошо известное "текстовое" понятие, а в правой — определение нового, "визуального" понятия.

Таблица 6

Определение понятия "логический вывод" [5]	**Определение понятия "видеовывод" (визуальный логический вывод)**
Вывод в исчислении V есть последовательность $C_1, \ldots, C_n$ формул, такая, что для любого i формула C_i есть либо аксиома исчисления V, либо непосредственное следствие предыдущих формул по одному из правил вывода. Формула C_n называется теоремой исчисления V, если существует вывод в V, в котором последней формулой является C_n	Видеовывод в видеоисчислении V есть последовательность $C_1, \ldots C_n$ видеоформул, такая, что для любого i видеоформула C_i есть либо видеоаксиома видеоисчисления V, либо непосредственное следствие предыдущих видеоформул по одному из правил видеовывода. Видеоформула C_n называется видеотеоремой видеоисчисления V, если существует видеовывод в V, в котором последней видеоформулой является C_n

Нетрудно заметить, что новое определение (справа) почти дословно совпадает с классическим (слева); разница состоит лишь в добавлении приставки "видео".

Таблица 7

Определение понятия “логическое исчисление” [6]	**Определение понятия “видеоисчисление” (визуальное логическое исчисление)**
Логическое исчисление может быть представлено как формальная система в виде четверки $V = <$ И, S_0, А, $F>$ где И — множество базовых элементов (букв алфавита); S_0 — множество синтаксических правил, на основе которых из букв строятся правильно построенные формулы; А — множество правильно построенных формул, элементы которого называются аксиомами; F — правила вывода, которые из множества A позволяют получать новые правильно построенные формулы — теоремы	Видеоисчисление может быть представлено как формальная система в виде четверки $V = <$ И, S_0, А, $F>$ где И — множество икон (букв визуального алфавита); S_0 — множество правил визуального синтаксиса, на основе которых из икон строятся правильно построенные видеоформулы; А — множество правильно построенных видеоформул, элементы которого называются видеоаксиомами; F — правила видеовывода, которые из множества A позволяют получать новые правильно построенные видеоформулы — видеотеоремы. (Множество теорем обозначим через Т.)

Развивая этот подход и опираясь на “текстовое” определение логического исчисления, можно по аналогии ввести понятие “видеоисчисление” (табл. 7).

ИСЧИСЛЕНИЕ ИКОН

Итак, мы определили нужные понятия визуальной математической логики. С их помощью можно построить исчисление икон.

- Множество икон И (букв визуального алфавита) задано тезисом 1 (см. гл. 15) и показано на рис. 1.
- Множество S_o правил визуального синтаксиса описано в гл. 15 в тезисах 2—37.
- Множество А визуальных аксиом включает всего два элемента: заготовку-примитив и заготовку-силуэт (рис. 115). Далее будем называть их *аксиома-примитив* и *аксиома-силуэт*.
- Множество Т, охватывающее все видеотеоремы исчисления V, есть не что иное как множество абстрактных дракон-схем. Заметим, что множество Т не включает аксиомы, так как последние содержат незаполненные критические точки и, следовательно, эквивалентны пустым операторам. Множество Т распадается на две части: множество примитивов T_1 и множество силуэтов T_2.

! Множество F правил видеовывода также делится на две части F_1 и F_2. Множество F_1 позволяет выводить все теоремы-примитивы, принадлежащие множеству T_1, из единственной аксиомы-примитива. Оно содержит пять правил вывода: ввод атома, добавление варианта, пересадка лианы, боковое присоединение, удаление конца примитива. Эти правила описаны в тезисах 10, 21, 28, 30, 31, 34 гл. 15.

! Множество F_2 дает возможность выводить все теоремы-силуэты множества T_2 из единственной аксиомы-силуэта. Оно содержит восемь правил вывода: ввод атома, добавление варианта, добавление ветки, пересадка лианы, заземление лианы, боковое присоединение, удаление последней ветки, дополнительный вход. Правила вывода для силуэта описаны в тезисах 10, 21, 28—33, 35 гл. 15.

На этом построение исчисления икон заканчивается.

Известно, что изучение исчислений составляет синтаксическую часть математической логики. Кроме того, последняя занимается семантическим изучением формальных языков, причем основным понятием семантики служит понятие истинности [1].

В исчислении икон семантика тривиальна. Различные видеоформулы (блок-схемы) могут быть истинными или ложными. Видеоформула называется *истинной*, если она — либо аксиома, либо выводится из аксиом с помощью правил вывода (т. е. является теоремой), и *ложной* в противном случае. Таким образом, все правильно построенные абстрактные дракон-схемы (теоремы) истинны. И наоборот, неправильно построенные схемы, не удовлетворяющие визуальным правилам языка ДРАКОН, считаются ложными. Примеры ложных схем показаны на рис. 131 и 132 в левой графе.

ЕЩЕ РАЗ О ШАМПУР-МЕТОДЕ

Ранее мы определили шампур-метод как теорию визуального структурного программирования. В этой главе появилась возможность значительно обогатить это понятие и рассматривать его как исчисление икон, включая вопросы интерпретации последнего.

Чтобы подчеркнуть теоретический характер шампур-метода, целесообразно слегка изменить терминологию. В частности, использование названия ДРАКОН, связанного с практической разработкой конкретных языков программирования, для теоретических целей представляется неуместным. Поэтому произведем замену терминов.

Шампур-схема — абстрактная дракон-схема. Подчеркнем, что шампур-схема по определению является абстрактной, т. е. полностью лишенной текста.

Шампур-язык — язык шампур-схем. Для шампур-языка задан только визуальный синтаксис, текстовый синтаксис не определен.

ШАМПУР-СХЕМА КАК АБСТРАКТНАЯ МОДЕЛЬ ПРОГРАММЫ

Уже говорилось, что для видеопрограммирования характерно "расщепление синтаксиса". Синтаксис S распадается на визуальный синтаксис S_0, определяющий правила построения шампур-схем, и текстовый синтаксис S_1, задающий алфавит текстоэлементов и правила записи текстовых операторов внутри икон. Исходя из этого, можно сказать, что *шампур-программа* B состоит из двух частей: B_0 и B_1, где B_0 — шампур-схема с синтаксисом S_0; B_1 — текстовая часть программы, т. е. совокупный текст, находящийся во всех иконах программы, определяемый синтаксисом S_1.

Бросается в глаза несомненное сходство между шампур-схемами и схемами программ. Заметив эту аналогию и повторяя — с некоторыми, почти очевидными изменениями — общую канву рассуждений, принятую в схематологии [7], можно сделать вывод, что шампур-схема B_0 описывает уже не одну программу, а целый класс программ, т. е. является полипрограммой, а шампур-язык служит мультиязыком — языком полипрограммирования [8].

Класс шампур-схем является подклассом класса крупноблочных схем, по степени абстрактности занимающий промежуточное положение где-то между схемами Мартынюка и стандартными схемами. Связь между шампур-схемами и схемами программ имеет фундаментальный характер и порождает ряд интересных проблем, связанных, в частности, с тем, что "задача эффективизации транслируемых программ перерастает в задачу автоматизации конструирования качественных программ" [8].

С точки зрения теории видеопрограммирования, граф-схемы, используемые в (текстовом) теоретическом программировании, обладают недостатком — как и обычные блок-схемы прикладного программирования, они являются неформальными. Хотя в работах А. Ершова сделан определенный шаг к формализации граф-схем, однако предложенное им решение [9] нельзя признать удовлетворительным, ибо использованный Ершовым визуальный синтаксис граф-схем не позволяет получить однозначную, строго детерминированную визуальную конфигурацию (топологию) граф-схем и, следовательно, не дает единственного решения визуальной задачи.

Впрочем, Ершов и не ставил перед собой подобных задач. Однако для наших целей строгая формализация визуального синтаксиса блок-схем (включая граф-схемы) играет принципиальную роль.

ПРЕОБРАЗОВАНИЕ ШАМПУР-СХЕМЫ В ШАМПУР-ПРОГРАММУ

Подчеркнем еще раз, что построенный нами язык (шампур-язык) — это не язык программирования, а язык крупноблочных схем программ, т. е. язык полипрограммирования. Однако его можно легко превратить в язык программирования, причем сделать это многими способами. Для этого необходимо дополнительно задать текстовый синтаксис S_1 и

семантику Q_1 текстовых операторов, помещаемых в иконах шампур-схемы. Например, если взять текстовый синтаксис и семантику, соответствующие языку ПАСКАЛЬ, получим язык визуального программирования, который можно назвать “шампур-паскаль”. Аналогично можно построить языки шампур-бейсик, шампур-си и т. д.

Используя терминологию схематологии, можно сказать, что шампур-программа есть интерпретированная шампур-схема, однако понятие интерпретации в нашем случае заметно отличается от классического [7]. Детальное рассмотрение вопроса выходит за рамки книги, ограничимся лишь кратким замечанием. Чтобы задать интерпретацию шампур-схемы и превратить ее в шампур-программу, необходимо, во-первых, доопределить шампур-язык и превратить его в язык программирования, описав синтаксис S_1 и семантику Q_1 текстовых операторов. Во-вторых, следует указать конкретные текстовые операторы, записанные в соответствии с синтаксисом S_1 и размещенные в иконах шампур-схемы B_0. Тем самым будет задана текстовая часть B_1 шампур-программы B. Таким образом, интерпретация шампур-схемы определяется как тройка $< S_1, Q_1, B_1 >$.

Отсюда вытекает следующее очевидное замечание. Поскольку шампур-язык есть абстрактная модель любого императивного языка программирования (импер-языка), постольку импер-язык есть интерпретированный шампур-язык. При этом интерпретация шампур-языка, превращающая его в конкретный импер-язык, определяется как пара $< S_1, Q_1 >$.

ШАМПУР-МЕТОД И ДОКАЗАТЕЛЬСТВО ПРАВИЛЬНОСТИ ПРОГРАММ

Согласно Р. Андерсону, “целью многих исследований в области доказательства правильности программ является... механизация таких доказательств” [10]. Д. Грис указывает, что “доказательство должно опережать построение программы” [11]. Объединив оба требования, получим, что автоматическое доказательство правильности должно опережать построение программы. Нетрудно убедиться, что шампур-метод обеспечивает частичное выполнение этого требования. В самом деле, в начале главы было показано, что любая правильно построенная шампур-схема является строго доказанной теоремой. В алгоритмах ДРАКОН-редактора закодировано исчисление икон, поэтому любая шампур-схема, построенная с его помощью, является истинной, т. е. правильно построенной. Это означает, что ДРАКОН-редактор осуществляет 100%-е автоматическое доказательство правильности шампур-схем, гарантируя принципиальную невозможность ошибок визуального синтаксиса. Поскольку шампур-схема является частью шампур-программы, сказанное равносильно доказательству частичной правильности шампур-программы.

В начале главы мы задали смешной вопрос: если дракон-схемы — это теоремы, кто должен их доказывать? Ответ прост. Их никто не должен доказывать, так как все они раз и навсегда доказаны благодаря тому, что работа ДРАКОН-редактора построена как реализация исчисления икон.

А теперь добавим ложку дегтя в бочку меда. К сожалению, данный метод позволяет доказать правильность шампур-схемы и только. Это составляет лишь малую часть от общего объема работы, которую нужно

выполнить, чтобы доказать правильность программы на все 100%. Правда, есть небольшое утешение: частичное доказательство правильности программы с помощью ДРАКОН-редактора осуществляется без какого-либо участия человека и достигается совершенно бесплатно, так как дополнительные затраты труда, времени и ресурсов не требуются. А дареному коню в зубы не смотрят.

ВОЗМОЖНА ЛИ ТЕОРИЯ ВИЗУАЛЬНОГО ПРОГРАММИРОВАНИЯ?

Хотя видеопрограммирование — сравнительно молодое направление, в этой области уже имеется значительное число интересных прикладных разработок. Однако теоретическое визуальное программирование только зарождается. В доступной литературе автору удалось обнаружить всего несколько строк, которые можно в какой-то степени трактовать как программу будущих исследований в области теории видеопрограммирования: “Для визуального программирования необходимо провести строгие научные обоснования, математические определения и модели — большинство разработок в этой области носит пока эмпирический характер. Перспективным может быть применение при графическом интерфейсе техники искусственного интеллекта, которая обычно используется для описания прикладной области. Система представления знаний может включать набор визуальных примитивов, их символические описания и правила вывода заключений” [12].

Как, вероятно, заметил читатель, в настоящей работе, решая сходную проблему (проблему вывода заключений путем выполнения формальных операций над визуальными примитивами, в качестве которых использовались иконы шампур-схем), мы пошли несколько иным путем. Отличие заключается в следующем. Авторы цитированной работы говорят о “символических описаниях” визуальных примитивов, подразумевая текстовые правила вывода заключений, принятые в традиционной текстовой математической логике. Однако еще А. Ершов при построении исчисления равносильных преобразований схем Янова предпринял первую попытку отойти от “чисто текстовой” математической логики, используя в формулах правил вывода не только символические описания, но и графические изображения [9, 13]. Вместе с тем метод Ершова из-за дефектов визуального синтаксиса нельзя считать полностью формальным.

В предлагаемой книге развитие идей Ершова шло по двум направлениям. Во-первых, устранены упомянутые дефекты, что позволило сделать формализацию синтаксиса абстрактных блок-схем полной и строгой. Во-вторых, была выдвинута и проведена в жизнь идея полного отказа от символического описания визуальных примитивов (внутримашинное двоичное представление не в счет).

Можно предположить, что изложенный выше принцип визуализации математической логики, реализованный с помощью понятий визуального исчисления и визуального логического вывода, может оказаться полезным для целей более полной и строгой формализации не только языка абстрактных блок-схем (шампур-языка), но и других языков визуального представления знаний и визуального программирования.

ГИПОТЕЗА О БУДУЩЕМ ИМПЕРАТИВНЫХ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Обобщая материалы, автор не удержался от соблазна заглянуть в будущее и в порядке предварительного обсуждения высказать свои, возможно, ошибочные предположения о путях развития императивных языков, которые излагаются ниже в форме восьми тезисов.

! Несмотря на резкую критику со стороны Джона Бэкуса и ряда других ученых фон-неймановские (императивные) языки по-прежнему находят широкое применение и продолжают занимать прочные, а в некоторых областях — господствующие позиции. Логично предположить, что такая или примерно такая ситуация сохранится и впредь. Сходную позицию занимают и другие авторы, по мнению которых императивные языки "в обозримом будущем сохранят доминирующее положение в практическом программировании" [14].

! В грядущем столетии вследствие дальнейшего уменьшения удельной стоимости аппаратуры у многих персональных компьютеров экраны, по-видимому, увеличатся до размеров письменного стола, что облегчит визуализацию программирования за счет возможности непосредственной работы с чертежами формата A1 или даже A0 на экране ПЭВМ по принципу *WYSIWYG — What You See Is What You Get* (Что видишь, то и имеешь). Согласно развиваемой гипотезе, это позволит более полно использовать телесный угол и структуру человеческого поля зрения, покончить наконец с систематическим недоиспользованием богатейших возможностей человеческого глаза, задействовать мощные резервы симультанного восприятия и тем самым значительно увеличить скорость работы и продуктивность мозга программистов и пользователей. Учитывая эти соображения и остроту проблемы производительности труда в программировании, мы предполагаем, что ожидаемое увеличение габаритов экранов даст мощный стимул для широкомасштабной замены текстовых императивных языков на визуальные.

! Если допустить, что визуализация императивных языков неизбежна, целесообразно проводить ее не стихийно, а по заранее намеченному и согласованному плану, одной из целей которого следует считать частичную унификацию языков.

! В связи с этим возникает вопрос: можно ли унифицировать (хотя бы частично) императивные языки программирования? Существует ли возможность выделить некий инвариант, который можно заранее ввести в состав всех или почти всех визуальных императивных языков? С точки зрения шампур-метода, любой визуальный императивный язык включает три языка: маршрутный, командный и декларативный (см. гл. 12).

! Предлагаемая унификация императивных языков относится не к нынешним (текстовым), а к будущим (визуальным) языкам. Суть предложения состоит в том, что все визуальные императивные языки должны включать в свой состав один и тот же маршрутный язык (шампур-язык) и могут иметь любые отличия в командном и декларативном языках.

- В рамках развиваемой гипотезы шампур-язык можно рассматривать не только как абстрактную модель, но и как логический инвариант любого визуального императивного языка (включая язык ассемблера, но в этом последнем случае может потребоваться некоторая модификация шампур-метода).
- Стандартизация выделенного инварианта императивных языков осуществляется благодаря использованию стандартных шампур-редакторов, применение которых должно регламентироваться процедурой сертификации, согласованной с международной организацией стандартизации *ISO*. Опыт создания национальных и международных стандартов на блок-схемы и их повсеместного использования позволяет надеяться на успех предлагаемого предприятия.
- Очевидно, в принципе можно построить компилятор, преобразующий чертеж шампур-программы непосредственно в ассемблер и объектный код, без промежуточного преобразования в исходный текст языка высокого уровня. Не исключено, что прогнозируемый процесс визуализации программирования и образование на этой основе нового поколения (визуальных) императивных языков создаст предпосылки, при которых подобная прямая компиляция во многих случаях окажется более предпочтительной. В подобных ситуациях понятие *исходный текст программы*, видимо, полностью исчезнет из лексикона "императивных" программистов, уступив место термину *исходный чертеж программы*.

ВИЗУАЛИЗАЦИЯ ЛОГИКИ И ИНТЕНСИФИКАЦИЯ ИНТЕЛЛЕКТУАЛЬНОЙ ДЕЯТЕЛЬНОСТИ

Итак, мы показали, что визуальный синтаксис шампур-языка представляет собой визуальное логическое исчисление — исчисление икон. По этому случаю полезно сделать несколько замечаний.

- Сам факт формализации системы визуальных образов в виде логического исчисления икон можно, по-видимому, рассматривать как доказательство полной несостоятельности "принципа абсолютизации текста" (см. с. 269). Иными словами, формализация человеческих знаний отнюдь не сводится только к текстовой форме, но включает также и визуальные представления. Важно подчеркнуть, что визуальная формализация знаний не является "продуктом второго сорта", она удовлетворяет самым строгим критериям математической логики и в этом смысле представляет собой вполне законный и полноценный интеллектуальный продукт. Наибольшая интеллектуальная эффективность достигается при использовании синтетического метода, объединяющего достоинства текстовой и визуальной формализации.
- Согласно традиционной точке зрения, словесно-логическое (левополушарное) мышление отличается четкостью и ясностью, в то время как образное (правополушарное) мышление есть нечто расплывчатое, интуитивное и почти неуловимое. Полученные результаты позволяют утверждать, что по крайней мере часть абстрактных зрительных образов может быть преобразована в безукоризненно строгую форму, расчленена на отдельные микрообразы (иконы), превра-

щена в компьютерное меню и снабжена строгими правилами, позволяющими строить большие и сложные зрительные образы из атомарных микрообразов.

! Известно, что “богатые формальные языки математической логики и успешный опыт работы с ними создали одну из объективных предпосылок для создания... вычислительных машин, пользующихся в настоящее время весьма разнообразным спектром формальных языков программирования” [1]. Это утверждение до последнего времени было ограничено рамками текстовой парадигмы: и в отношении логики, и в отношении программирования. Появление визуальных исчислений позволяет расширить эти рамки и распространить их на визуальный случай.

! Необходимость в этом давно назрела, так как теория в данном вопросе отстает от практики. С появлением интегрированных *CASE*-технологий компьютерные чертежи (например, схемы “сущность-связь”, схемы декомпозиции, схемы потоков данных и т. д.) приобрели целый ряд замечательных свойств. Они превратились в формальные визуальные языки высокой точности. Компьютер может понимать точные значения указанных чертежей, хранить их в виде, пригодном для глубокой обработки, преобразовывать чертежи друг в друга, выявлять несоответствие между ними, а также их неполноту, чтобы гарантировать целостность общей картины. И, что особенно важно, извлекая из чертежей нужную информацию, компьютер с помощью программы “генератор кода” получает выполнимый код. Таким образом, можно надеяться, что в будущем традиционная дружба математической логики и информатики уже не будет ограничена дряхлым забором текстовой парадигмы и распространится на более широкое визуальное проблемное поле.

! Когнитивная формализация знаний — это синтез логико-математической формализации и когнитивного подхода. Цель метода — улучшить понимаемость сложных формальных описаний и проблем за счет учета реальных интеллектуальных характеристик человека на основе достижений эргономики. Выше автор попытался продемонстрировать применение метода когнитивной формализации на живом примере — разработке языка ДРАКОН. В гл. 1—15 систематически подчеркивалось, что при создании языка ДРАКОН эргономические соображения являются главными, основополагающими; было описано большое количество конкретных эргономических приемов, образно говоря, эргономических кирпичиков, из которых строился законченный эргономический облик языка. При этом, однако, в тени оставался вопрос: обладает ли построенный язык (точнее говоря, его визуальная часть) строгими формальными характеристиками? В настоящей главе, опираясь на исчисление икон, мы вправе дать обоснованный ответ: да, обладает.

Таким образом, когнитивная формализация знаний — это не утопия, не благое пожелание или розовая мечта, а работоспособный метод, способный приносить нужные плоды и обеспечить увеличение продуктивности человеческого мозга.

! Логическая формализация знаний, восходящая к силлогистике Аристотеля, который впервые использовал буквы для обозначения поня-

тий, явилась замечательным достижением человеческого гения. За две тысячи лет своего существования логическая наука добилась выдающихся успехов. Но вот парадокс: называя себя наукой о законах и формах человеческого мышления [5], логика вместе с тем полностью абстрагировалась от конкретных характеристик человеческого мышления и мозга, изучаемых в психологии, нейробиологии и эргономике. На предыдущих этапах развития человечества, когда интеллектуальные задачи были относительно простыми, а число интеллектуальных работников невелико, подобное игнорирование не приносило заметного вреда. Однако сегодня положение изменилось.

! Лавинообразное усложнение цивилизационных процессов привело к появлению интеллектуальных задач немыслимой прежде сложности, находящихся на пределе возможностей человеческого мозга. Острая необходимость интенсификации интеллектуальной деятельности, связанная с этим процессом, требует создания принципиально новых форм и методов интеллектуальной работы, способных качественным образом увеличить умственную производительность (мозга) интеллектуальных работников и учащихся, улучшить качество интеллектуального взаимодействия между людьми, обеспечить максимально возможную защиту от интеллектуальных заблуждений, просчетов, путаницы и взаимного непонимания, усилить эффективность индивидуального и коллективного интеллекта людей. Думается, что когнитивная формализация знаний, объединяющая всю мощь традиционных математических и логических методов с развиваемым в эргономике (включая психологию и нейробиологию) точным учетом когнитивно-значимых характеристик человеческого мозга и интеллекта в рамках единой целостной концепции, может в немалой степени содействовать решению обозначенной проблемы.

! Представленные материалы позволяют сделать обоснованные предположения о развитии информационных технологий в XXI веке. При дальнейшем совершенствовании компьютеров недостаточная продуктивность человеческого мозга станет основным фактором, сдерживающим рост эффективности организаций и ограничивающим интеллектуальные возможности человечества.

! Можно ожидать, что улучшение работы ума, повышение интеллектуальной производительности человека превратится в центральную проблему информационных технологий. Однако существующая теория и практика информатизации и компьютеризации не располагают эффективными средствами для решения задачи. Отсюда вытекает необходимость создания новых теоретических подходов, чтобы с их помощью построить информационные технологии нового типа — когнитивные информационные технологии, отличительная особенность которых состоит в том, что повышение продуктивности мозга рассматривается как высшая, приоритетная цель, которой подчинены все остальные цели (с учетом необходимых компромиссов, диктуемых экономическими и иными ограничениями).

! По нашему мнению, в XXI веке произойдет переход к широкому использованию когнитивных информационных технологий в науке, технике, образовании, медицине, обороне, бизнесе, государственной службе и других сферах, что позволит значительно увеличить интел-

лектуальный потенциал и интеллектуальные возможности общества и таким образом проложить надежный путь к новому качеству интеллектуальной жизни.

Это громадная, поистине необъятная по сложности задача, выходящая далеко за рамки данной книги. Для ее решения необходим перелом в сознании: надо уяснить, что человеческий мозг обладает колоссальными интеллектуальными резервами, которые сегодня не используются, но которые можно и нужно задействовать с помощью когнитивно-эргономических методов. Снова оговоримся: существующие когнитивные методы для этого недостаточны, нужны новые подходы. Откуда их взять? По мнению автора, материалы данной книги, хотя и относятся к частному случаю, тем не менее обладают достаточной общностью и могут послужить основой для разработки — с необходимыми уточнениями — нового поколения формализованных когнитивно-эргономических методов.

ВЫВОДЫ

1. Противоречие между скромными интеллектуальными возможностями отдельного человека и почти неограниченным объемом знаний, который он должен приобрести в течение жизни, — одно из наиболее драматических противоречий современного общества, основанного на знаниях. Сегодня наука не располагает эффективными средствами для решения этой проблемы.
2. Выход из положения мы видим в тотальной эргономизации науки и образования, цель которой — коренным образом улучшить визуальные формы фиксации знаний, согласовав их с тонкими характеристиками глаза и мозга.
3. Разработка исчисления икон говорит в пользу этой гипотезы и служит примером, подтверждающим актуальность нового междисциплинарного направления — логико-эргономических исследований.
4. Стихийный процесс визуализации логики, который начинает разворачиваться в последнее время (см., например [15]), должен опираться на прочную эргономическую основу.
5. Серьезным тормозом для реализации идей эргономизации знания является устаревшее представление, согласно которому в системе научного знания наглядно-чувственные образы занимают подчиненное место и служат только для создания неформальных наглядных моделей, рисунков, чертежей.
6. Сторонники идеи тотальной эргономизации науки и образования должны вести борьбу на два фронта:
   - против приверженцев принципа абсолютизации текста;
   - против глашатаев стихийной визуализации, которые не понимают разницы между кустарным преобразованием текста в изображение и научными методами визуализации, опирающимися на проч-

ся развитие идей когнитивной формализации знаний и когнитивных информационных технологий, которым, на наш взгляд, принадлежит будущее.

Глава 18

МЕСТО ЯЗЫКА ДРАКОН В СИСТЕМЕ ЧЕЛОВЕЧЕСКОЙ КУЛЬТУРЫ

> Язык... представляет собой одно из главнейших орудий или пособий мысли... Несовершенство этого орудия... мешает делу и уничтожает всякое доверие к его результатам.
>
> *Джон Стюарт Милль*

МЕЖДУ СЦИЛЛОЙ И ХАРИБДОЙ

В истории искусственных языков можно выделить два периода. На первом этапе (создание языков воляпюк, эсперанто и т. д.) ставилась амбициозная задача построения всемирного международного языка, призванного улучшить взаимопонимание между людьми и народами. К сожалению, из-за чрезвычайной сложности задачи и недостаточной теоретической проработки попытка потерпела провал — гора родила мышь. Удивительно другое: несмотря на неудачу, проект вызвал всеобщий интерес и получил мировую известность. Этот факт свидетельствует о том, что уже в то время идея "языка для взаимопонимания" задевала нервный центр важной общественной потребности.

На втором этапе был предпринят более реалистичный подход "по одежке протягивай ножки". Направление поиска было резко сужено, масштаб проблемы уменьшен и ограничен частными задачами по созданию формальных языков программирования. На этом пути, как хорошо известно, достигнуты впечатляющие успехи.

Между тем проблема непонимания продолжала обостряться и сегодня вступила в критическую фазу, которую можно охарактеризовать как "паралич понимания". Возникла настоятельная необходимость еще раз вернуться к идее всемирного языка понимания и взаимопонимания и критически ее переосмыслить. По-видимому, современный язык для понимания следует строить на принципиально иной концептуальной основе, позволяющей совершить удачный маневр и провести корабль нового проекта в узком проливе между Сциллой несбыточной "всемирности" (где потерпели крах воляпюк и эсперанто) и Харибдой узкой специализации (которая превращает языки программирования в никому не понятные египетские иероглифы и тем самым неоправданно сужает их социальную базу).

Спастись от Сциллы довольно просто — надо лишь отказаться от претенциозной идеи одного всемирного языка (построить который скорее всего в принципе невозможно) и сделать акцент на создании частных языков, каждый из которых полезен в своей области, которая, впрочем, не должна быть слишком узкой.

Гораздо труднее избавиться от Харибды языковой специализации (когда язык создается "только для своих") и придумать универсальный язык, способный удовлетворить интересы самых различных групп спе-

циалистов. Задача состоит в том, чтобы найти спасительную идею, которая позволила бы резко расширить социальный плацдарм языка и сделать его полезным для миллионов. Надо превратить язык, понятный только членам какой-то одной узкой касты (например, программистов), в язык взаимопонимания для широкого круга интеллектуальных работников и учащихся.

ПРИНЦИП СТРУКТУРИЗАЦИИ ДЕЯТЕЛЬНОСТИ

Чтобы прояснить суть вопроса, вернемся еще раз к структурному программированию. Из теоремы Бома и Джакопини вытекает, что логическая структура программы может быть выражена комбинацией ограниченного числа базовых структур. Это означает, что идея структурных конструкций дает читателю программы столь необходимый компас. Пробираясь сквозь джунгли программного текста, он как бы обретает "третий глаз": разбиение сложной программы на структурные конструкции облегчает понимание и упрощает работу. Говоря языком эргономики, это достигается за счет укрупнения оперативных единиц восприятия[1].

Наибольший недостаток структурного программирования лежит не в области техники, а в социальной плоскости. Дело в том, что этот метод помогает улучшить работу ума очень небольшого числа людей, а именно — программистов. Все остальные работники умственного труда не имеют к этому "празднику" никакого отношения и ничего не выигрывают.

К счастью, данный недостаток можно и нужно устранить, поскольку *идея структуризации является универсальной и допускает обобщение на любую деятельность*, относящуюся к любым социальным и профессиональным группам. Можно предложить

Принцип структуризации деятельности. Любая деловая деятельность независимо от ее характера, сложности, профессиональной принадлежности, социальной направленности и предметной области может быть описана с помощью ограниченного числа структурных конструкций, которые можно охарактеризовать как логические инварианты деятельности. В качестве последних предлагается использовать конструкции визуального структурного программирования или, что одно и то же, конструкции визуального синтаксиса техноязыка ДРАКОН. Примеры реализации этого принципа были разъяснены в гл. 13.

ГЕНЕРАЛЬНАЯ КОНЦЕПТУАЛЬНАЯ СХЕМА

Программирование есть частный вид деятельности. Это исторически первый тип деятельности, к которому был применен принцип структуризации. Обобщение данного принципа на любую деятельность мы рас-

---

[1] Оперативная единица восприятия — это семантически целостное образование, формирующееся в результате перцептивного обучения и создающее возможность практически одноактного, симультанного и целостного восприятия объектов внешнего мира независимо от числа содержащихся в них признаков [1].

сматриваем как искомую “спасительную идею”, как важный шаг, имеющий прямое отношение к главной проблеме — проблеме улучшения работы ума. Данный вывод нуждается в пояснениях.

На рис. 138 представлена схема, позволяющая выявить восемь источников (предпосылок), совместный анализ которых порождает “короткое замыкание” идей, следствием чего и является принцип структуризации деятельности.

1. Первым источником является принятое в искусственном интеллекте деление знаний на декларативные и процедурные.
2. Для наших целей эта схема подвергается некоторым изменениям (рис. 138, блок 2):
   - ! предполагается, что речь идет о письменном представлении знаний, предназначенных для зрительного восприятия человеком;
   - ! не совсем удачный термин “процедурные” заменяется на более знакомое и широко распространенное слово “технологические” (см. гл. 3);
   - ! технологические знания делятся на командные и управляющие (см. гл. 12).
3. Третьим источником является структурное программирование (рис. 138, блок 3).
4. Дальнейшее развитие структурной идеи приводит к переходу от текстового структурного программирования к визуальному (рис. 138, блок 4), так как последнее обладает многочисленными достоинствами, подробно рассмотренными в гл. 16.
5. Пятым источником служат блок-схемы, нашедшие широкое распространение с первых дней появления программирования.
6. Следующий шаг рассуждений приводит к отказу от блок-схем и замене их на дракон-схемы (рис. 138, блок 6), которые обладают неоспоримыми преимуществами (см. гл. 6—16).
7. В качестве седьмого источника используется исчисление икон (рис. 138, блок 7).
8. И наконец, последним источником служит обобщенная трактовка понятия “деятельность” (рис. 138, блок 8), охватывающая не только действия, совершаемые людьми, но и операции, выполняемые машинами. Последние рассматриваются как делегированная деятельность, исполнение которой человек поручает (делегирует) спроектированным им техническим устройствам.

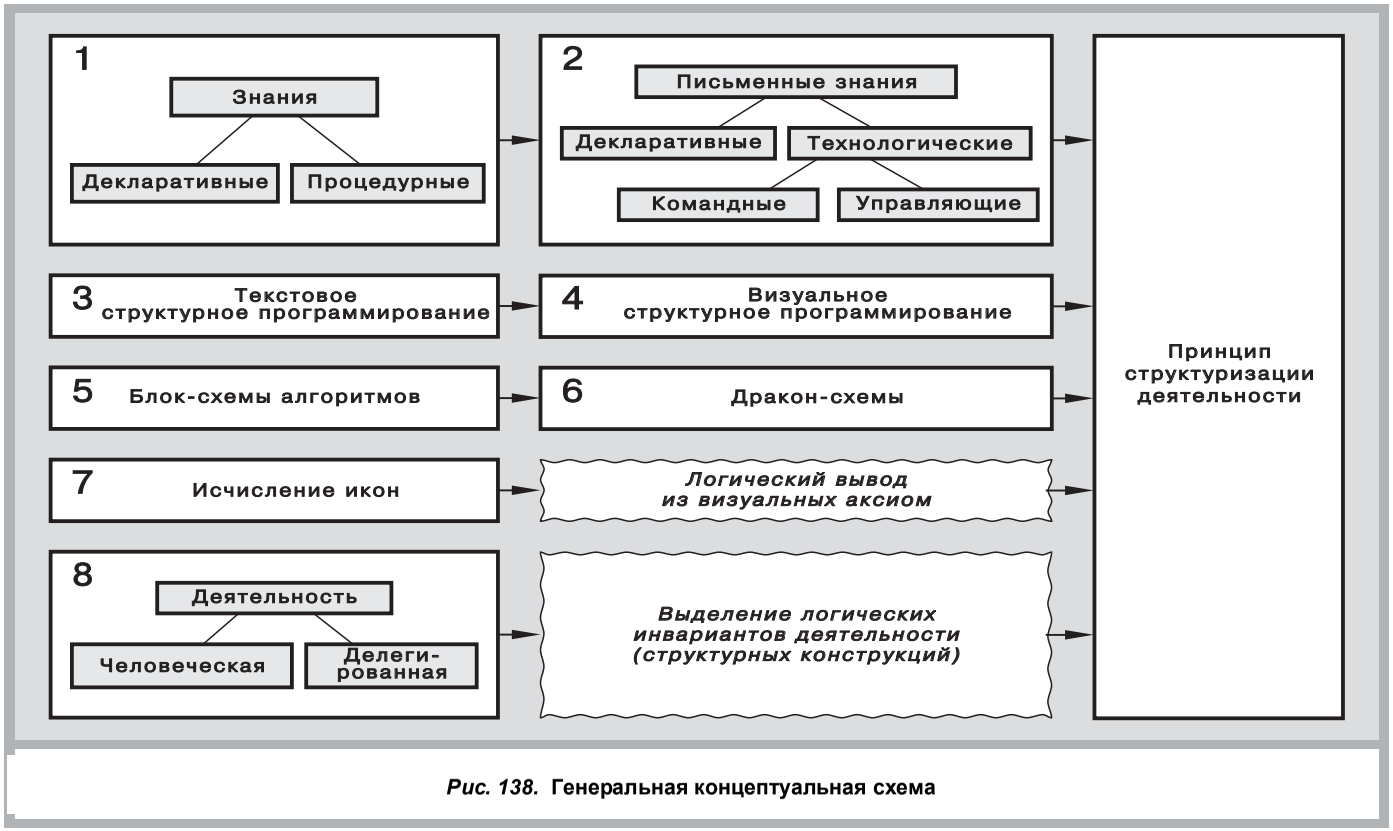


Рис. 138. **Генеральная концептуальная схема**

Важно подчеркнуть, что техноязык ДРАКОН предоставляет единые стандартные средства для описания как собственно человеческой, так и делегированной (машинной) деятельности. Благодаря этому появляется возможность системного видения и анализа проблем, так как на одном и том же чертеже оба вида действий отображаются взаимоувязанно в рамках одного алгоритма — как его чередующиеся фрагменты.

Схема на рис. 138 названа генеральной концептуальной схемой, так как она обобщает значительную часть содержания этой книги, представляет его в сжатой форме, изображает панораму основных идей и взаимосвязь понятий, позволяет в наглядной форме проследить ход рассуждений и логику развития мысли.

ПРОБЛЕМА ДЕЯТЕЛЬНОСТИ В ЭРГОНОМИКЕ

Два понятия: “алгоритм” и “деятельность” олицетворяют две огромные страны, лежащие на разных научных континентах и разделенные океаном взаимного отчуждения и недоверия. Это два мира (мир математики и программирования и мир эргономики и гуманитарных наук), в которых господствует разный стиль научного мышления. Это две грандиозные области весьма глубоких и тонких исследований, между которыми нужно построить широкий и прочный мост взаимопонимания с интенсивным двусторонним движением идей. Строительство такого моста мы рассматриваем как обобщение высшего ранга, тесно связанное с проблемой улучшения работы ума и способное проложить путь к новым мощным научным прорывам.

Категория деятельности является важнейшей в системе эргономического знания. Деятельность в эргономике выступает как:

1) предмет объективного научного изучения;
2) объект управления;
3) предмет проектирования и моделирования;
4) предмет многоплановой оценки.

По мнению В. Мунипова, исследование деятельности связано, в частности:

! с проблемой конструирования и реконструирования профессий, формирования сочетания отдельных операций и действий, образующих целостную деятельность;

! с развитием научной организации и охраны труда, всех наук, изучающих трудовую деятельность человека в интересах решения важнейшей народнохозяйственной задачи — кардинального повышения производительности труда и качества работы.

Исследованию деятельности посвящено большое число работ [2]. К середине 60-х годов были разработаны методы алгоритмизации деятельности человека-оператора, выполняющего функции контроля и управления технологическим объектом (Д. Агейкин, А. Галактионов). На протяжении ряда лет изучалась деятельность авиадиспетчера (М. Груздев), диспетчера железнодорожного узла (Д. Завалишина и В. Пушкин), оператора дистанционного управления технологическими процессами (Д. Ошанин, В. Венда), оперативного персонала электростанций (С. Гаджиев, К. Гуревич).

Развивался алгоритмический подход к анализу деятельности оператора (А. Галактионов, А. Чачко). Г. Зараковским был разработан формализо-

ванный язык для записи и анализа профессиограмм машинистов и рулевых крупных морских судов.

Результаты работ этого направления позволили перейти к созданию математических моделей, методов количественной оценки эффективности и надежности деятельности человека-оператора, определения его загрузки, подойти к решению важной проблемы инженерной психологии — распределению функций между человеком и машиной в системе “человек — машина”.

Были разработаны различные концепции и подходы к проектированию операторской деятельности.

- Системный подход к анализу и оптимизации взаимодействия человека и машины (Б. Ломов).
- Психофизиологический и алгоритмический анализ деятельности (Г. Зараковский).
- Структурно-эвристическая концепция послойной переработки информации оператором (В. Рубахин).
- Принцип “включения” (А. Крылов).
- Структурно-алгоритмический подход к анализу и проектированию деятельности (Г. Суходольский).
- Функционально-структурная теория (А. Губинский).
- Структурно-психологическая концепция (В. Венда).
- Концепция генезиса психологической системы деятельности (В. Шадриков).
- Концепция идеализированных структур деятельности (А. Галактионов).

Подробный обзор зарубежных работ содержится в [3].

Анализируя эргономические проблемы проектирования и моделирования деятельности, можно отметить разнообразие различных подходов и методов, среди которых достойное место занимает алгоритмический (процессуальный) подход. К сожалению, существующие способы описания алгоритмов во многом устарели и не позволяют в полной мере выявить все преимущества этого метода. Думается, использование языка ДРАКОН в качестве стандарта для описания структуры деятельности — в разумном сочетании с другими методами — позволит сделать заметный шаг вперед при решении многих задач, связанных с созданием современных человеко-машинных систем.

ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ: АЛГОРИТМИЗАЦИЯ — ЭТО НОЧНОЙ КОШМАР!

Нынешняя наука похожа на лабиринт, составленный из клеток, некоторые из которых наглухо изолированы друг от друга. Представители разных научных дисциплин порою как бы “не слышат” друг друга. Поэтому сегодня, как никогда, нужны общезначимые языки, удобные для всех, способные “взламывать” междисциплинарные перегородки и укреплять взаимопонимание между разными отрядами ученых. В качестве примера рассмотрим “взаимоисключающие” высказывания некоторых специалистов по искусственному интеллекту (ИИ) и инженерных психологов.

Специалисты по ИИ: долой алгоритмизацию!

Во многих случаях полезно различать два вопроса: ЧТО надо сделать? и КАК это сделать? Большинство приверженцев ИИ отстаивают принцип приоритета декларативных знаний, согласно которому человеку намного легче ответить на вопрос “что?”, чем на вопрос “как?” Поэтому они призывают отказаться от императивных языков (языков типа “как”) и заменить их декларативными языками (языками типа “что”).

Достигаемое преимущество, по их мнению, объясняется тем, что программисту гораздо удобнее давать определения ситуаций и формулировать задачи вместо того, чтобы во всех деталях описывать способ решения этих задач. При декларативном подходе от программиста уже не требуется построения алгоритма, решающего задачу, поскольку программы изменяют свой облик и “всегда будут логически описывать саму задачу, а не процесс ее решения” [4].

Итак, почему следует отказаться от алгоритмов? На это есть три причины. Во-первых, декларативное описание — это хорошее (рациональное), описание, а процедурное — плохое (нерациональное), поэтому “требование процедурного описания задачи пользователя с самого начала означает отказ от рационализации решения проблемы” [5]. Во-вторых, человеческие мозги не годятся для создания и понимания алгоритмов: мы (люди) “с трудом составляем алгоритмы, потому что само это понятие нам несвойственно”, ибо “действия людей совершенно не похожи на работу привычных алгоритмов” [6]. Исходя из этого (в-третьих), экстремисты от ИИ заявляют, что вскоре декларативные языки одержат всемирно-историческую победу над процедурными “заморышами”; последние — это языки проклятого прошлого, а декларативные языки-победители — это, разумеется, “языки будущего” [6, 7].

Такие или примерно такие рассуждения встречаются в работах если не всех, то многих специалистов по искусственному интеллекту, превратившись в некое общее место, своего рода “символ веры” ИИ.

Естественно возникает вопрос: являются ли эти рассуждения доказательными и, если да, при каких условиях эти доказательства справедливы?

Инженерные психологи: алгоритмизация деятельности — наше спасение!

В отличие от ИИ-специалистов большинство эргономистов не спешат предавать алгоритмы анафеме. Почему? В частности, потому, что “алгоритмы суть единственное средство для объективного выражения… тех составляющих операторскую деятельность нормативных и инициативных действий и взаимодействий, которые вне алгоритмизации остаются только компонентами личного профессионального опыта квалифицированных операторов. А значит, остаются малодоступными для профессиональной подготовки оперативного персонала, для эффективизации и гуманизации операторского труда. Алгоритмизация действий и взаимодействий есть важнейший результат и основа для получения остальных результатов инженерно-психологического проектирования операторской деятельности [8].

Другой пример. Группа военных эргономистов изучала проблему организации взаимодействия членов экипажей многоместных летатель-

ных аппаратов при отказе авиационной техники в полете. Выяснилось, что около половины летных происшествий, возникших на многоместных самолетах из-за отказов в полете, могли бы быть предотвращены при оптимальном взаимодействии между членами экипажа. Как же добиться оптимальности? Авторы предлагают улучшить методы обучения и тренировки экипажей, подчеркивая, что при обучении "немалое значение имеют формы и способы представления членам экипажа информации о порядке их взаимодействия". По результатам исследований авторы приходят к выводу, что наиболее приемлемой формой представления указанной информации является "алгоритмический способ", в котором схема алгоритма представляется в графической форме [9].

Работники образования: алгоритмизация — это хорошо!

Хотя некоторые преподаватели вслед за Р. Ковальским ведут активные эксперименты по внедрению в средние школы языка Пролог, большинство ученых, работающих в сфере среднего образования, по-видимому, являются сторонниками императивных языков. По их мнению, развитие алгоритмического стиля мышления школьников "есть основная цель курса информатики", так как школьный курс "должен следовать науке, а процедурная традиция является сегодня ведущей и наиболее разработанной" [10].

Кто же прав: декларативисты или императивисты?

По нашему мнению, императивный и декларативный подходы являются не альтернативными, а взаимодополняющими: для одного круга задач лучше подходит первый способ представления знаний, для другого — второй. Подобная сбалансированная позиция приходит на смену прежнему экстремизму и начинает завоевывать все больше сторонников. В частности, В. Сергеев, анализируя "процедурно-декларативную контроверзу", делает следующее заключение: "Нам представляется, что эти два подхода выявляют две стороны, два способа человеческого мышления и являются дополнительными примерно в том смысле, в котором различные представления в виде частицы и волны присутствуют в принципе дополнительности Н. Бора".

Обобщая изложенное, можно предложить несколько замечаний.

! Утверждения типа "императивные методы нерациональны", "человеку легче описать задачу, чем процесс ее решения", "люди с трудом составляют алгоритмы, потому что само это понятие им несвойственно" не содержат необходимых ограничительных формул и не удовлетворяют критериям научной строгости. Проще говоря, они голословны и бездоказательны. Существует обширный класс задач, для которых названные положения в корне неверны.

! Парадокс в том, что искусственный интеллект как научное направление нисколько не нуждается в приведенных выше некорректных (чтобы не сказать — нелепых) утверждениях, так как реальные и общепризнанные достижения искусственного интеллекта совершенно не зависят от этой дезориентирующей словесной мишуры. По-

следняя напоминает пыль, которую нерасторопные служители забыли стереть с золотой статуэтки.

! Хотя аргумент “все декларативное хорошо, все императивное плохо” и его многочисленные вариации являются некорректными, тем не менее в период “детства” ИИ они сыграли позитивную и мобилизующую роль, объединяя усилия ИИ-специалистов в борьбе против общего “императивного врага” и тем самым концентрируя их внимание на пионерской идее декларативного языка. Однако теперь, когда декларативная задача решена, нет никакой необходимости сохранять образ мифического “императивного врага”. Сегодня — в эпоху зрелости искусственного интеллекта — указанный аргумент полностью себя скомпрометировал и должен быть отброшен. Мавр сделал свое дело, мавр должен уйти.

ЭРГОНОМИЧЕСКИЙ АНАЛИЗ ПРОЕКТНО-КОНСТРУКТОРСКОЙ ДЕЯТЕЛЬНОСТИ

Социальная значимость языка ДРАКОН как инструмента для описания структуры деятельности тесно связана с той особой ролью, которую деятельность (как понятие и социальное явление) играет в системе человеческой культуры.

По мнению В. Сагатовского, “деятельность — это фундаментальное философское понятие, сопоставимое по своей общности с категориями общественного бытия и сознания... Это ключевое понятие для понимания специфики “мира человека”. Выходит немало книг и статей, в которых говорится о важности принципа деятельности, о необходимости деятельностного подхода, иной раз даже провозглашается идея “всеобщей теории деятельности” [11, 12].

История, заметил классик, есть не что иное, как деятельность преследующего свои цели человека. В наши дни представления о деятельности начинают коренным образом меняться. В чем суть изменений?

На протяжении тысячелетий человеческая деятельность не проектировалась, а формировалась стихийно. Однако на современном этапе остро встает вопрос о проектировании, моделировании и формализации описания деятельности, которые оказываются полезными при решении многих задач.

Вопрос о проектировании деятельности впервые был поставлен Б. Ломовым в 1967 г. в рамках инженерной психологии, поскольку проект деятельности должен выступать как основа решения всех остальных задач построения систем “человек—машина”. На первом этапе инженерная психология и эргономика сконцентрировали свое внимание на проектировании деятельности оператора (летчика, диспетчера и т. д.), так как профессия оператора получила большое распространение и в ряде случаев оказалась решающей. Однако операторский “бум” не мог длиться бесконечно. Стало ясно, что интенсивное изучение деятельности оператора не снимает с повестки дня другие необходимые исследования, в первую очередь исследование деятельности проектировщиков и конструкторов (В. Моляко, 1983).

Проектно-конструкторская деятельность — один из наиболее сложных видов интеллектуального творческого труда. Эта деятельность становится все более важной, так как от нее зависит создание принципиально новых видов техники, разработка сложных социотехнических

систем (например, автоматизированных систем управления), разработка новых технических и социальных проектов и технологий и т. д. Повышение эффективности конструкторской деятельности человека в связи с ускорением научно-технического прогресса предполагает разработку инженерно-психологических средств и методов активизации творчества конструктора с учетом общих концепций теории деятельности.

ПОДВОДНЫЕ КАМНИ ПРОЕКТНО-КОНСТРУКТОРСКОЙ ДЕЯТЕЛЬНОСТИ

Во многих случаях причиной промышленных катастроф и аварий являются проектно-конструкторские недоработки. Вызванные ими отказы техники можно разбить на две группы:

1) фатальные (которые невозможно предугадать заранее);
2) прочие (назовем их дефектами).

Фатальные отказы связаны с тем, что наше знание ограничено. Подобные отказы имеют место лишь тогда, когда научно-конструкторский коллектив столкнулся с новым, доселе неизвестным фактом или явлением. Такие случаи мы исключаем из рассмотрения, потому что на нет и суда нет. Если наука не в состоянии предвидеть и предотвратить негативные последствия, возникает безвыходная ситуация, когда авария может стать неизбежной.

Дело несколько облегчается тем, что отказы и аварии чаще всего вызываются не фатальными отказами, а *дефектами*. Термин “дефект” описывает ситуацию, когда авария не приводит к приращению научного знания, потому что вся необходимая для ее предотвращения информация была известна заблаговременно, но не была использована.

Почему? Возможны три причины:

- чтобы избежать дефекта и предотвратить аварию, конструктор должен сопоставить и логически увязать слишком большое количество хорошо известных фактов, т. е. выполнить огромный объем умственной работы, намного превышающий реальные возможности его мозга;
- хорошо известные факты находятся в головах разных людей, между которыми отсутствует должное интеллектуальное взаимопонимание и взаимодействие на основе четко налаженной коммуникации;
- у конструктора недостаточная мотивация к выполнению работы, следствием чего является безответственность и халатность (этот случай мы не рассматриваем, так как сегодня имеются эффективные механизмы управления мотивацией).

Первые два случая представляют наибольший интерес, так, причиной дефекта является отсутствие понимания и взаимопонимания. Один из путей предотвращения дефектов — эргономизация науки и проектно-конструкторской деятельности с помощью специальных средств, обеспечивающих улучшение работы ума, среди которых не последнее место занимает язык ДРАКОН. Обобщая эту мысль, следует поставить вопрос о качественно новом шаге в развитии эргономических идей и распространить понятие “проект деятельности” на сложную творческую деятельность.

Переход к научно-обоснованному проектированию творческих видов деятельности, в частности проектно-конструкторской деятельности, — это принципиально новая и крайне сложная задача, которая требует создания новых теоретических средств. Ошибки здесь недопустимы, так как неудачное проектирование конструкторской деятельности может привести к серьезным негативным последствиям.

Многие аварии и катастрофы имеют первопричиной не только того человека (человека-оператора), который управлял техникой в момент аварии, но и других людей, которые проектировали эту несовершенную технику и создали недостаточно приемлемую организацию и условия для работы исполнителей.

В связи с этим возникает ряд теоретических проблем, которые анализируются далее на примере аварии Чернобыльской АЭС.

ПОЧЕМУ ВЗОРВАЛСЯ ЧЕРНОБЫЛЬСКИЙ РЕАКТОР?

Традиционный подход к анализу причин чернобыльской аварии

Согласно традиционной точке зрения, авария на четвертом энергоблоке чернобыльской АЭС произошла по нескольким причинам:

1) научное заблуждение (не были раскрыты и поняты физические особенности существовавшей до аварии активной зоны реакторов РБМК, заключающиеся в большом положительном паровом коэффициенте реактивности и пространственной неустойчивости нейтронного потока);
2) конструктивный дефект системы управления и защиты реактора;
3) ошибка эксплуатационников, которые нарушили правила и привели реактор в такое состояние, в котором система защиты (из-за упомянутого конструктивного дефекта) уже не могла предотвратить взрыв [13, 14].

Правомерно спросить: являются ли эти объяснения исчерпывающими? Дают ли они возможность выявить “истинного” виновника чернобыльской катастрофы? Позволяют ли они — если смотреть на проблему не с позиций частного случая, пусть даже важного и трагического, а с позиций системного подхода к анализу глобальных проблем, порождаемых неуклонным усложнением цивилизационных процессов, — создать научно обоснованные и гарантоспособные механизмы предотвращения крупномасштабных катастроф и аварий? По нашему мнению, на все эти вопросы следует ответить отрицательно.

Возможна ли гарантоспособная деятельность?

История науки и техники — это не только успехи и достижения. Это и берущая начало в глубине веков непрерывная цепь заблуждений, промахов и упущений, негативные последствия которых постепенно нарастали, пока, наконец, не достигли драматических масштабов чернобыльского инцидента. По мнению экспертов, “чернобыльская авария — величайшая катастрофа за всю историю Земли, в том числе и истории культуры, однако это еще предстоит осознать человечеству. Ликвидация всех ее последствий невозможна, ибо они вечны, сейчас лишь

начался процесс их осмысливания... Она станет переломным пунктом в развитии современной технологической цивилизации...” [15]. Благодаря Чернобылю проблема безопасного развития цивилизации приобрела небывалую остроту и стимулировала поиск новых подходов к ее изучению.

При анализе проблемы цивилизационной безопасности приходится исходить из того, что в процессе крупномасштабных исследований и разработок, при создании сложных объектов и систем, могут участвовать десятки и сотни тысяч людей, причем существует огромное количество сбоев в человеческой деятельности, неблагоприятное сочетание которых может привести к аварии объекта или экологической катастрофе. Возникает вопрос: можно ли решить эту проблему в принципе? Можно ли иметь гарантию успешного ведения дел? Может ли человеческая деятельность быть гарантоспособной?

Понятие гарантоспособности *(dependability)* предложили Альгирдас Авиженис и Жан-Клод Лапри применительно к анализу вычислительной техники. Согласно разработанной ими концепции, гарантоспособность — “это свойство вычислительной системы, позволяющее обоснованно полагаться на выполнение услуг, для которых она предназначена” [16].

По нашему мнению, понятие гарантоспособности является плодотворным. Более того, мы считаем возможным существенно расширить объем понятия и распространить его на любые эргатические системы (системы с участием человека), т. е. использовать понятие гарантоспособности для анализа любых видов деловой активности людей, в том числе наиболее сложных видов интеллектуальной деятельности, включая деятельность руководителей, ученых и специалистов. В рамках такого подхода можно сказать, что *гарантоспособность* — это свойство эргатической системы, позволяющее обоснованно полагаться на выполнение задач, для которых она предназначена.

Принцип проектирования гарантоспособной деятельности

За отказом или сбоем любой технической или социальной системы стоят люди, которые ее исследовали, анализировали, проектировали, создавали, инициировали, испытывали, включали в состав более крупной системы и эксплуатировали. Но еще более важно понять, что есть (или, по крайней мере, должен быть) и другой, в некотором смысле “более высокий” слой людей. Речь идет о тех, кто призван воспитывать и обучать людей из предыдущего слоя, с ранних лет формировать их личность, повышать квалификацию и в явной или неявной форме проектировать их деятельность.

В настоящее время в большинстве сложных случаев человеческую деятельность никто специально не проектирует, она складывается стихийно — как эмпирическое обобщение опыта, традиций и соображений здравого смысла тех или иных работников и социальных групп. С другой стороны, известно, что здравый смысл хорошо работает лишь в относительно простых ситуациях, а в сложных случаях полагаться на здравый смысл опасно — здесь нужен научный подход к проблеме. Отсюда вытекает несколько выводов, которые в совокупности можно

охарактеризовать как *принцип проектирования гарантоспособной деятельности*.

- ! Поскольку цивилизованный мир — продукт человеческой деятельности, постольку любые промышленные аварии и социальные инциденты — это следствие тех или иных сбоев и дефектов человеческой деятельности.
- ! Сбои и дефекты человеческой деятельности — это (прямой или опосредованный) продукт человеческих заблуждений, просчетов, ошибок и взаимного непонимания, неумения организовать эффективное интеллектуальное взаимодействие.
- ! Чтобы устранить сбои и дефекты человеческой деятельности (или, по крайней мере, уменьшить их вероятность), необходимо научиться проектировать деятельность. Для этого необходима *теория проектирования человеческой деятельности*, которая должна обеспечить эффективные и согласованные действия больших и малых человеческих коллективов. Эта теория должна объяснить природу человеческих заблуждений, просчетов, ошибок, взаимного непонимания и указать метод, позволяющий уменьшить их вероятность. *Главная задача теории* — повысить качество деятельности таким образом, чтобы, не ущемляя свободу личности и права человека, вместе с тем сделать ее эффективной и гарантоспособной.

Теория проектирования гарантоспособной деятельности должна охватывать все виды деловой активности людей: научную, техническую, производственную, политическую, управленческую, учебную деятельность и т. д.

Гарантоспособный совокупный работник

Введем понятие “совокупный работник” для обозначения всех людей, прямо или косвенно участвующих в создании крупномасштабного объекта, например АЭС. Будем считать, что совокупный работник является *гарантоспособным*, если заблуждения, просчеты и ошибки отдельных индивидов своевременно выявляются и устраняются, нейтрализуются или предотвращаются и, следовательно, не могут оказывать негативное влияние на эффективность и качество конечных результатов деятельности совокупного работника.

В теории надежности рассматривается проблема: как построить надежную систему из ненадежных элементов? Применительно к эргатическим системам (состоящим из людей) эта проблема формулируется так: как спроектировать гарантоспособного совокупного работника из ненадежных (негарантоспособных) индивидов?

Понятие “совокупный работник” охватывает две группы людей, для обозначения которых можно использовать условные термины “совокупный проектировщик” и “совокупный исполнитель”. В первую группу входят:

1) авторы проектов: исследователи, разработчики и проектировщики, например конструкторы, технологи, математики и программисты;
2) преподаватели, эргономисты, психологи;

3) политики, чиновники, руководители корпораций, ведомств, учреждений, регионов и правительств, которые принимают решение о реализации проекта и его размещении на той или иной территории.

Во вторую группу входят производственники, воплощающие замысел ученых и разработчиков в социальном проекте или “в металле”, строители и монтажники, а также операторы и ремонтники, занимающиеся эксплуатацией и обслуживанием созданного объекта.

Проведенный анализ позволяет ввести следующую систему постулатов.

А. Недопустимо проектировать технические объекты и социальные системы таким образом, чтобы ошибочные действия исполнителей могли привести к серьезной аварии или инциденту.

Б. Если постулат А нарушается, значит, совокупный проектировщик допустил ошибку. Иными словами, сама возможность того, что ошибочные действия исполнителей могут привести к серьезной неприятности, свидетельствует об ошибке проектировщиков.

В. При осуществлении нововведений следует использовать такие методы, при которых безопасность населения, экономики и природы обеспечивается гарантированно (или с высокой вероятностью) независимо от ошибок исполнителей и проектировщиков.

Главное зло — плохо спроектированная деятельность творческого персонала

Таким образом, если при поверхностном анализе в центре внимания находится ошибка исполнителя, то при глубинном анализе в фокус исследования помещается ошибка проектировщиков, которые не сумели предвидеть и предотвратить ошибку исполнителя.

Если при поверхностном анализе исследуются условия и средства труда, а также психофизиологические и иные характеристики исполнителя (например, оператора АЭС), которые привели к его ошибке, то при глубинном анализе исследуются условия и средства труда, а также психофизиологические и иные характеристики проектировщиков, которые повлекли за собой ошибку совокупного проектировщика. При этом факт неумения предвидеть и предотвратить ошибку исполнителя рассматривается как ошибка совокупного проектировщика.

Нетрудно сообразить, что переход от поверхностного анализа к глубинному, от проектирования деятельности исполнителей к проектированию деятельности проектировщиков (т. е. ученых, конструкторов, технологов, программистов, математиков, эргономистов, руководителей, преподавателей и т. д.) означает кардинальное изменение традиционного подхода к разработке фундаментальных принципов обеспечения эффективности и безопасности. Преимущество нового подхода состоит в том, что он позволяет сделать деятельность творческого персонала (проектировщиков) и эксплуатационного персонала (операторов) удовлетворяющей критерию гарантоспособности и за счет этого обеспечить качественно новый уровень безопасности и эффективности цивилизованного мира.

Отсюда проистекает вывод: при любых промышленных авариях и социальных катастрофах корень зла — не в плохой технике, не в плохой идеологии, политике и технологии, не в плохой организации и не в ошибках низового персонала (все это следствия, а не причина), а в плохо спроектированной человеческой деятельности творческих работников. При подобной постановке вопроса (а она, как нетрудно сообразить, коренным образом отличается от традиционной) перед наукой ставится принципиально новая задача: опираясь на достижения всего комплекса наук о человеке разработать интегральную теорию человека, человеческого мозга и человеческого интеллекта, новые более эффективные методы улучшения работы ума и на их основе создать теорию проектирования гарантоспособной деятельности творческого персонала.

Необходимость разработки указанной теории вызвана резким усложнением деятельности. Если раньше творческие работники могли действовать на основе своих знаний, опыта, интуиции и здравого смысла, то теперь — в условиях беспрецедентного усложнения цивилизационных процессов и вызванного им беспрецедентного увеличения цены ошибок творческого персонала — в этих условиях традиционные подходы к организации творческой деятельности человечества следует признать недостаточными, устаревшими, крайне опасными и неприемлемыми.

СОН РАЗУМА РОЖДАЕТ ЧУДОВИЩ

Гарантоспособность коллективной деятельности людей существенно зависит от их коллективного разума, причем этот вывод справедлив как в локальном, так и в глобальном масштабе.

В локальном масштабе факт чернобыльской аварии свидетельствует о неэффективности коллективного интеллекта участников создания и эксплуатации чернобыльской АЭС, об их неспособности предвидеть и предотвратить отрицательные последствия собственных действий. В глобальном масштабе чернобыльская авария свидетельствует о слабости всемирного коллективного разума (т. е. совокупного интеллекта всего человечества), о его неспособности заблаговременно выявить угрозу, понять уязвимость и условность национальных границ перед лицом радиационной опасности, о неспособности объединить усилия, создать необходимые гарантии и защитить планету от отрицательных последствий научно-технического прогресса.

Эффективность коллективного разума ученых, специалистов и чиновников, объединенных в большую совокупность международных и национальных творческих и иных коллективов, состоящую из многих научных, конструкторских, технологических, проектных, производственных, управленческих, надзорных, учебных и прочих организаций, решающих общую задачу, зависит не только от индивидуального интеллекта людей, но и от умения преодолеть раздробленность индивидуальных и коллективных интеллектов, от интеллектуального взаимопонимания и эффективности обмена знаниями между людьми и коллективами, от тех механизмов, посредством которых интеллект отдельного специалиста подключается к общим интеллектуальным ресурсам кол-

лектива и отрасли, а также к общим ресурсам мировой науки, т. е. к совокупному интеллекту человечества. При этом важную роль играет надлежащий выбор визуально-языковых средств понимания, представления, формализации и передачи знаний. Если эти средства будут эффективными, эффективность коллективного разума может быть высокой, в противном случае — низкой.

Сегодня жизнь предъявляет новые, значительно более высокие требования к человеческому разуму, интеллекту. Необходимо помнить: сон разума рождает чудовищ. Немощь разума породила чернобыльскую катастрофу.

ИНТЕНСИФИКАЦИЯ ИНТЕЛЛЕКТА И ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Развиваемая в этой книге концепция улучшения работы ума опирается на гипотезу максимизации мозговой продуктивности (см. гл. 3). Исходя из этой гипотезы мы предполагаем, что главное (хотя и не единственное) средство для обеспечения гарантоспособной деятельности и предотвращения будущих технических и социальных чернобылей — совершенствование человеческого интеллекта [17]. Однако некоторые авторы считают проблему улучшения интеллекта некорректной и надуманной, поскольку “за всю историю развития человеческой цивилизации не отмечено сколь-нибудь заметное совершенствование человеческого интеллекта” [18].

На наш взгляд, подобное возражение основывается на недоразумении: ведь мы говорим не о генетически заданных предпосылках развития мышления (геном человека, как предполагают ученые, действительно не меняется на протяжении последних тысячелетий), а о механизмах социального наследования и передачи знаний и связанных с ними приобретенных возможностях интеллекта, которые можно значительно усилить.

Предпринятое нами изучение проблемы улучшения работы ума есть развитие давней научной традиции, в рамках которой в связи с отсутствием устоявшейся терминологии разные ученые используют следующие сходные по смыслу, хотя и не полностью совпадающие понятия и выражения: “расширение способностей ума” *(Декарт)*, “улучшение наших умозаключений”, “облегчение процесса нашего мышления” *(Лейбниц)*; “усиление человеческого мышления” *(Шредер)*, “облегчение мышления и придание ему большей точности и силы” *(Фреге)*, “улучшение понимания” *(Вертгеймер)*, “усиление мыслительных возможностей” *(Брунер)*, “усиление природных психических процессов” *(Верч)*, “увеличение КПД функционирования человеческого мозга”, “возрастание эффективности человеческого мышления”, “усиление знаниепорождающих, творческих функций естественного интеллекта человека”, “качественная интенсификация массового научного творчества” *(Зенкин)*.

Уместно привести мнение академика А. Ершова: “Человек неизмеримо усилит свой интеллект, если сделает частью своей натуры способность планировать свои действия, вырабатывать общие правила и способ их применения к конкретной ситуации, организовывать эти правила в осознанную и выразимую структуру, — одним словом, сделается программистом”. Ясно однако, что задача улучшения работы ума должна

решаться не только применительно к программистам, но и к миллионам других людей.

Известно, что “программист мыслит категориями, которые дает ему в распоряжение язык программирования” [19]. По мнению экспертов, влияние языка “независимо от нашего желания сказывается на нашем способе мышления” [20]. Язык оказывает глубокое воздействие “на навыки мышления и изобретательские способности”, причем “царящий в существующих языках беспорядок” непосредственно отражается на стиле и эффективности труда [19].

Чем определяется тот предел, до которого может усилить свой интеллект программист? Эта грань жестко регламентируется его личным интеллектуальным опытом и характеристиками тех языков, которые он использует в работе. Между тем все без исключения известные языки программирования наряду с многочисленными достоинствами имеют существенный недостаток — это кастовые языки ограниченного применения. Они не в состоянии обеспечить необходимое облагораживающее воздействие на интеллектуальную жизнь общества, преодолеть раздробленность индивидуальных интеллектов и обеспечить необходимое усиление могущества всемирного коллективного разума, соответствующее новым требованиям.

УЛУЧШЕНИЕ РАБОТЫ УМА — ПРОБЛЕМА НОМЕР ОДИН

Наряду с классом языков программирования (которые, разумеется, должны продолжать функционировать в культуре) необходимо создать класс принципиально новых языков, для обозначения которых предлагается термин “суперязыки интеллектуального общения” (для краткости — суперязыки).

Одно из наиболее драматических противоречий нынешней фазы развития цивилизации состоит в следующем. С одной стороны, немощь разума ставит под угрозу судьбу цивилизации, причем наука не имеет ответа на вопрос: как получить необходимый для спасения прирост интеллекта? С другой стороны, громадные интеллектуальные резервы человеческого мозга по-прежнему не используются, потому что люди “эксплуатируют” свой мозг из рук вон плохо, неграмотно, совсем не так, как того требуют его “проектные” (эргономические и нейробиологические) характеристики.

Итак, зачем нужны суперязыки? Чтобы устранить это противоречие, преодолеть нынешний интеллектуальный тупик за счет выявления и научно-обоснованного использования скрытых резервов мозга.

Социальный успех любого искусственного языка, его укорененность в культуре, возможность крупномасштабного расширения сферы его применения и международного признания зависят от общедоступности и полезности языка. Полезность суперязыков определяется тем, что они должны облегчить понимание и взаимопонимание, обеспечить стратегический интеллектуальный прорыв, позволяющий качественным образом увеличить умственную мощь цивилизации.

Можно ли решить подобную задачу в принципе? Строго говоря, доказательный ответ на этот вопрос пока отсутствует. Вместе с тем можно высказать некоторые предположения. Письменный язык — это система

нотаций, а нотация есть “средство мышления” [21]. Анализируя проблему улучшения нотаций, известный английский логик, математик и философ Альфред Норт Уайтхед пишет: “Освобождая мозг от всей необязательной работы, хорошая нотация позволяет ему сосредоточиться на более сложных проблемах и в результате увеличивает умственную мощь цивилизации” [21].

Обобщая вышеизложенное, можно сделать ряд замечаний.

! Способность или неспособность человечества решать жизненно важные глобальные, локальные и иные проблемы прямо зависит от эффективности всемирного разума и его локальных компонентов, от качества средств представления и формализации знаний, языковых средств понимания и взаимопонимания, удовлетворяющих критерию сверхвысокого понимания, эргономическому критерию Декарта.

! Чернобыль и другие инциденты со всей ясностью обнажили беспомощность нынешних форм планетарного интеллекта и поставили вопрос о разработке нового поколения интеллектуальных средств — суперязыков, способных обеспечить качественно новый уровень человеческого ума.

! Традиционные технократические методы разработки крупномасштабных промышленных объектов и социальных нововведений полностью исчерпали свои возможности, устарели и должны уступить место новым социотехническим методам, которые подразумевают системный подход к проектированию социосферы, техносферы и гарантоспособной человеческой деятельности.

ВЫВОДЫ

1. Языки программирования играют заметную роль в человеческой культуре, являясь составным элементом компьютерной революции, которая, в свою очередь, есть необходимое условие перехода к информационному обществу. Наряду с этим все больше ощущается потребность в суперязыках, которые предназначены для кардинального улучшения работы ума, облегчения понимания и взаимопонимания, обеспечения более эффективного интеллектуального взаимодействия между людьми и в конечном итоге — улучшения человеческой деятельности (как теоретической, так и практической).

2. Некоторые суперязыки могут выполнять функции языков программирования, другие — нет. В роли суперязыков могут выступать только эргономичные визуальные языки.

3. Проблема создания суперязыков приобрела особую актуальность в последнее время, когда выявилась жесткая связь между интеллектом и выживанием и несостоятельность традиционных методов интеллектуальной работы.

4. Выяснилось также, что языки программирования не могут обеспечить требуемый прирост интеллектуальной силы человеческих коллективов (локальных и глобальных). Из-за своего кастового характера и узости социальной базы языки программирования очень слабо влияют на улучшение коллективной работы ума в масштабах общества, на рост могущества социального интеллекта. Фактически они продемонстрировали свою непригодность для решения этой задачи.

— Почему у него такой странный язык?

— ДРАКОН — это не язык. Это суперязык!

5. Суперязыки должны сделать то, чего не могут языки программирования: стимулировать стратегический интеллектуальный прорыв, в максимально возможной степени увеличить умственную мощь человечества, обеспечить качественно новый уровень понимания и взаимопонимания и на этой основе — гарантоспособность планетарной и локальной человеческой деятельности.

6. Первым и, к сожалению, пока единственным примером суперязыка является техноязык ДРАКОН. Создание остальных суперязыков — дело будущего. Возможно, они будут появляться в результате эволюции (в сторону большей наглядности, доступности и общезначимости) графических языков новейших модификаций методологии *RAD* и *CASE*-технологий. Возможны и другие варианты. ДРАКОН — это всего лишь начальное звено в цепи суперязыков, которая обязательно должна наращиваться.

7. Язык ДРАКОН — это первый сознательно сделанный эргономический шаг в языковом строительстве. Эргономический — значит, во-первых, нацеленный на улучшение работы ума, во-вторых, опирающийся на всю мощь науки о человеческих факторах — эргономики и когнитивной науки. Сказанное можно резюмировать в форме краткого лозунга: ДРАКОН — это эргономическая революция в “языкостроении”.

8. Использование эргономики в языкостроении открывает перед этой наукой новые вдохновляющие перспективы, выявляет недостаточность нынешнего уровня эргономических знаний и требует дальнейшего развития ее идей.

9. Использование ДРАКОНА в качестве стандарта для описания структуры деятельности (как человеческой, так и делегированной) рассматривается как пробный шаг в направлении решения важнейшей цивилизационной задачи — проектирования гарантоспособной деятельности.

Глава 19

ВОЗМОЖНА ЛИ ЭРГОНОМИЗАЦИЯ МАТЕМАТИКИ?

Ученый, ты объясняешь нам науку, но кто объяснит нам твое объяснение?

Джордж Байрон

ПОЧЕМУ ДЖОН ФОН НЕЙМАН ПРОВАЛИЛСЯ НА ЭКЗАМЕНЕ?

Математика — одна из разновидностей человеческой деятельности, поэтому логично поставить вопрос об эргономизации математики.

Актуальность этой проблемы определяется следующим. Современная наука и система образования немыслимы без математики, но, к сожалению, математика чрезвычайно трудна и продолжает усложняться. Изучение математики требует колоссальных трудозатрат. Однако речь не только об учащихся. Появляющееся новое математическое знание стало столь разветвленным, что охватить его одному человеку или даже коллективу единомышленников (например, известной группе Николас Бурбаки) "физически невозможно". Даже великий Джон фон Нейман признавался, что знает не больше чем $1/3$ корпуса математики и, чтобы проверить себя, попросил С. Улама проэкзаменовать его. Тот вспоминает: "Я устроил ему некоторое подобие докторантского экзамена в различных областях, стараясь выделить те вопросы, на которые он не мог бы ответить. Я нашел пробелы в дифференциальной геометрии, теории чисел, алгебре, где его ответы были неудовлетворительными" [1].

А ведь фон Нейман был выдающимся математиком! Но если трудности испытывают фигуры такого масштаба, с какими же поистине невероятными трудностями сталкивается подавляющее большинство математиков? Недаром говорят, что "в условиях современной математики математик не знает, как это ни парадоксально, математики" [1]. Что же тогда можно сказать о математических знаниях нематематиков и многочисленных армиях студентов?

Проблема в том, что для большинства нематематических специальностей требования к математической подготовке постоянно возрастают, поскольку возрастает роль математики в современной науке и технике, теории и практике. В итоге получается, что люди не обладают знаниями, которые необходимы им для эффективного выполнения профессиональных обязанностей. Отпугивающая многих трудность изучения математики и вызванный ею недобор математических знаний неизбежно приводит к снижению интеллектуального потенциала общества, еще более усугубляя интеллектуальный кризис цивилизации.

Попытка "запихнуть" в человеческую голову больше, чем она способна переварить за известный промежуток времени, ведет к экстенсификации учебного процесса, перегрузкам учащихся и педагогов, низкой эффективности обучения. Получается замкнутый круг, из которого в рамках традиционных подходов фактически нет выхода.

СУЩЕСТВУЕТ ЛИ ПРОПАСТЬ МЕЖДУ МАТЕМАТИКОЙ И ЭРГОНОМИКОЙ?

Чтобы преодолеть или хотя бы ослабить названные трудности, необходимо:

- улучшить понимаемость (когнитивное качество) математической литературы и за счет этого облегчить изучение математики;
- улучшить взаимопонимание между специалистами, работающими в разных отраслях и отделах математики (а их свыше трех тысяч [1]), а также в прикладных областях, тесно с нею связанных;
- уменьшить интеллектуальные усилия (интегральные умственные трудозатраты), которые общество расходует на изучение и решение математических проблем (при сохранении заданных стандартов качества).

Легко сообразить, что перечисленные задачи имеют эргономический характер. При разработке принципов эргономизации математики полезно учесть мировой опыт развития и комбинирования математической символики от древнейших времен до наших дней, но самое главное, уяснить, что эргономизация не есть нечто внешнее по отношению к математике. Как раз наоборот, эргономизация — это неотъемлемая внутренняя компонента исторического процесса развития математических идей. Тем не менее остается фактом, что сегодня эргономика и математика почти не имеют точек соприкосновения (применение математических методов в эргономике не в счет). Их разделяет пропасть взаимного непонимания. Другой удивительный факт состоит в том, что математики на протяжении всей истории применяли и применяют эргономические методы. Однако этот процесс носил и носит неосознанный, стихийный и кустарный характер, ощутимо снижая когнитивное качество математических текстов, ухудшая понимание и взаимопонимание в математическом сообществе.

Грубо говоря, эргономическая беспечность (а если отбросить деликатность — эргономическая неграмотность) традиционных математических методов представляет собой одну из причин неоправданно высокого интеллектуального барьера, окружающего величественный дворец математической науки. Для обозначения этого барьера появился даже специальный термин — математический терроризм [2].

Мы исходим из того, что в наш междисциплинарный век пренебрежение к эргономическим методам следует признать устаревшим. К тому же оно вступает в противоречие с уроками истории. Чтобы сделать эту мысль более наглядной, проиллюстрируем ее на примере развития правил записи алгебраических уравнений, выбрав в качестве точки отсчета алгебру Диофанта.

АЛГЕБРА ДИОФАНТА

Греческий математик Диофант (III век нашей эры) был последним великим математиком античности. До него уравнения решались в словесной либо в геометрической форме. Он впервые использовал в алгебре буквенную символику. Но введенные им буквенные обозначения были сокращениями соответствующих математических терминов, а не алгеб-

раическими символами в нашем понимании. Тем не менее новшества Диофанта имели революционный характер. Он ввел обозначения неизвестной величины, ее первых шести положительных и отрицательных степеней, отрицательные числа, знак “минус” и знак “равенство”. Благодаря этому Диофанту удалось получить ряд важных результатов. Он даже сумел предвосхитить подстановки Эйлера [3, 4]. Однако современному читателю символика Диофанта кажется странной, запутанной и очень непривычной.

Попробуем запастись терпением и совершим небольшую прогулку по джунглям диофантовой алгебры, ограничившись одним-единственным вопросом: как Диофант записывал уравнения?

Прежде всего, попытаемся выяснить: как будут выглядеть наши современные знаки

$$x,\ x^2,\ x^3,\ x^4,\ x^5,\ x^6,\ =,\ -,\ +\ ,$$

если их перевести на язык Диофанта? Ответ дает табл. 8.

Таблица 8

Современная алгебра		**Алгебра Диофанта**	
Обозначение	**Как читается**	**Обозначение**	**Как читается**
x	Неизвестная величина	ζ	Число
x^2	Икс квадрат	Δ^{v}	Квадрат
x^3	Икс куб	K^{v}	Куб
x^4	Икс в четвертой степени	$\Delta\Delta^{\text{v}}$	Квадратоквадрат
x^5	Икс в пятой степени	$\Delta\text{K}^{\text{v}}$	Квадратокуб
x^6	Икс в шестой степени	KK^{v}	Кубокуб
=	Равно	ι	Равно
–	Минус	⅄	Недостаток
+	Плюс	Плюс у Диофанта отсутствует. Слагаемые пишутся подряд	

Таблица 9

Единицы		Десятки		Сотни		Тысячи	
Современная запись	Греческая запись	Современная запись	Греческая запись	Современная запись	Греческая запись	Современная запись	Греческая запись
1	$\overline{\alpha}$	10	$\overline{\iota}$	100	$\overline{\rho}$	1000	$,\overline{\alpha}$
2	$\overline{\beta}$	20	$\overline{\chi}$	200	$\overline{\sigma}$	2000	$,\overline{\beta}$
3	$\overline{\gamma}$	30	$\overline{\lambda}$	300	$\overline{\tau}$	3000	$,\overline{\gamma}$
4	$\overline{\delta}$	40	$\overline{\mu}$	400	$\overline{\upsilon}$	4000	$,\overline{\delta}$
5	$\overline{\varepsilon}$	50	$\overline{\nu}$	500	$\overline{\varphi}$	5000	$,\overline{\varepsilon}$
6	$\overline{\varsigma}$	60	$\overline{\xi}$	600	$\overline{\chi}$	6000	$,\overline{\varsigma}$
7	$\overline{\zeta}$	70	$\overline{o}$	700	$\overline{\psi}$	7000	$,\overline{\zeta}$
8	$\overline{\eta}$	80	$\overline{\pi}$	800	$\overline{\omega}$	8000	$,\overline{\eta}$
9	$\overline{\vartheta}$	90	$\overline{q}$	900	$\overline{\lambda}$	9000	$,\overline{\vartheta}$

Примечание: в греческой системе счисления нуль отсутствует.

Для обозначения чисел Диофант использует общепринятую у греков символику [5] (табл. 9).

Греческие числа образуют непозиционную десятичную систему: как $\overline{\iota}\overline{\delta}$, так и $\overline{\delta}\overline{\iota}$ может значить только 14. Отсутствие позиционной системы вносит большие когнитивные затруднения: задача $4 \times 10 = 40$ ($\overline{\delta} \times \overline{\iota} = \overline{\mu}$) у греков не имеет ничего общего с “родственной” задачей $4 \times 100 = 400$ ($\overline{\delta} \times \overline{\rho} = \overline{\upsilon}$). По этому поводу немецкий психолог Фридхарт Кликс восклицает: “Насколько проще метод нашей позиционной системы, насколько меньше затрат при решении одинаковых задач!” [5].

Каждый член уравнения Диофант пишет так: сначала указывается степень неизвестного, потом числовой коэффициент; перед отрицательными членами ставится знак недостатка $\curlywedge$ (по-нашему, минус), перед положительным — ничего (плюсом Диофант не пользуется) (табл. 10).

При записи свободного члена уравнения Диофант сначала ставит символ М° (признак свободного члена), а затем его числовое значение (табл. 11).

Таблица 10

Запись членов уравнения			
Современная	**У Диофанта**	**Современная**	**У Диофанта**
x	$\zeta\,\overline{\alpha}$	x^3	$\mathrm{K}^{\nu}\,\overline{\alpha}$
$2x$	$\zeta\,\overline{\beta}$	$2x^3$	$\mathrm{K}^{\nu}\,\overline{\beta}$
$3x$	$\zeta\,\overline{\gamma}$	$3x^3$	$\mathrm{K}^{\nu}\,\overline{\gamma}$
x^2	$\Delta^{\nu}\,\overline{\alpha}$	x^4	$\Delta\Delta^{\nu}\,\overline{\alpha}$
$2x^2$	$\Delta^{\nu}\,\overline{\beta}$	$2x^4$	$\Delta\Delta^{\nu}\,\overline{\beta}$
$3x^2$	$\Delta^{\nu}\,\overline{\gamma}$	$3x^4$	$\Delta\Delta^{\nu}\,\overline{\gamma}$
$-3x^2$	$\pitchfork\,\Delta^{\nu}\,\overline{\gamma}$	$-3x^4$	$\pitchfork\,\Delta\Delta^{\nu}\,\overline{\gamma}$

Таблица 11

Запись свободного члена	
Современная	**У Диофанта**
0	У греков нуля нет
1	$\mathrm{M}^{\mathrm{o}}\,\overline{\alpha}$
2	$\mathrm{M}^{\mathrm{o}}\,\overline{\beta}$
3	$\mathrm{M}^{\mathrm{o}}\,\overline{\gamma}$

С учетом сказанного приведем несколько примеров записи уравнений в символике Диофанта. Обратите внимание: чтобы избавиться от нуля, во втором и третьем примерах Диофанту пришлось перенести свободный член в правую часть уравнения (табл. 12).

Таблица 12

Современная запись уравнения	Запись уравнения у Диофанта
$2x = 4$	$\overline{\zeta}\,\overline{\beta}\,\iota\,\mathrm{M}^{\mathrm{o}}\,\overline{\delta}$
$x^2 + 3 = 0$	$\pitchfork\,\Delta^{\nu}\,\overline{\alpha}\,\iota\quad\mathrm{M}^{\mathrm{o}}\,\overline{\gamma}$
$3x^2 - 5x + 2 = 0$	$\pitchfork\Delta^{\nu}\,\overline{\gamma}\,\pitchfork\zeta\,\overline{\varepsilon}\,\iota\quad\mathrm{M}^{\mathrm{o}}\,\overline{\beta}$
$2x^4 + 4x = 1$	$\Delta\Delta^{\nu}\,\overline{\beta}\,\zeta\,\overline{\delta}\,\iota\,\mathrm{M}^{\mathrm{o}}\,\overline{\alpha}$
$x^3 + 2x - (5x^2 + 1) = x$	$\mathrm{K}^{\nu}\,\overline{\alpha}\,\zeta\pitchfork\overline{\beta}\quad\Delta^{\nu}\,\overline{\varepsilon}\,\mathrm{M}^{\mathrm{o}}\,\overline{\alpha}\,\iota\,\zeta\,\overline{\alpha}$

ЭРГОНОМИЧЕСКИЙ АНАЛИЗ АЛГЕБРЫ ДИОФАНТА

Попытаемся оценить способ записи уравнений, использованный Диофантом, с точки зрения современных когнитивных представлений. Не

претендуя на составление полной “дефектной ведомости”, укажем лишь несколько эргономических недостатков диофантовой алгебры.

1. Когда человек смотрит на запись уравнения, он должен решить несколько визуальных задач, обеспечивающих правильное зрительное восприятие и понимание сути дела, причем сделать это “без излишней траты умственных сил”. Первая задача — разбить зрительную сцену на два главных смысловых блока (левую и правую части уравнения). Для этого нужно выполнить три зрительных операции:
   - ! бегло просмотрев все символы уравнения, найти среди них знак равенства;
   - ! сделать вывод: зрительная зона, находящаяся левее этого знака, есть левая часть уравнения;
   - ! сделать вывод: зона, лежащая правее, есть правая часть.

   Современный знак равенства “=” с пробелами до и после него имеет удачную эргономическую конфигурацию, как нельзя лучше подходящую для этой цели. По форме он резко отличается от других знаков и эффектно “разламывает” уравнение на две части, благодаря чему человеческий глаз моментально решает первую визуальную задачу. Что касается алгебры Диофанта, то греческая буква иота ι, используемая как знак равенства, слишком невзрачна и невыразительна — она теряется среди других букв. Из-за этого зрительное выделение и опознание двух частей равенства заметно усложняется.

2. Следующая визуальная задача — разбить каждую часть уравнения на смысловые блоки, т. е. выделить и опознать *члены уравнения*. Сегодня мы решаем эту задачу, фиксируя взором зрительные зоны, расположенные между знаками сложения и вычитания. Последние, выполняя функцию разделителей, моментально расчленяют уравнение, превращая его в гирлянду, состоящую из отдельных членов, “склеенных” с помощью символов + и –. У Диофанта дело обстоит гораздо хуже. Использование знака $\curlywedge$ в качестве минуса трудно признать удачным. А полное отсутствие знака “плюс” — это уже бедствие, которое приводит к визуальной неоднозначности.

   В самом деле, современное выражение $x^2 + 3x$ по Диофанту записывается так:

   $$\Delta^{\upsilon}\,\overline{\alpha}\,\zeta\,\overline{\gamma}$$

   В последней формуле присутствуют три пары из двух смежных букв. Все они трактуются по-разному, что создает неоправданную нагрузку на память человека:
   - ! запись $\Delta^{\upsilon}\,\overline{\alpha}$ обозначает x^2;
   - ! запись $\overline{\alpha}\,\zeta$ обозначает операцию сложения, причем в записи складываются члены $(\Delta^{\upsilon}\,\overline{\alpha}) + (\zeta\,\overline{\gamma})$;
   - ! запись $\zeta\,\overline{\gamma}$ обозначает умножение $x \times 3$.

   Столь сложные правила расшифровки зрительной сцены вносят дополнительные и никому не нужные трудности в процесс зрительного восприятия.

3. Три класса объектов (неизвестные, цифры и операции) имеют сходные обозначения (всюду — буквы), что затрудняет визуальное различение указанных классов и кристаллизацию соответствующих понятий.
4. Не существует обозначений для нуля, плюса и степеней неизвестной выше шестой.
5. Степени неизвестной величины обозначаются не путем вариации одного знака (x, x^2, x^3, x^4, x^5, x^6), но имеют неоправданные различия и отсутствие общих элементов. В итоге нет визуальной подсказки, облегчающей опознание и выявление "родственных черт" у сходных понятий (степеней).
6. Отсутствует единая символика для чисел и показателей степени. Например, цифра 3 обозначается через $\overline{\gamma}$, а x^3 — через $\text{K}^{\text{v}}\overline{\alpha}$, но в последнем выражении цифра 3 отсутствует.

Проведенный анализ показывает, что по эргономическим показателям символика Диофанта значительно уступает современным обозначениям.

ЭРГОНОМИЗАЦИЯ АЛГЕБРЫ ПОСЛЕ ДИОФАНТА

Алгебра Диофанта для своего времени была крупнейшим достижением. Однако ее постигла печальная участь. Оригинал на греческом языке в Европе был утерян и вновь найден только в XV веке [4]. К счастью, идеи Диофанта получили распространение в арабском мире и вернулись в Европу кружным путем — через арабское культурное влияние. При этом многое было утеряно и переоткрыто заново.

Важным этапом на пути эргономизации европейской алгебры была замена греческих цифр на арабские. Самый древний европейский манускрипт, содержащий эти десять цифр, — "Вигиланский кодекс", написанный в Испании в 976 г. Однако по ряду причин, которых мы не касаемся, арабская система счисления распространялась в Европе очень медленно и стала общепринятой лишь к 1500 г. [6].

К сожалению, имело место и попятное движение. Целый ряд эргономических находок Диофанта на несколько столетий был забыт. В частности, оказалась утраченной его буквенная символика, уступившая место словесным описаниям. Например, Леонардо Пизанский (1180—1240) называет неизвестную *res* (вещь) или *radix* (корень), квадрат неизвестной — *census* (имущество) или *quadratus* (квадрат); данное число — *numerus*. Все это — латинские переводы соответствующих арабских слов [6].

Наваждение слов преследовало математиков многие века. Поскольку не существовало символов даже для самых простых арифметических действий, каждый автор по-своему записывал сложение и вычитание, возведение в степень и извлечение корня. Требовалось немало усилий, чтобы разобраться в таком сложном переплетении форм записи. Поэтому ученые доверяли больше словам, чем знакам, тяготели больше к словесным описаниям, чем к формулам.

Однако мало-помалу формульная запись начала пробивать себе дорогу. Но, Боже, что это были за формулы! Например, уравнение $\sqrt{4x^2+4x}+2x+1=100$ записывалось в XV веке во Франции таким образом [7]:

$R^2\,4^2\,\tilde{p}\,4^1\,\tilde{p}\,2^1\,\tilde{p}\,1$ exaluxa 100

Смысл уравнения ясен из переводной табл. 13.

Обратите внимание: в данном случае явное обозначение для неизвестной величины отсутствует — она в уравнении не записывается, а только подразумевается! Это серьезный эргономический недостаток. Есть и другой дефект: окончание подкоренного выражения никак не обозначено — его надо запоминать, что создает неоправданную нагрузку на память читателя.

Таблица 13

Французское обозначение XV века	**Современное обозначение**
R^2	$\sqrt{\ }$
4^2	$4x^2$
$\tilde{p}$	+
4^1	$4x$
2^1	$2x$
1	1
exaluxa	=
100	100

Однако двинемся дальше. В 1489 г. в учебнике арифметики Яна Видмана впервые в печатном издании появились символы + и −, введенные чуть раньше немецкими алгебраистами (коссистами).

Франсуа Виет (1540—1603) придумал знаки для произвольных величин, называемых сегодня параметрами, и предложил правило: неизвестные обозначать гласными буквами, а параметры — согласными (табл. 14).

Таблица 14

Уравнение в записи Виета	Современная запись уравнения
A cubus + *B* planum in *A*7 aequatur *C* solidum	$x^3 + 7Bx = C$

В табл. 14 planum и solidum — паразитные слова, которые можно безболезненно опустить[1]. Сделав это, получим

A cubus + *B* in *A*7 aequatur *C*

Последнее выражение можно легко понять с помощью переводной табл. 15. Нетрудно видеть, что запись уравнений у Виета неэргономична, громоздка и словообильна.

Таблица 15

Обозначение Виета	Современное обозначение
A	x
A cubus	x^3
in	Знак умножения
aequatur	=
*A*7	$7x$
B in *A*7	$7Bx$
C	C

Тем не менее дело потихоньку двигалось вперед. Английский математик Р. Рекорд (1510—1558) придумал знак равенства =. Томас Гарриот (умер в 1621 г.) сделал следующий эргономический шаг, полностью исключив словесные описания. Уравнение $x^3 - 3bx^2 + 3b^2x = 2b^3$ в записи Гарриота имело “почти современный” вид

$aaa - 3.baa + 3.bba = 2.bbb$

Следующее эргономическое новшество принадлежит Рене Декарту. Он обозначил неизвестные величины буквами x, y, z, а известные — буквами a, b, c и ввел обозначения степеней: x^3, x^4, a^3, a^4. Правда, квадраты он иногда выражал с помощью символов xx, aa. Обозначение корня несколько отличается от современного: знак $\sqrt{c}$ означает у Декарта кубический корень. Есть и другие недостатки: Декарт “испортил” знак равенства, изображая его как $\propto$ [4].

---

[1] Как и древние греки, Виет придерживался правила: сторону можно складывать только со стороной, квадрат — с квадратом, куб — с кубом и т. д. Поэтому для придания уравнению однородности Виет после входящих в него параметров писал *planum* (плоскость), *solidum* (тело) и т. д.

ОСОЗНАНИЕ ПОЛЕЗНОСТИ ЭРГОНОМИЧЕСКОГО ПОВОРОТА В МАТЕМАТИКЕ

Подведем некоторые итоги. Эргономизация алгебры очень важна. Как отмечает голландский историк математики Дирк Стройк, “новые результаты часто становятся возможными лишь благодаря новому способу записи... Подходящее обозначение лучше отображает действительность, чем неудачное, оно оказывается как бы наделенным собственной жизненной силой, которая в свою очередь порождает новое. За усовершенствованием алгебраических обозначений Виета поколение спустя последовало применение алгебры к геометрии у Декарта” [6].

С ним согласен В. Катасонов: «Весь XVI век проходит под знаком настойчивых поисков удобной (читай — эргономичной!) алгебраической символики, которая позволила бы создать некое “исчисление” для решения задач... без излишней траты умственных сил, механически следуя простым правилам».

Суть эргономизации алгебры не только и не столько в том, что она выработала очень компактные и достаточно удобные обозначения, но прежде всего в том, что эргономичные знаки и (что самое главное) их эргономичные комбинации образуют эргономичные зрительные сцены (диосцены), которые точно и строго отражая математическую реальность, вместе с тем гораздо лучше согласуются с нейробиологическими характеристиками человеческого глаза и мозга, что создает важные предпосылки для улучшения творческой продуктивности человеческого ума.

Известно, что “прогресс науки... связан с созданием и совершенствованием символики. В свою очередь, сам этот прогресс зависит от того, насколько компактна и совершенна та знаковая система, которая отображает существенные свойства и черты реальности” [7].

Конечно, каждый отдельно взятый символ не может играть слишком большой роли. Но вся совокупность символов в целом, взаимоотношения между которыми складываются по вполне определенным правилам, — это уже целый язык, новая знаковая система. В таком едином ансамбле знакам присущи новые эргономические и иные свойства, которых не имеет каждый из них в отдельности.

Эргономическая сущность замены словесных описаний на эффективные и компактные алгебраические формулы состоит в том, что происходит “удивительная рационализация визуального поля” [7]. В результате в оптическом поле одномоментного восприятия, визуального охвата оказывается значительно большая информация в удобной и доступной для восприятия форме. Это означает, что имеет место эффект симультанизации, который, как мы знаем, является одним из наиболее мощных рычагов, обеспечивающих ускорение работы мозга. Согласно взглядам Лазаря Карно, математические знаки “не являются только записью мысли, средством ее изображения и закрепления, — нет, они воздействуют на саму мысль, направляют ее и бывает достаточно переместить их на бумаге согласно известным, очень простым правилам, чтобы безошибочно достигнуть новых истин”. Логика знаковых преобразований наталкивает людей “на глубокие, не всегда привычные взаимосвязи явлений” [7].

Кроме того эргономизация алгебры включает в себя все ходы мысли, связанные с дроблением и вычленением новых понятий, доведением их до логического совершенства. Приведем пример. Алгебра Декарта неэргономична еще и потому, что он использовал неудачное определение

понятия “коэффициент”, полагая, что все буквенные коэффициенты в алгебраических формулах положительны. Это ограничение имело вредные последствия. Например, если $p > 0$ и $q > 0$, уравнение в записи Декарта полностью отвечает современным требованиям и является вполне эргономичным (если забыть про неудачный знак равенства):

$$x^4 + px^3 - qx \propto 0$$

Если же знак коэффициента произволен, Декарт вводит странные многоточия [4]:

$$x^4 \cdots px^3 \cdots qx \propto 0$$

Эти никому не нужные многоточия, усложняющие символику, являются следствием неудачного определения понятия “буквенный коэффициент”. Чтобы устранить недостаток, надо улучшить определение, т. е. сделать его более эргономичным. Для этого будем считать, что буквы в уравнениях обозначают любое действительное число[1]. При таком определении нелепые многоточия становятся ненужными и исчезают, превращаясь (как и положено) в знаки + и –. Данный пример показывает, что эргономизация понятий в свою очередь может оказывать благотворное влияние на эргономичность обозначений.

ЭРГОНОМИЧЕСКАЯ ПОБЕДА ЛЕЙБНИЦА

Понятие эргономизации является универсальным: оно применимо не только к алгебре, но и ко многим другим вопросам. Чтобы убедиться в этом, рассмотрим еще один пример, связанный с эргономизацией исчисления бесконечно малых.

Почти одновременное изобретение дифференциального и интегрального исчислений Лейбницем и Ньютоном явилось восхитительным достижением. Однако в противоположность удобным обозначениям Лейбница символика Ньютона оказалась громоздкой и эргономически неудачной. К чему это привело? Как пишет историк математики М. Кроу, “общее мнение британских математиков XIX в. состояло в том, что к 1800 г. континентальные математики далеко превзошли английских главным образом потому, что обозначения Лейбница в математическом анализе оказались лучше Ньютоновых” [8]. Этот факт еще раз подтверждает, что когнитивное качество знаковых систем сильно влияет на интеллектуальную продуктивность.

Ньютон был гением идеи, а Лейбниц — не только гением идеи, но и гением символики. Стремясь построить “всеобщую характеристику” (универсальный язык), он пришел ко многим новшествам в математических обозначениях. Лейбниц — один из самых плодовитых изобретателей математических символов. Немногие так хорошо понимали единство формы и содержания. Именно это внимание к знаку, к формальной стороне метода позволило Лейбницу сформулировать ряд важных положений математического анализа: это и знаменитая формула Лейбница для дифференцирования произведения, и многое другое. Только со времен

[1] Декарт не мог этого сделать, поскольку он не знал действительных чисел. Последнее понятие появляется только в трудах Ньютона, который впервые провел арифметизацию алгебры, окончательно отделив ее от геометрии.

Лейбница в математике стали широко использоваться знаки "·", ":" для умножения и деления. Он же ввел в обиход символы $\log$, $\int$, d для логарифма, интеграла и дифференциала. Символы Лейбница настолько ясно выражали смысл и значение новых понятий, что легко привились и стали общепринятыми [6, 7].

Речь, разумеется, идет не только о выборе одиночных обозначений (буквенных или иных), но прежде всего о правилах построения суперзнаков. Мы используем термин "суперзнак" для обозначения правил комбинирования одиночных символов и правил их взаимного расположения в двумерном поле бумажной страницы, позволяющих получить полезный формальный или смысловой результат. Таким образом, суперзнак — это диосцена либо ее смысловой фрагмент. Примером суперзнака является выражение

$$\int_a^b f(x)\,dx\,,$$

а также любые формулы или цепочки формул, частью которых является указанное выражение[1].

Лейбниц хорошо понимал, что создание эффективных знаков и суперзнаков и пользование ими заключает в себе огромные возможности. Как бы предвосхищая будущие достижения когнитивной эргономики и семиотики, открывающие и облегчающие путь к мощным интеллектуальным прорывам, он пишет: "Общее искусство знаков, или искусство обозначения представляет собой чудесное пособие, так как оно разгружает воображение... Следует заботиться о том, чтобы обозначения были удобны для открытий. Это большей частью бывает, когда обозначения коротко выражают и как бы отображают интимнейшую суть вещей. Тогда поразительным образом сокращается работа мысли".

Поучителен известный спор Лейбница с Вагнером.

Лейбниц. Удобная символика очень много значит для науки.

Возражение Вагнера. Умный справится с любой задачей и без вспомогательных средств, а неумному не помогут никакие руководства.

Ответ Лейбница. Я уверен, что плохая голова, упражняясь в использовании вспомогательных средств [знаков], может превзойти самую лучшую, подобно тому как ребенок может провести линию по линейке лучше, чем самый искусный мастер от руки. Гениальные же умы, снабженные такими преимуществами, пошли бы несравненно дальше.

Школа Лейбница была "гораздо более блестящей, чем школа Ньютона" [6]. Что касается эргономически неудачной символики Ньютона, то она потерпела полный крах. Впрочем, поначалу деканы Кембриджа и Оксфорда рассматривали любую попытку своих студентов использовать обозначения Лейбница как нечестивый бунт против священной тени Ньютона. В итоге ньютонова школа в Англии и школа Лейбница на континенте настолько разошлись, что Леонард Эйлер в своем "Интегральном исчислении" (1768 г.) рассматривал объединение обоих способов записи, как бесполезное. Но эта преграда была сломлена в 1812 г. группой молодых кембриджских математиков, среди которых был и Чарльз

---

[1] Обобщая, можно сказать, что любой рисунок в этой книге, а также его достаточно крупная смысловая часть также является суперзнаком.

Бэббедж, которого иногда называют отцом современной вычислительной техники. Они пытались, говоря словами Бэббеджа, проповедовать принципы чистого *d*-изма[1] в противоположность университетскому *dotage* (игра слов: *dotage* — старческое слабоумие; с другой стороны, *dot* — точка, *age* — эпоха; получается *dotage* — эпоха точек; более чем прозрачный намек на Ньютоновы обозначения производных $\dot{x}, \dot{y}, \dot{z}$) [6]. Справедливости ради добавим, что символика Ньютона полностью не исчезла: с позором изгнанная из математического анализа, она тем не менее иногда применяется в теоретической механике.

МЕТОДОЛОГИЧЕСКАЯ ОШИБКА ИСТОРИКОВ МАТЕМАТИКИ

История математики показывает, что многие разделы этой науки стали успешно разрабатываться только после того, как были введены удобные (эргономичные) знаки, способствующие развитию соответствующих рассуждений и построений. Так, Джузеппе Пеано настаивал на важном значении символического обозначения во всяком математическом предложении, его полезности в трудных и тонких вопросах. Стефен Клини совершенно правильно отмечает: “Открытие простых символических обозначений, которые сами приводят к манипуляциям по формальным правилам, явилось одним из путей, на которых развивалась мощь современной математики”.

Отсюда вытекает, что длительный исторический процесс изобретения, изменения, улучшения и реконструкции математической символики нельзя рассматривать как нечто второстепенное для математических исканий. Данный процесс (эргономический по своей природе) в немалой степени отражает волнующий путь зарождения и развития новых математических идей и понятий, он теснейшим образом связан с сокровенными тайнами математического творчества. Изучение названного эргономического процесса должно стать одним из важных направлений *истории математики*, если она хочет претендовать на статус научной дисциплины, удовлетворяющей современным междисциплинарным требованиям.

К сожалению, в работах по истории науки в целом (и по истории математики в частности) эта сторона дела нередко выпадает из поля зрения ученых. Во многих публикациях господствует “модернизаторский” подход, когда старинные формулы и чертежи грубо искажаются в результате некритического перевода на современный язык. При этом эргономическая драма древних обозначений и понятий полностью ускользает от внимания исследователя. Например, в работе Б. Спасского даже чертеж Галилея, при помощи которого тот решал свою задачу, сильно изменен и упрощен и все изложение ведется на современном языке [9]. Отто Нейгебауэр довольно резко замечает: “Только в последнее время ученые, вслед за А. Ромом и А. Делаттом, начали публиковать тексты вместе с рисунками и обозначениями на них. Кроме этих недавних исключений, ни одному изданию верить нельзя, во всяком случае в отно-

---

[1] Игра слов: деизм — религия разума эпохи Просвещения; с другой стороны, *d*-изм — применение буквы *d*, которую Лейбниц широко использовал в выражениях *dx*, *dy*, *dx*/*dy* и т. д.

шении вида, буквенных обозначений и даже наличия рисунков” [10]. Пример сознательно модернизированного издания — Арифметика Диофанта. Например, символ Диофанта □ (неопределенный квадрат) переделан на понятный современному читателю лад x^2 (переменная x во второй степени) [9].

Модернизированные издания древних математических рукописей позволяют быстро проследить ход решения задач и математических преобразований в современных обозначениях. Однако достигается это слишком дорогой ценой. Важнейшая задача историко-математического исследования — реконструкция математического мировоззрения древних авторов — остается невыполненной, так как отказ от старинных обозначений зачастую до неузнаваемости искажает старинную систему понятий. Другой недостаток состоит в том, что исследователь, упуская из виду междисциплинарный характер стоящей перед ним задачи, как бы “перепрыгивает” через ее самый трудный участок (который можно охарактеризовать как историко-эргономический анализ эволюции математических понятий и символики) и тем самым полностью лишает себя возможности выявить и тщательно исследовать эргономическую суть проблемы. В результате историко-математическое исследование не достигает цели и по крайней мере частично превращается в неумышленную фальшивку.

АНАЛОГИЯ МЕЖДУ МАТЕМАТИЧЕСКОЙ ДИОСЦЕНОЙ И ПАНЕЛЬЮ ОТОБРАЖЕНИЯ ИНФОРМАЦИИ

Вспомним некоторые сведения из “классической” эргономики.

Рассмотрим работу оператора, управляющего реакторным отделением атомной электростанции. Оператор трудится за пультом управления, на котором — наряду с другим оборудованием — размещается комплекс средств отображения информации, обеспечивающий выдачу оператору итоговой информации о работе реактора в виде, удобном для использования. *Средства отображения информации* представляют оператору возможность работать с информационной моделью объекта (реактора), которая должна быть адекватна реальной ситуации и соответствовать закономерностям и характеристикам человеческого восприятия, памяти, мышления. Модель должна быть наглядной, чтобы позволить человеку-оператору понять суть проблемной ситуации быстро, без трудоемкого анализа. При этом информация должна предъявляться человеку в “разжеванном виде” и не требовать от него дополнительного перекодирования в более понятную форму [11].

На основании теоретического анализа и обширного практического опыта специалисты по эргономике разработали многочисленные и весьма ценные правила, которым должна удовлетворять форма представления информации для человека-оператора [11, 12].

А теперь зададим вопрос: есть ли что-либо общее у двух, казалось бы, столь непохожих объектов, как панель отображения информации и страница математического текста? Оказывается, есть, причем можно указать четыре общих свойства. Во-первых, и панель индикации, и математический текст представляют собой диосцены. Во-вторых, они

несут информацию, предназначенную для зрительного восприятия. В-третьих, целью восприятия является понимание важной информации человеком. Наконец, в-четвертых, крайне желательно представить информацию в таком виде, чтобы человек мог достичь понимания за минимальное время ценою минимальных интеллектуальных усилий в соответствии с критерием Декарта.

Исходя из сказанного, можно сделать четыре вывода.

- Система "математик — математический текст" в определенном отношении похожа на систему "оператор — средства отображения информации".
- Математический текст и отображаемая на пульте информация являются аналогами, ибо представляют собой разные формы кодирования оптической информации, предназначенной для зрительного восприятия.
- Чтобы улучшить понимаемость математического текста, следует попытаться использовать разработанные в инженерной психологии эргономические правила, применяемые при проектировании средств отображения информации.
- В тех случаях, когда указанные правила "не работают", следует доработать и улучшить когнитивно-эргономическую теорию, расширив ее возможности применительно к проектированию интересующих нас систем "человек—знание" (рис. 139).

МАТЕМАТИЧЕСКАЯ И ЭРГОНОМИЧЕСКАЯ ЭФФЕКТИВНОСТЬ

Вся математическая литература по определению представляет собой диоинформацию. Математические знания, которые мы получаем при помощи компьютерного экрана, принтера и плоттера, — это тоже диоинформация. Таким образом, в 99% случаев человек получает математические знания в форме диоинформации. Учитывая этот факт и опираясь на положения нового когнитивного подхода (см. гл. 5), можно предложить семь тезисов.

Тезис 1. В подавляющем большинстве случаев математические идеи, теории и методы (за исключением тривиальных) становятся достоянием математического сообщества после представления их в письменном виде, предназначенном для визуального восприятия, т. е. после того, как математическая мысль приобрела форму диоинформации.

Тезис 2. Математическая диоинформация — совокупность диосцен, каждую из которых можно рассматривать как суперзнак. Суперзнак — осмысленная двумерная комбинация знаков, целиком находящаяся в поле зрения.

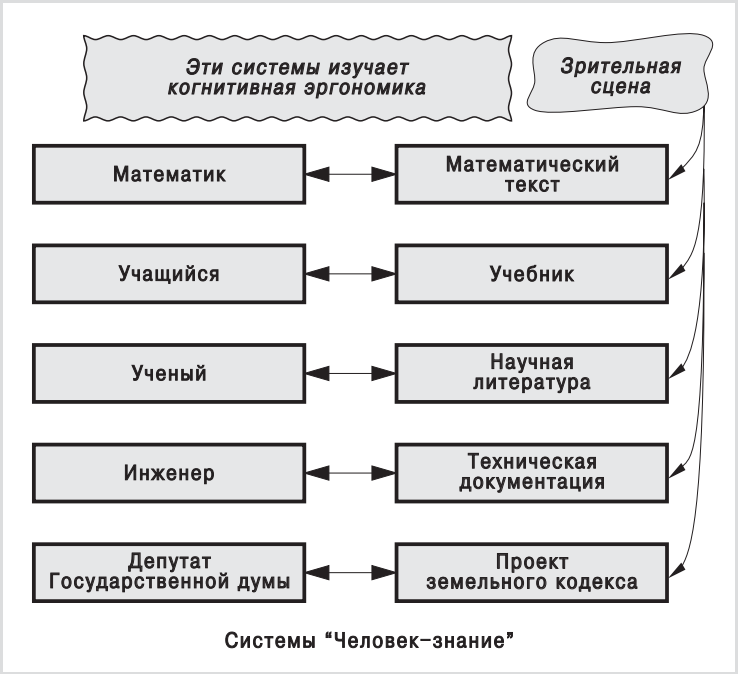
Тезис 3. Математическая диосцена обладает двумя фундаментальными свойствами. Во-первых, она несет определенное смысловое содержание и отображает математическую реальность. Во-вторых, это оптическая зрительная сцена, предназначенная для зрительного восприятия человеком.

Тезис 4. Отсюда вытекает, что форма математической диосцены есть объект двойного назначения. Во-первых, отражая математическую реальность, она должна обеспечить выполнение формальных операций со знаками, преобразование знаков по определенным правилам, позволяющим получить полезный математический результат. Во-вторых, именно форма знаков и суперзнаков позволяет получить конечный результат зрительного восприятия, понимание человеком сущности математических идей и преобразований.

Тезис 5. Разрушение или искажение формы приводит к двум "катастрофам": 1) математическое содержание пропадает, превращаясь в ничто, в бессмыслицу; 2) осмысленное восприятие человеком математической диосцены рассыпается, понимание исчезает, а сама диосцена воспринимается как абсурдный набор бессодержательных пятен и линий.

Система “человек–знание” аналогична системе “человек–техника”. В обоих случаях информация, содержащаяся в зрительной сцене (диосцене), поступает через зрительный аппарат человека в его мозг.

Система “Человек – техника”



Системы “Человек–знание”

Рис. 139. **Чтобы спроектировать эффективную систему “человек — знание”, необходимо видоизменить и улучшить диосцену, т. е. согласовать характеристики диосцены с характеристиками человеческого глаза и мозга**

Тезис 6. Следует различать два понятия: 1) математическая эффективность диосцены; 2) эргономическая эффективность диосцены.

Первая имеет место, если диосцены и связанные с ними математические идеи и знаковые системы позволяют получить полезный математический результат, удовлетворяющий критерию математической строгости и другим разумным математическим критериям.

Эргономическая эффективность имеет место, если удовлетворяется эргономический критерий Декарта, т. е. если человек может ценою минимальных интеллектуальных усилий либо воспринять и усвоить математическое содержание последовательности диосцен (если речь идет об изучении математического материала), либо с помощью указанных диосцен решить соответствующую математическую задачу (если речь идет не об изучении, а о решении новых задач).

Тезис 7. Органический, принципиально неустранимый дефект понятия математической эффективности состоит в том, что оно почти полностью игнорирует проблему понимаемости математических текстов и связанный с нею критерий Декарта. Если на земном шаре отыщется пара суперматематиков, способных понять новую сложную математическую идею, этого вполне достаточно, чтобы дать ей путевку в жизнь. (При этом "мучения" всех остальных специалистов и студентов, связанные с трудностями понимания, категорически не принимаются в расчет.)

Понятие эргономической эффективности математики вводится для того, чтобы облегчить и упростить процесс понимания сложных математических проблем, создать научную основу для введения понятия "производительность математического труда" и на этой основе обеспечить реальное повышение продуктивности работы ума математиков, а также миллионов людей, изучающих эту сложную науку.

Утрируя, можно предложить вымышленный диалог.

— Почему математика такая трудная?

— Потому что до сих пор никто не пытался создать научно-обоснованный метод, позволяющий сделать ее более легкой.

КАК ПОВЫСИТЬ ПРОИЗВОДИТЕЛЬНОСТЬ МАТЕМАТИЧЕСКОГО ТРУДА?

Развитие математики и логики увенчалось созданием богатого набора формальных правил, использование которых позволяет решать обширный класс математических и логических задач. При разработке этих правил математики преследовали две цели. Первая цель была основной и явно выраженной: обогатить математическое знание. Вторая цель состояла в том, чтобы сделать знаковые системы и математические преобразования по возможности удобными и обозримыми. Эта вторая (эргономическая) цель была скорее интуитивной, чем осознанной и научно обоснованной.

В настоящее время, когда сложность математических знаний превысила "критический порог", настало время коренным образом изменить подход к проблеме. В математике следует выделить два набора правил:

- традиционную математическую и логико-математическую формализацию, обеспечивающую достижение математической эффективности;
- набор эргономических формальных правил, цель которых — добиться эргономической эффективности математических методов.

Необходимо “уравнять в правах” и объединить оба набора. В итоге получим единый набор формальных правил, для обозначения которых выше предложен термин “когнитивная формализация знаний”. Совместное применение правил позволит получить удвоенный выигрыш.

Суть вопроса в том, что повышение производительности математического труда, понимаемое как повышение продуктивности человеческого мозга (мозга математиков), — не столько математическая, сколько эргономическая проблема. Человеческий ум и методы улучшения его работы — предмет изучения не математики, а когнитивной эргономики. Эту мысль следует подчеркнуть особо: ***эргономический выигрыш в математике есть не что иное как повышение производительности математического труда***.

Когнитивная формализация как научная идея, направленная на улучшение работы человеческого ума, — это новорожденный младенец, делающий самые первые шаги, но которому, по нашему мнению, суждено большое будущее.

ДВА МЕТОДА ВИЗУАЛИЗАЦИИ МАТЕМАТИКИ

Математический текст в общем случае строится из трех “строительных блоков”:

Диоматематика	=	Словесный текст	+	Формулы	+	Изображения

История математики показывает, что по мере развития математических знаний часть словесного текста постепенно заменяется формулами и изображениями. Указанная замена представляет собой один из важных аспектов эргономизации математики, так как при этом сукцессивное (медленное) восприятие текста заменяется симультанным (быстрым) восприятием формул и изображений.

Важным, хотя и не единственным методом математической эргономизации следует признать визуализацию (замену текста изображением), которая, как отмечает С. Клименко, призвана “делать видимым невидимое”.

Повышение интереса к визуализации породило в США инициативу *VISC (Visualization in Scientific Computing)*, что означает “визуализация в научных вычислениях” [12].

Визуализация математики — это обширная область исследований, охватывающая большое число разнообразных способов и приемов. Среди них выделим два, которые назовем формальным и неформальным методами.

Примером формального метода визуализации является визуальный синтаксис языка ДРАКОН. В самом деле, сравнивая левую и правую

части на рис. 90, 91, легко убедиться, что текстовые и визуальные формулы, во-первых, являются строго формальными, во-вторых, эквивалентны друг другу.

Другим примером формального подхода служит развитие визуального программирования в *CASE*-технологиях и компьютерных методологиях, а также когнитивной компьютерной графики. В последнем случае используются так называемые "когнитивные изображения", позволяющие показать "внутреннее содержание, идею, суть изображаемого оригинала, которым может быть любое абстрактное научное понятие, гипотеза или теория" [13]. По мнению А. Зенкина, использовавшего идеи когнитивной графики для формального и весьма плодотворного исследования хорошо известной в математике классической проблемы Варинга и ряда других вопросов, этот метод дает возможность "прямого воздействия на сам процесс интуитивного образного мышления исследователя", в связи с чем "эффективность человеческого мышления, прежде всего в процессе научного познания, способна возрасти уже на многие порядки" [13].

Формальные методы визуализации математики обладают чрезвычайно большим творческим потенциалом. Вместе с тем они имеют очевидный недостаток: каждый такой метод является "штучным произведением искусства", он появляется на свет в результате индивидуального творческого акта и — в общем случае — не дает никаких прямых указаний, никакой явной подсказки, позволяющей поставить дело на поток и "штамповать" подобные изобретения в массовом порядке на воображаемом "математическом конвейере". От этого недостатка свободен неформальный метод математической визуализации, о котором пойдет речь ниже.

ПРОЕКТ "КОГНИТИВНЫЙ СТИЛЬ" *(COGNISTYLE)*

Было бы крайне желательно предложить простой и универсальный визуально-эргономический метод, позволяющий сделать более понятной и наглядной любую или почти любую сложную абстрактную математическую идею или задачу.

Одним из возможных подходов к проблеме является метод *CogniStyle*. Согласно предварительному, пока еще не завершенному проекту он представляет собой универсальный метод визуализации научной и учебной литературы, который, в частности, можно применить для визуализации произвольных математических текстов. Все без исключения рисунки в этой книге спроектированы методом *CogniStyle*.

Для примера перечислим несколько наиболее простых эргономических приемов из набора *CogniStyle*.

- Визуальная очная ставка: рядом помещаются два рисунка, сравнивая которые нужно выявить сходство и отличие признаков (рис. 3).
- Тройная подпись: общая нижняя надпись и два прямоугольных "подвала" (рис. 3).
- Выделение группы графоэлементов с помощью тонирования (рис. 3*б*).
- Инверсное тонирование группы и входящих в нее элементов, чтобы подчеркнуть иерархическое строение группы (рис. 3*б*).

- Облако со стрелкой, идентифицирующее группу элементов (рис. 3*б*).
- Использование прямоугольных контуров с номерами объектов и стрелками (рис. 5).
- Вариация толщины соединительной линии (рис. 8).
- Два облака, указывающие разные свойства одной и той же линии (рис. 8).
- Пояснения в волнистом прямоугольнике без стрелки (рис. 9, сверху).
- Оценки ситуаций в прямоугольниках со стрелками (рис. 9, снизу).
- Полые стрелки с поясняющим текстом (рис. 9, слева).
- Выделение важных неравенств серым цветом на белом фоне без использования контура (рис. 11).
- Контраст графики и надписи: графика на сером фоне, надпись в подвале — на белом (рис. 13).
- Графические заголовки в полых стрелках (рис. 16).
- Использование крупных черных стрелок для облегчения выявления очной ставки (рис. 21, 22).
- Групповая надпись в облаке, из которого исходит несколько стрелок, указывающих на несколько объектов (рис. 23*а*).

Внимательно анализируя полный набор иллюстраций к этой книге, любознательный читатель сумеет “выудить” из них большое количество полезных эргономических приемов и таким путем составит более глубокое представление о методе эргономической визуализации “Когнитивный стиль” *(CogniStyle)*.

ПРИМЕР МАТЕМАТИЧЕСКОЙ ВИЗУАЛИЗАЦИИ С ПОМОЩЬЮ МЕТОДА *COGNISTYLE*

Чтобы убедиться в универсальности метода *CogniStyle*, рассмотрим в качестве примера математическую задачу, а именно: возведение матрицы в степень (рис. 140). Особенность задачи в том, что требуется возвести в степень матрицу частного вида, которая называется сопро-

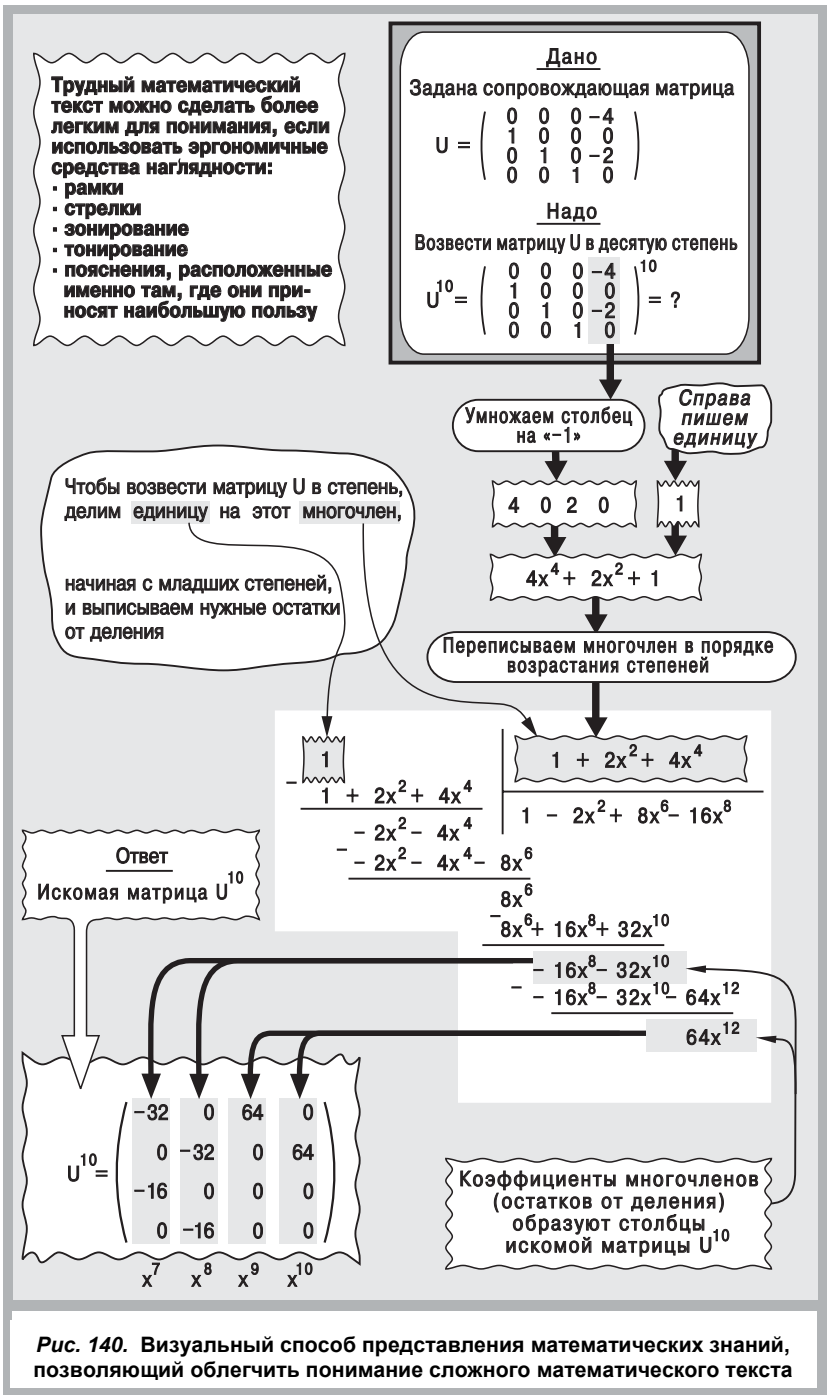


Рис. 140. **Визуальный способ представления математических знаний, позволяющий облегчить понимание сложного математического текста**

вождающей матрицей многочлена[1] [14]. Можно показать, что строки искомой матрицы — это векторы, являющиеся коэффициентами многочленов, принадлежащих определенной последовательности многочленов, а сами многочлены (члены последовательности) суть остатки от деления единицы на некоторый многочлен (получаемый путем преобразования нижней строки заданной матрицы), причем деление единицы на многочлен производится, начиная с младших степеней (как при делении рядов) [15]. Впрочем, в данный момент нас интересует не математическая суть вопроса, а конкретный набор эргономических средств и приемов, используемых для описания математической задачи, чтобы сделать ее предельно ясной, доходчивой и легкой для понимания.

На рис. 140 использованы очень простые, но с эргономической точки зрения весьма выразительные визуальные средства:

- Рамки различной формы, акцентирующие внимание на нужных математических объектах.
- Стрелки, устанавливающие связи между понятиями, заключенными в рамки.
- Стрелки, связывающие пояснения (расположенные в округлых рамках типа “облако”) с нужными понятиями.
- Зонирование текста, т. е. выделение тех или иных фрагментов текста с помощью замкнутых контуров различной формы. Форма контура имеет некоторый алфавит (визуальный синтаксис) и соответствующую визуальную семантику. Зонирование выполняет три функции:
  1) четко отделяет математический текст (математические объекты и связанные с ними математические преобразования) от метатекста (словесных пояснений к математическому тексту);
  2) организует “статическую” структуру диосцены;
  3) вместе со стрелками организует “динамическую” структуру диосцены, позволяющую проследить ход математических преобразований.
- Тонирование текста с помощью оттенков серого и белого цветов помогает зрительно дифференцировать разные визуально-математические понятия и облегчить зрительное восприятие их иерархической структуры.

Перечисленные и другие средства позволяют улучшить восприятие математического текста, сократить интеллектуальные усилия, которые человек затрачивает на процесс чтения, понимания и усвоения знаний.

Каждый из названных пунктов представляет некоторый общий эргономический принцип, который может быть детализирован и превращен в большое число конкретных формально-эргономических правил. Впрочем, с математической точки зрения они не являются формальными. Чтобы достичь терминологического компромисса, их можно назвать “полуформальными”.

Таким образом, *CogniStyle* — это “полуформальный” метод.

---

[1] По соображениям удобства на рис. 140 мы изображаем сопровождающую матрицу в транспонированном виде.

ВЫВОДЫ

1. Одной из главных проблем математики является ее неимоверная трудность, которая во многих случаях превышает возможности человеческого ума, что является основной причиной математического терроризма.
2. Сегодня производительность математического труда крайне низка, однако повышение этой производительности есть проблема не столько математики, сколько эргономики.
3. Математическая формализация является — по крайней мере частично — эргономическим процессом, а раз так, ее можно и нужно рассматривать с когнитивно-эргономических позиций.
4. Применение когнитивно-эргономических методов для улучшения письменной (визуальной) формы представления математических знаний позволяет существенно улучшить когнитивное качество и понимаемость математических текстов.
5. Традиционные формы и методы развития и фиксации математических идей, игнорирующие эргономические аспекты математических проблем и тем самым создающие питательную среду для математического терроризма, следует признать устаревшими.
6. Для успешного продвижения вперед необходимо осуществить синтез идей математики и эргономики в рамках нового междисциплинарного направления — эргоматематики.
7. В настоящее время имеются все необходимые предпосылки для практического создания нового поколения учебников математики, построенных с учетом когнитивно-эргономических принципов.
8. Цель создания нового поколения математических учебников и книг состоит в том, чтобы сократить интеллектуальные трудозатраты на восприятие, понимание и глубокое усвоение математических знаний в несколько раз, возможно на порядок.

Глава 20

МОЖНО ЛИ СТАТЬ ИНТЕЛЛЕКТУАЛЬНЫМ СУПЕРМЕНОМ?

> Нужно, как то свойственно сильным, отдавать предпочтение вопросам, которые никто не осмеливается ставить; необходимо мужество, чтобы вступить в область запретного... и новые уши для новой музыки... новая совесть, чтобы расслышать истины, прежде немотствовавшие.
>
> *Фридрих Ницше*

НА ПОРОГЕ СОЗДАНИЯ ТЕОРИИ УЛУЧШЕНИЯ РАБОТЫ УМА

Наше извилистое путешествие сквозь множество пестрых и порою нелегких глав может вызвать у читателя недоумение: в чем главная идея книги? Ответ мы получим в этой главе, которая представляет собой нервный центр работы, ее интеллектуальное ядро. Подобно развязке детективного романа, она призвана расставить все по местам и превратить запутанную мозаику фактов в целостную и четкую картину.

Диоинформация — это любая информация, представленная в форме одной или нескольких диосцен. Обобщая формулу, приведенную в гл. 19, можно записать:

Диоинформация	=	Словесный текст	+	Формулы	+	Изображения

В основе книги лежит гипотеза о максимизации мозговой продуктивности. Из нее вытекает принцип эргономизации диоинформации, который гласит: чтобы улучшить работу ума, необходимо улучшить когнитивное качество диоинформации, эргономизировать ее. При проектировании диосцен необходимо, в частности, заботиться о том, чтобы представить заданное смысловое содержание с помощью оптимального сочетания эргономичного текста, эргономичных формул и эргономичных изображений (статичных и динамических).

Стремясь найти свидетельства в поддержку названного принципа, мы представили на суд читателя несколько примеров эргономизации диоинформации:

- язык ДРАКОН;
- примеры эргономизации логики;
- примеры эргономизации математики;
- примеры эргономизации рисунков (метод “Когнитивный стиль” *CogniStyle*).

Этот перечень можно продолжить. При участии автора разработана технология эргономизации учебников и учебных пособий, предназначенная для создания эргономичных (электронных и бумажных) альбомов графических схем (так называемых *схемокурсов*) по гуманитарным дисциплинам: философии, экономике, психологии, культурологии, социологии, истории,

юридическим наукам и т. д. В ближайшее время первые учебные пособия, разработанные по этой технологии, выйдут в свет и, как предполагается, начнется их экспериментальная отработка в реальном учебном процессе.

Опираясь на эти и другие примеры, мы предположили, что:

ЭРГОНОМИЗАЦИЯ ДИОЯЗЫКОВ И ДИОИНФОРМАЦИИ ЕСТЬ УНИВЕРСАЛЬНЫЙ МЕТОД УЛУЧШЕНИЯ РАБОТЫ УМА, ПРИМЕНИМЫЙ К ЛЮБОЙ ОТРАСЛИ ЗНАНИЯ, ЛЮБОЙ НАУЧНОЙ И УЧЕБНОЙ ДИСЦИПЛИНЕ, ЛЮБЫМ ВИДАМ ИНТЕЛЛЕКТУАЛЬНОЙ ДЕЯТЕЛЬНОСТИ — КАК ПРОФЕССИОНАЛЬНОЙ, ТАК И УЧЕБНОЙ.

До сих пор мы доказывали это утверждение на примерах, т. е. индуктивным методом. А нельзя ли дополнить индуктивное доказательство дедуктивным? Нельзя ли построить общую теорию интенсификации интеллекта, из которой принцип эргономизации и принцип максимизации мозговой продуктивности вытекают как логическое следствие?

ЧЕЛОВЕЧЕСКИЙ МОЗГ НУЖНО ГРАМОТНО ПРОЕКТИРОВАТЬ

Попытаемся сделать набросок теории интенсификации интеллекта, в основу которой положим два базовых понятия — *знаки* и *предметы*. В рамках предлагаемой модели человек рассматривается как биокибернетический робот-автомат, взаимодействующий со знаковой (информационной) и предметной (физической) средой обитания (рис. 141).

Тезис 1. (Модифицированный тезис Л. Выготского). *Знаки* — созданные человеком чувственно воспринимаемые социальные инструменты, единственное предназначение которых — управлять человеческим мозгом, а через него — человеческим телом, поведением и всей человеческой жизнью (ср. [1]).

Тезис 2. Знаковое управление мозгом включает в себя две операции: проектирование мозга и программирование мозга. В первом случае изменяется нейростатическая конструкция мозга (конфигурация и характер межнейронных связей), во втором — меняется нейродинамическая внутримозговая информация.

Тезис 3. (Модифицированный тезис А. Полторацкого и В. Швырева). *Предметы* определяются как не-знаки [2]. Проще говоря, все, что не является знаком, по определению является предметом.

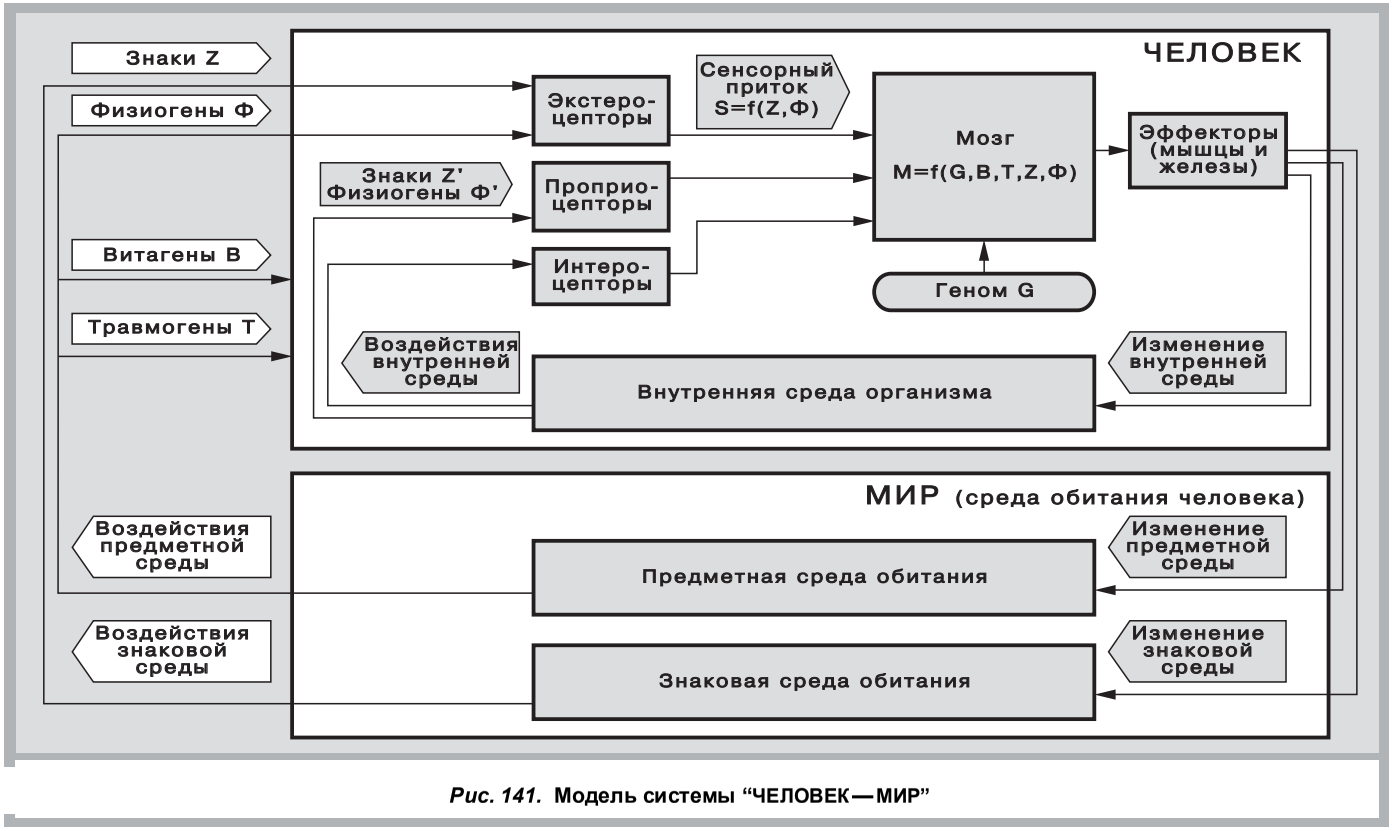


Рис. 141. **Модель системы "ЧЕЛОВЕК — МИР"**

Примерами знаков служат любые диосцены, книги, техническая и управленческая документация, информация на дисплеях, тексты программ, своды законов, показания приборов, произведения искусства, речь, мимика, деньги, музыка, украшения, игрушки, храмы. К предметам относятся девственная природа, дороги, станки, трубопроводы, микросхемы, а также все, что недоступно органам чувств: электромагнитные поля, атомы, электроны и т. д. В некоторых случаях знаки нельзя физически отделить от предметов. Например, телевизор — *предмет*, а изображение на экране — *знак*. Если телевизор выключить, знак исчезнет, а предмет останется.

Тезис 4. В мире нет ничего кроме предметов и знаков.

Тезис 5. Человеческая деятельность делится на предметную и знаковую. Используя предметные орудия (например, пилу, топор и рубанок), человек изменяет мир. Используя знаковые орудия (например, речь, молитву или язык программирования), человек изменяет собственный мозг и, следовательно, поведение.

Тезис 6. *Человек* — это животное, которое потенциально (благодаря особенностям генома) имеет возможность эффективно проектировать и программировать собственный мозг (с помощью специально созданных для этой цели социальных инструментов — знаков) и за счет этого многократно усиливать свои духовные, культурные, интеллектуальные и иные потенции и свойства. В настоящее время указанные возможности используются далеко не полностью из-за ошибочных представлений о сущности человека.

Основанием для разграничения знаковой и предметной среды обитания является тот факт, что первая из них играет неизмеримо более важную роль для формирования и развития интеллекта и духовной культуры человечества. Эту мысль выражает

Тезис 7. (Тезис Выготского о знаковой детерминации сознания и мышления, изложенный в форме метафоры). Интеллект, сознание и духовная культура людей на 99% определяются знаковой средой обитания и всего лишь на 1% — предметной средой обитания.

Тезис 8. *Сенсорный приток S* — поток информации о состоянии внешнего мира и человеческого тела, поступающий через рецепторные клетки органов чувств по нервным волокнам в мозг человека [3].

Тезис 9. *Мозг M* — биокомпьютер, обрабатывающий сенсорный приток *S* и управляющий работой эффекторов (мышц и желез), а через них — человеческим телом, поведением и всей человеческой жизнью.

Как отмечает Д. Сахаров, “в основании нейробиологии лежит простая, разумная и вместе с тем болезненно непривычная мысль, что нервная система — это [информационное] управляющее устройство, выполненное биологическими средствами” [4].

Тезис 10. Мозг *M*, как и весь организм, образуется из одной единственной первоклетки (*зиготы*) в процессе онтогенетического (индивидуального) развития под влиянием генома *G* и среды *C*:

$$M = f(G,C)\,. \tag{1}$$

Формула (1) представляет собой математическую модель, определяющую развитие целостного мозга, содержащего десятки миллиардов нейронов и триллионы межнейронных связей, из одной клетки — зиготы. Модель задана как функция двух переменных, аргументами которой являются генетический код (геном) и воздействия, поступающие из среды C (рис. 141).

Тезис 11. Среда C обитания человека структурно делится на четыре части: витагены B, травмогены T, знаки Z и физиогены Ф:

$$C = \{B, T, Z, \Phi\}, \tag{2}$$

причем перечисленные части образуют полный набор воздействий среды на человека.

Тезис 12. *Витагены* B — жизнеподдерживающий поток веществ, вводимых в организм в ходе обмена веществ, а также пригодные для жизни условия предметной среды (температура, давление и т. д.).

Тезис 13. *Травмогены* T — негативные воздействия предметной среды на организм человека: травмы, ранения, ожоги, инфекции.

Тезис 14. *Знаки* Z и *физиогены* Ф — элементы знаковой и предметной среды обитания соответственно, воздействующие на рецепторы органов чувств, которые преобразуют их в сенсорный приток S:

$$S = f(Z,\Phi)\,. \tag{3}$$

Заметим, что витагены и травмогены влияют не на органы чувств, а на организм человека, причем их действие может быть либо позитивным или нейтральным (витагены), либо негативным (травмогены). Витагены, травмогены и физиогены — это не знаки, а предметы.

Из формул (1) и (2) получаем

$$M = f(G, B, T, Z, \Phi)\,. \tag{4}$$

Тезис 15. Смысл формулы (4) таков: все процессы, связанные с развитием мозга в результате деления зиготы, а также любые процессы, протекающие внутри мозга и нервной системы при любых видах деятельности (включая наиболее сложные виды интеллектуальной работы), полностью определяются квинтетом “царских переменных”, который включает геном, витагены, травмогены, знаки и физиогены. Согласно развиваемой модели, никакие другие причины (переменные) на развитие и деятельность мозга и интеллекта влияния не оказывают.

С биологической точки зрения любое поведение человека (будь то прогулка, работа за компьютером или игра на скрипке) есть не что иное как последовательность отдельных движений человеческого тела, причем в каждом движении участвуют тысячи эффекторов. Что же вызывает срабатывание эффекторов? При активации двигательных нервов в нервномышечном синапсе происходит выброс химического медиатора ацетилхолина, который передает мышце команду сокращаться [5]. Известно, что произвольные движения тысяч эффекторов (которые в

своей совокупности образуют поведение П) выполняются по командам мозга *M* и нервной системы. Значит, поведение П есть функция процессов, происходящих в мозгу *M*:

$$\Pi = f(M)\,. \tag{5}$$

Тезис 16. Смысл формулы (5) состоит в том, что поведение однозначно задается командами нервной системы: ни один эффектор не изменит свое состояние, если нервная система не выдаст нужную команду.

Из (4) и (5) получаем

$$\Pi = f(G, B, T, Z, \Phi)\,. \tag{6}$$

Тезис 17. Формула (6) говорит о том, что любое сколь угодно сложное поведение человека определяется квинтетом царских переменных.

Предположим, что жизнеподдерживающие условия — витагены — находятся в норме ($B \approx const$), а негативные воздействия — травмогены — отсутствуют ($T = 0$). Кроме того, учитывая (3), две простые переменные (Z и Ф) можно заменить одной сложной — сенсорным притоком *S*. В этом случае формула (6) упрощается и принимает вид

$$\Pi = f(G, S)\,. \tag{7}$$

Тезис 18. Формула (7) означает: если в среде обитания отсутствуют экстремальные условия, поведение человека зависит всего от двух переменных: генома и сенсорного притока.

Информацию в мозг поставляют два источника: геном и прижизненный чувственный опыт (далее — опыт). Нейробиологическим аналогом понятия “опыт” является интегральный сенсорный приток

$$S(T) = \int_{t=0}^{T} S(t)dt\,. \tag{8}$$

где $S(t)$ — мгновенное значение сенсорного притока;

$t = 0$ — момент зачатия и образования зиготы;

T — текущий возраст человека, измеряемый от момента зачатия.

Таким образом, интегральный сенсорный приток (прижизненный опыт) — это полная информация, поступающая в мозг в течение прожитого отрезка жизни от зачатия до текущего возраста *T*.

Понятие “опыт” охватывает информацию, поступающую в мозг от экстероцепторов[1] и (по петле обратной связи) проприоцепторов[2]. Общее число сенсорных клеток (датчиков информации), включая интеро-

---

[1] Экстероцептор — биологический датчик информации (рецептор), передающий в мозг информацию об окружающем мире.

[2] Проприоцептор — биологический датчик, расположенный в тканях мышечного и суставного аппарата, воспринимающий их растяжение и сокращение и передающий в мозг информацию о положении мышц и суставов. Благодаря этой информации человек может, например, с закрытыми глазами попасть пальцем в кончик носа.

цепторы[1], измеряется сотнями тысяч. Это сотни тысяч[2] "пушек", осуществляющих в течение всей жизни человека непрерывную "сенсорную бомбардировку" мозга и нервной системы, которая оказывает огромное влияние на поведение, жизнь и судьбу человека.

Громадный объем сенсорной информации в сочетании со сверхсложным механизмом экспрессии генетического кода и определяет кажущуюся "непостижимой" сложность человеческого поведения.

Тезис 19. (Модифицированный тезис В. Штерна и Ж. Пиаже). *Человеческий интеллект* I — свойство мозга M, позволяющее человеку приспособиться к знаковой и предметной среде обитания:

$$I = f(M)\,. \tag{9}$$

Из (4) и (9) получаем

$$I = f(G, B, T, Z, \Phi)\,. \tag{10}$$

Тезис 20. Интеллект I любого человека определяется квинтетом царских переменных этого человека.

Повторяя рассуждения, изложенные в тезисах 17 и 18, получаем

$$I = f(G, S)\,. \tag{11}$$

Тезис 21. При отсутствии экстремальных условий человеческий интеллект I зависит от двух переменных: генома G и интегрального сенсорного притока S.

Если речь идет об уже родившемся конкретном человеке, не имеющем генетических дефектов (это означает, что геном G задан и находится в пределах нормы), из (11) вытекает

Тезис 22. Интеллект I является функцией сенсорного притока S:

$$I = f(S)\,. \tag{12}$$

Тезис 23. Чтобы улучшить качество человеческого интеллекта, необходимо улучшить качество сенсорного притока S.

Тезис 24. Из (3) и (12) следует, что интеллект I есть функция знакового и предметного (физиогенного) притоков:

$$I = f(Z, \Phi)\,. \tag{13}$$

Предположим, что физиогенный приток, поступающий в мозг от экстероцепторов и проприоцепторов, находится в пределах нормы. Имеется в виду, что онтогенетическое развитие человека в предметной среде обитания организовано разумно с учетом сенситивных (критических) периодов развития. При этих условиях справедлив

---

[1] Интероцептор — биологический датчик, передающий в нервную систему информацию о состоянии внутренних органов (желудок, легкие, сердце и т. д.). Эта информация необходима для работы систем автоматического (бессознательного) управления пищеварением, дыханием, кровообращением и т. д.

[2] См.: Р. Флиндт. Биология в цифрах. М.: Мир, 1992. С. 248, 277.

Тезис 25. Человеческий интеллект I есть функция интегрального притока знаковой информации, воздействующего на мозг в течение всей прожитой жизни от зачатия до текущего момента:

$$I = f(Z). \tag{14}$$

Тезис 26. Для любого новорожденного человека единственный путь к улучшению его интеллекта состоит в обогащении и улучшении качества знаков, потребляемых им на протяжении жизни. Особенно эффективным является синхронное поступление обогащенного знакового притока, передаваемого в мозг от экстероцепторов и проприоцепторов (например, при самостоятельном решении усложняющихся задач).

Тезис 27. *Интеллектуальный приток* Q — часть знакового притока Z, поступающая в мозг через зрительные, слуховые и кожные рецепторы и влияющая на умственную продуктивность человека.

Тезис 28. Поскольку тактильная (кожная) речь не используется при обучении зрячеслышащих, у последних интеллектуальный приток поступает в мозг преимущественно через зрительные и слуховые рецепторы.

Предположим, что интеллектуальный приток, направляемый в мозг через глаза и уши, распределяется между ними пропорционально числу зрительных и слуховых рецепторов, число которых у человека равно 253 млн. и 47 000 соответственно. Тогда справедлив

Тезис 29. Интеллект зрячих людей на 99,99% определяется зрительной знаковой информацией, в первую очередь — диоинформацией. Чтобы улучшить интеллект, нужно улучшить качество последней.

Тезис 30. (*Принцип когнитивного диоинтерфейса*). Чтобы система “человек—диоинформация” была эффективной и обеспечивала максимально возможную продуктивность мозга и интеллекта, нужно осуществить взаимную адаптацию человека и диоинформации. В частности, нужно проектировать диоязыки таким образом, чтобы согласовывать между собой когнитивно значимые оптические характеристики диоинформации и когнитивные характеристики человеческого глаза и мозга.

Согласование характеристик следует проводить таким образом, чтобы, во-первых, добиться максимально быстрого, точного, полного и безошибочного восприятия, понимания и усвоения знаний, во-вторых, решить эту задачу ценою минимальных интеллектуальных усилий со стороны человека, потребляющего диоинформацию.

Тезисы 31—33 были изложены ранее в гл. 5. Имеются в виду:

- принцип симультанизации;
- принцип зависимости эффективности восприятия от используемой доли поля зрения;
- принцип приоритета целостного образа.

РАЗГАДКА ТАЙНЫ ЧЕЛОВЕЧЕСКОГО ИНТЕЛЛЕКТА

Чтобы разгадать “главную тайну” человеческого интеллекта и мозга, нужно ответить на вопрос: в чем заключается принцип, определяющий

связь между знаковой средой обитания человека и основополагающими элементами конструкции мозга?

Тезис 34. Основополагающими конструктивными элементами мозга следует считать *гибкие межнейронные связи* [6], поскольку именно они играют ключевую роль при запоминании новой информации и приспособлении к изменяющейся среде обитания.

Тезис 35. Фундаментальная роль знаков состоит в том, что они выполняют роль "хирургических инструментов", изменяющих основополагающие характеристики мозга и, следовательно, служат первоисточником всех тех преобразований (и позитивных, и негативных), которые осуществляет человек, изменяя лицо планеты.

Тезис 36. Совершенствуя знаки (создавая новые научные теории, идеи и проекты, новые произведения искусства и более совершенные религиозные учения), человек совершенствует свой мозг и тем самым обеспечивает поступательное развитие цивилизации.

Тезис 37. Научившись конструировать, изменять, дорабатывать и улучшать знаки, человек тем самым научился конструировать, изменять, дорабатывать и улучшать свой собственный мозг, непрерывно увеличивая его интеллектуальное могущество. Ясно, что подобного рода "операции на мозге" не требуют ни скальпеля, ни вскрытия черепа, ибо осуществляются информационным путем — путем воздействия знаков на рецепторы органов чувств, которые передают в мозг нужную информацию.

Отсюда вывод: мозг — это глина, а знаки — пальцы скульптора. Чтобы у ребенка "вырос" наилучший мозг (какой только возможен при заданных генах), необходимо с момента рождения "кормить" его знаками наивысшего качества, учитывая критические (сенситивные) периоды онтогенетического развития.

Тезис 38. Механизм, посредством которого люди улучшают конструкцию мозга, таков. Сначала некоторый автор (производитель знаков) нарочно или случайно создает новые знаки. Затем потребитель знаков (в роли которого выступает не только другой человек, но и сам автор) воспринимает новые знаки с помощью своих органов чувств. При этом поток знаковой информации поступает в мозг, в котором при выполнении определенных условий происходят конструктивные изменения межнейронных связей. В этих дополнительных (вновь образованных) связях закодирована программа решения новых задач, которые раньше были непосильными для мозга.

Таким образом, умело проектируя знаки, мы умело проектируем мозг, придавая ему желаемые характеристики. И наоборот, неграмотное проектирование знаков дает в результате неграмотно спроектированный мозг.

Тезис 39. Творческая деятельность человека по созданию новой среды обитания делится на три этапа. Сначала люди придумывают новые знаки (например, теорию космических ракет), затем с их помощью изменяют свой мозг (появляются группы людей, освоивших новую теорию) и только после этого создают новые предметы (ракеты-носители и космические корабли).

Те же три этапа присутствуют и в более простых творческих задачах. Предположим, нужно создать новую машину, технологию или компьютерную программу. В исходный момент решение задачи неизвестно, стало быть, соответствующие связи в мозгу отсутствуют. Поэтому сначала (для того, чтобы "поумнеть") человек непременно должен преобразовать свой мозг. Как это сделать? Прежде всего нужно найти, заимствовать, освоить или придумать новые знаковые комплексы. С их помощью человек посредством многочисленных итераций переделывает (улучшает) конструкцию своего мозга, делая его более "умным" и, следовательно, пригодным для выполнения новой сложной работы. И лишь на третьем этапе, опираясь на возросшую интеллектуальную мощь своего "улучшенного" мозга, человек решает проблему, т. е. создает новую машину, технологию или программу.

Подведем итоги. Главная мысль состоит в том, что интеллект есть функция знаков. Из этого вытекает, что ***мощь человеческого интеллекта зависит от качества знаковых комплексов, узоров и комбинаций, которые человеческий мозг получает от рецепторов органов чувств в течение жизни. Чем лучше характеристики знаков, тем оптимальнее порождаемые ими гибкие межнейронные связи мозга, тем выше скорость решения (мозгом) интеллектуальных задач.*** Значит, чтобы повысить качество человеческого интеллекта, необходимо обеспечить более высокое качество знаков, в частности когнитивное качество диоязыков и диоинформации.

Особенно важную роль для увеличения производительности мозга играет диоинформация. К сожалению, нынешние методы ее создания крайне неэффективны. Поэтому в подавляющем большинстве случаев она не удовлетворяет принципу когнитивного диоинтерфейса и тормозит интеллектуальную производительность людей.

Чтобы поправить дело, во всех когнитивно сложных случаях (а таких очень много или даже большинство) необходимо полностью или частично реализовать программу из трех пунктов:

1) перейти от текстовых диосцен к изобразительным (трехэлементным — см. с. 326);
2) увеличить формат диосцен в соответствии с принципом симультанного охвата поля зрения;
3) при построении диосцен использовать строго определенный набор диосинтаксических правил, вытекающих из принципа когнитивного диоинтерфейса.

На наш взгляд, при производстве любой когнитивно сложной диоинформации принцип когнитивного интерфейса целесообразно рассматривать в качестве *критерия научности*. Это означает, что научным должно быть не только содержание знаний; форма их представления также должна быть научно обоснованной.

РАЗВИТИЕ И ИНТЕНСИФИКАЦИЯ ИНТЕЛЛЕКТА

Развитие интеллектуальных способностей и интенсификация интеллекта — существенно разные понятия. Различие между ними поясним с помощью мысленного эксперимента.

Предположим, два студента, имеющие в точности одинаковые интеллектуальные способности, решили сдать экзамен экстерном, изучив предмет самостоятельно, по учебнику. Первому студенту достался плохой учебник, путаный и трудный. Зато второй, наоборот, получил хорошую книгу, написанную ясным языком, где сложнейшие вопросы излагаются отчетливо, в наглядной и доходчивой форме. В результате первому студенту, очевидно, придется затратить на изучение предмета больше усилий и времени, скажем, сто часов, а другому — намного меньше, например пятьдесят часов. Если согласиться с этими цифрами (а они вполне правдоподобны), мы вправе сказать, что умственная производительность второго студента оказалась в два раза выше, хотя их интеллектуальные способности одинаковы.

Двукратное увеличение умственной продуктивности у второго студента по сравнению с первым произошло благодаря ясности и доходчивости учебного материала, т. е. за счет улучшения когнитивного качества учебника. Отсюда проистекает

Тезис 40. Интеллектуальная производительность человека R в процессе диопознания (т. е. при изучении и работе с научно-технической и учебной литературой, документацией и компьютерной диоинформацией) зависит от двух переменных: от его интеллектуальных способностей I и от когнитивного качества изучаемого материала L:

$$R = f(I, L). \qquad (15)$$

Тезис 41. Из (15) следует, что возможны два взаимодополняющих подхода к решению задачи увеличения интеллектуальной производительности человеческого мозга: улучшение интеллекта I и улучшение когнитивного качества диоинформации L.

ЗНАКОВАЯ И ПРЕДМЕТНАЯ ИНФОРМАЦИЯ

Весь объем информации, имеющейся на нашей планете, можно разбить на два больших класса:

- доступную для восприятия с помощью человеческих органов чувств (назовем ее *знаковой*);
- чувственно не воспринимаемую, т. е. недоступную для человека (назовем ее *предметной*).

К счастью, человек в большинстве случаев умеет преобразовать предметную информацию в знаковую и благодаря этому узнать все, что ему нужно. Например, информация, спрятанная в микросхемах процессора или на винчестере персонального компьютера, является предметной (невидимой), однако ее можно легко преобразовать в знаковую форму. Для этого достаточно вызвать ее на экран дисплея, распечатать на принтере, а в более сложных случаях — подключить осциллограф. Информация в памяти космического зонда, который бороздит звездные просторы где-то возле Сатурна, сама по себе также является предметной (недосягаемой для наших органов чувств). Однако, будучи переданной на Землю, она превращается в знаковую форму — либо в фотографию Сатурна, либо в диагностическое сообщение на экране: дескать, так и так, блок питания зонда дышит на ладан и скоро совсем загнется.

Следует уяснить, что знаковая и предметная формы информации — принципиально разные вещи. Бытующее среди специалистов по информатике стремление не различать и даже отождествлять эти формы (мол, это одна и та же информация!) является грубой методологической ошибкой. Эта ошибка "ослепляет" сознание и мешает сделать существенный для нас вывод, что продуктивность мозга сильно зависит от качества именно знаковой информации и совершенно не зависит от любых свойств предметной информации.

ЗНАКОВОЕ И ПРЕДМЕТНОЕ ОБЕСПЕЧЕНИЕ ИНФОРМАТИКИ

Согласно классической точке зрения всю совокупность средств информатики принято разделять на аппаратуру *(hardware)* и программное обеспечение *(software)*. Эта типология принесла немало пользы, однако сегодня не менее важно научиться различать предметное *(objectware)* и знаковое *(signware)* обеспечение информатики (рис. 142).

Предметное обеспечение охватывает все средства информационной технологии, которые относятся к классу предметов:

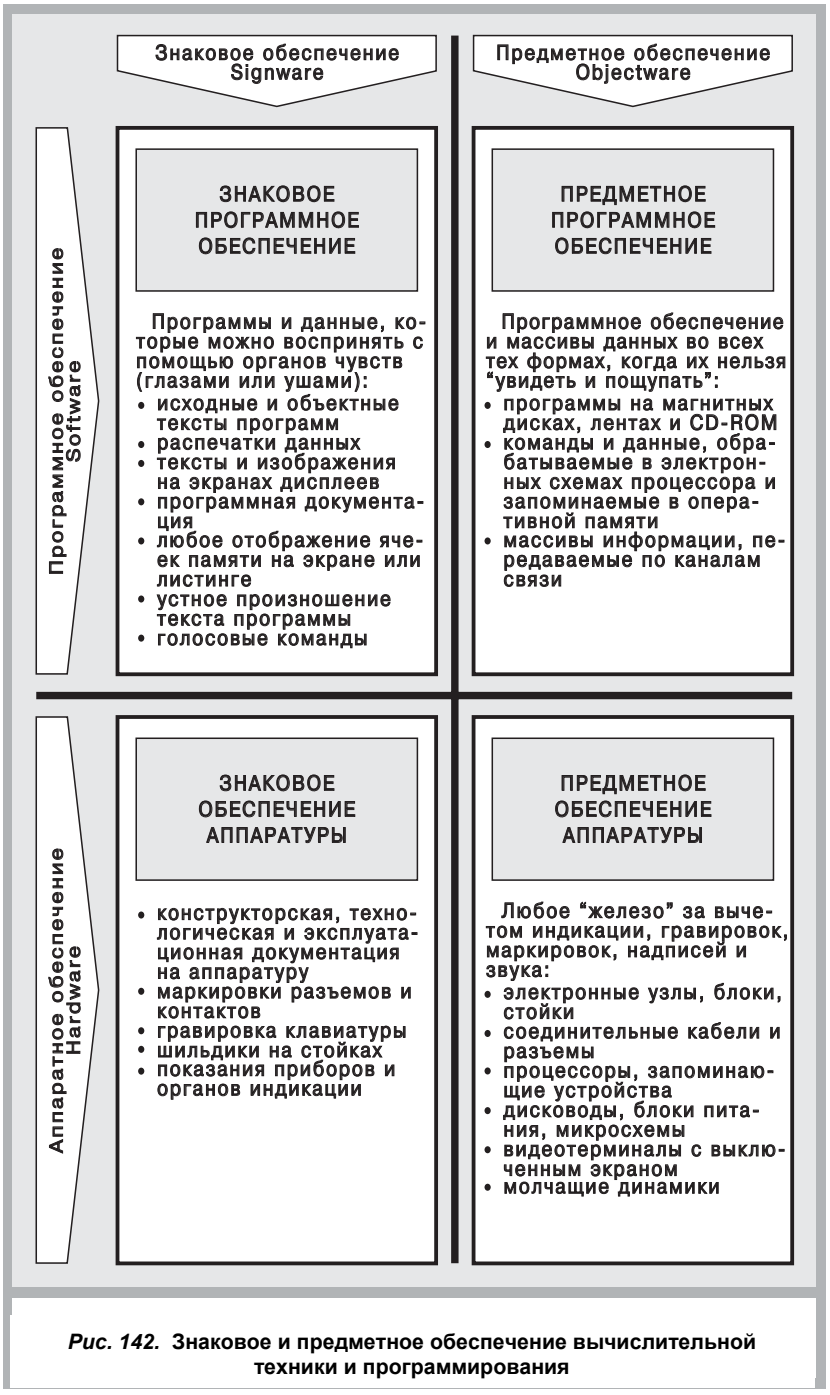


Рис. 142. **Знаковое и предметное обеспечение вычислительной техники и программирования**

- процессоры, память, дисководы, блоки питания, дисплеи с выключенным экраном и т. д.;
- программное обеспечение во всех формах, когда его нельзя “увидеть и пощупать”.

К знаковому обеспечению относятся:

- тексты и изображения на экране дисплея, а также бумажная продукция принтеров и графопостроителей;
- научно-техническая и учебная литература и документация;
- маркировки и другие обозначения на приборах и кабелях.

Для некоторых специалистов использование предметно-знаковой терминологии на первых порах оказывается довольно болезненным. Нужно научиться мысленно отделять текст или изображение на экране дисплея (знак) от самого дисплея, который является предметом. Надо осознать, что данные в оперативной памяти — это предметы, а те же самые данные на распечатке — это уже знаки.

ЗНАКОВАЯ И ПРЕДМЕТНАЯ ПРОГРАММА

Изложенные соображения позволяют сделать вывод, что основное понятие теории программирования — программа — является в высшей степени неудачным, неточным и дезориентирующим. Недостаток в том, что термин “программа” маскирует тот факт, что речь идет о двух принципиально разных объектах, между которыми нет почти ничего общего. Для обозначения этих объектов мы предлагаем термины “знаковая программа” и “предметная программа”.

Первая обычно имеет свое физическое бытие либо в виде написанного от руки черновика, либо текста на экране терминала, либо распечатки. Вторая может существовать в виде кодов в оперативной памяти, на магнитном либо лазерном диске, в виде массива данных, передаваемого по проводам и поступающего в схемы процессора.

Знаковая программа предназначена для управления деятельностью человека (но не машины), предметная — для управления компьютером (но не человеком). Если говорить более точно, знаковая программа служит для управления действиями автора программы или его коллег, которые пытаются изучить, понять, проверить, локализовать ошибку, вспомнить или модифицировать программу, а также для управления работой пользователя, который вводит программу в машину. Написанные на черновике символы программы, воздействуя на нейронные коды мозга, однозначно определяют последовательность движений (“кинетическую мелодию”) рук пользователя. В свою очередь движения пальцев преобразуются в механические перемещения клавиш и далее в электрические и магнитные сигналы.

Последние и представляют собой предметную программу, которая осуществляет непосредственное управление работой компьютера.

В связи с этим необходимо различать:

- качество знаковой программы (знаковое качество),
- качество предметной программы (предметное качество).

Качество предметной программы характеризуется, в частности, временем ее исполнения в машине и потребным объемом памяти. Критерием качества знаковой программы служат затраты энергии мозга (интеллектуальные усилия), необходимые для восприятия и понимания программы.

Чем лучше знаковая программа, чем она понятнее и доходчивее, тем меньше вероятность того, что в ней затаились скрытые ошибки, тем выше надежность предметной программы. Качество и интеллектуальная производительность труда программистов совершенно не зависят от предметного качества программы и всецело определяются знаковым качеством. Отсюда вывод: чтобы повысить продуктивность мозга разработчиков программ, необходимо повысить знаковое качество программного обеспечения.

ПЕРЕЛОМНАЯ ВЕХА В ИСТОРИИ ИНФОРМАТИКИ

В гл. 3, анализируя проблему "интеллектуальной конкуренции" человека и компьютера, мы убедились, что сегодня узким местом являются не машинные, а человеческие ресурсы, т. е. недостаточная производительность персонала. В этих условиях все больше авторов приходит к выводу, что из всего многообразия причин, влияющих на развитие информатики, человеческий фактор является решающим, приоритетным [7]. Этот вывод, который можно охарактеризовать как *принцип приоритета человека*, означает признание того факта, что человек (заказчик, разработчик, программист, пользователь и т. д.) является центральным звеном любой информационной технологии. А раз так, то главным элементом теоретического фундамента информатики должна стать теория человека, человеческого интеллекта и человеческого мозга.

С учетом сказанного нельзя не согласиться с Н. Бьерн-Андерсеном, который считает, что традиционный подход к исследованию человеческого фактора в компьютерных системах является бесперспективным "до тех пор, пока приоритет не будет отдан человеческим и социальным ценностям, не удастся решить все те проблемы, которые встают сейчас в связи с разработкой и эксплуатацией информационных технологий" [8]. В связи с этим сделаем ряд замечаний. Сам факт признания принципа приоритета человека мы склонны трактовать как переломную веху в истории информатики. До этого момента информатика *(computer science, information science)* рассматривалась как естественная и техническая наука, которая не имеет никакого или почти никакого отношения к человеку и его проблемам.

Принцип приоритета человека, согласно которому теория интеллекта и мозга должна стать главным элементом теоретических основ информатики, означает коренной пересмотр прежних взглядов, означающий, что информатика превращается в междисциплинарную науку, охватывающую два направления: *техническую* информатику и *гуманитарную* информатику. В связи с этим возникает острая необходимость в создании нового понятийного аппарата, общего для двух ветвей информатики, который естественным образом объединяет их в единое целое. Предметно-знаковая модель представляет собой искомый понятийный аппарат.

Нужда в таком аппарате велика. Глобальный процесс информатизации, качественно преобразующий социосферу земного шара, беспрецедентные масштабы интеллектуальной революции, вовлекающей в свою орбиту миллионы людей, делают проблему все более актуальной. Парадокс в том, что сегодня люди с нарастающей активностью "эксплуатируют" собственный мозг, толком не представляя механизмов его работы. В результате многие возможности мозга остаются неиспользованными.

Чтобы улучшить экономические и иные показатели компьютерной революции, повысить ее интеллектуальную эффективность, надо, чтобы широкие массы специалистов были вооружены ясной руководящей идеей об устройстве собственного мозга. Жизнь властно требует простого, пусть не совсем точного, но непременно эффективного объяснения "тайны" человеческого интеллекта и мозга. Что толку без конца повторять, что мозг сложен, а психика еще сложней. Наука продвигается вперед, когда сложным явлениям удается найти простое и полезное объяснение.

Выше в форме тезисов мы предложили чрезвычайно простую (предметно-знаковую) концептуальную модель мозга и интеллекта, которая имеет ряд достоинств:

- модель не противоречит никаким экспериментально установленным и признанным наукой фактам;
- для некоторых из этих фактов, в частности, касающихся информатизации и интеллектуализации, она дает стандартный подход к их объяснению и пониманию;
- она обладает прогностическими возможностями, позволяя выявить недостатки нынешнего процесса информатизации и указать пути их устранения;
- модель указывает обоснованный путь к интенсификации интеллекта. Надо сосредоточить усилия на принципиально новой научной проблеме — разработке методов улучшения когнитивного качества знаковой информации, в первую очередь диоинформации;
- модель является открытой для усложнения и наращивания.

ОДНОГЛАЗЫЕ МИССИОНЕРЫ, ИЛИ ЗАБРОШЕННОЕ ДИТЯ ИНФОРМАТИКИ

Специалистам по информатике принадлежит особая, выдающаяся роль в современном обществе. Они являются носителями идей информатизации, миссионерами новой компьютерной религии, которая на наших глазах изменяет лицо планеты, превращая ее в единое информационное пространство. Киберпространство *(cyberspace)* становится частью повседневной жизни миллионов людей.

Происходящий сегодня "сверхсинтез" сообщества людей и сообщества машин, приводящий к объединению людей и компьютеров с помощью локальных и глобальных систем связи, создает качественно новый феномен человеческой цивилизации, который называют поразному: "глобальный гиперинтеллект" *(А. Ракитов)*, "сверхинтеллект, охватывающий всю нашу планету", "интегрированная система естественного интеллекта" *(А. Урсул)*, "суперинтеллект в мировом масштабе" *(Дж. Вакка)*, "разум человечества", "центральная нервная система человечества" *(В. Лищук)*, "коллективный интеллект" *(Н. Моисеев)*.

Все это, разумеется, хорошо. Плохо то, что эти восторженные оценки прячут, маскируют чрезвычайно важную проблему, которая, к сожалению, остается заброшенной нищенкой на роскошном празднике глобальной информатизации. Как уже догадался читатель, речь идет о проблеме интенсификации интеллекта. Подобный перекос в общественном сознании становится опасным и далее нетерпимым. Пришло время поместить проблему повышения продуктивности мозга на подобающий ей царский трон и привлечь к ней пристальное внимание специалистов.

Всю совокупность приемов, используемых для повышения интегральной производительности систем “персонал—компьютеры”, разделим на две части:

- сильно влияющие на творческую продуктивность человеческого мозга (назовем их *когнитивными*);
- никак не влияющие на продуктивность мозга или влияющие слабо (назовем их *служебными*).

Служебными приемами являются:

- автоматизация умственного труда (компьютеры берут на себя решение многих задач, освобождая мозг от необязательной работы);
- создание локальных и глобальных компьютерных сетей, которые облегчают доступ к информации, доставляя ее в нужное место, именно к тому человеку, который в ней нуждается;
- улучшение пользовательского интерфейса, включая графический интерфейс *GUI (Graphic User Interface)*.

Первые два метода не влияют на продуктивность мозга, а третий влияет незначительно, так как облегчает решение лишь ограниченного круга второстепенных задач, связанных с навигацией по информационному пространству, работой с окнами, меню и т. д.

Служебные приемы не имеют никакого отношения к вопросу, который для нас является центральным: какой должна быть форма представления сложных профессиональных и учебных знаний, обладающая наивысшим когнитивным качеством? Проблема проектирования наилучшей формы представления знаний — это не служебная, а когнитивная проблема. Она-то и есть заброшенное дитя информатики. Появление в последнее время серьезных работ по визуализации представления знаний [9] — шаг в правильном направлении, но он далеко не исчерпывает проблему. Принцип приоритета человека, признание важности проблемы максимизации мозговой продуктивности требует радикального пересмотра прежней позиции. Это значит: не ослабляя усилий по совершенствованию служебных методов и неуклонно наращивая их, вместе с тем центр тяжести исследований следует перенести со служебных методов на когнитивные.

КОГНИТИВНАЯ ПИСЬМЕННОСТЬ — НОВЫЙ СПОСОБ ПРЕДСТАВЛЕНИЯ ЗНАНИЙ

Изобретение письма явилось важнейшим событием в истории человечества. Тем не менее прежнее письмо уже устарело. Мы являемся свидетелями нового революционного изменения письменности, для которого характерны следующие особенности.

- Одномерный словесный текст заменяется двумерной диосценой, что позволяет перейти от медленного сукцессивного восприятия к быстрому симультанному.
- Одноэлементное письмо (содержащее только текст) заменяется оптимальным эргономичным сочетанием трех элементов (эргономичный текст + эргономичные формулы + эргономичные чертежи).

Ранее науки об оптимальном способе визуального представления знаний не существовало. Поэтому автор научной книги или учебника нередко выступал в роли “диктатора”, а читатель приспосабливался к нему, зачастую с трудом продираясь сквозь непролазные джунгли невразумительного словесного текста.

Долой когнитивно бездарные компьютеры и костыли старомодных “хелпов”!

Изложенная выше модель направлена на научное обеспечение когнитивной письменности. При этом во главу угла ставится необходимость обеспечить максимальную производительность умственного труда читателя, нацеленного на восприятие, понимание и усвоение знаний. Знаки и знаковые комплексы (диосцены) предназначены прежде всего для взаимодействия с сенсорным и нейронным аппаратом человека. Человек и знаки образуют систему, элементы которой должны быть согласованы между собой. Поэтому новая письменность опирается на принцип когнитивного диоинтерфейса.

При когнитивном подходе ручной способ письма становится практически невозможным. Это объясняется тем, что когнитивная диосцена должна обладать очень точными характеристиками, обеспечивающими максимальную скорость понимания. Поэтому когнитивная письменность должна иметь компьютерную точность. Однако современные информационные технологии не годятся для этой цели — они слишком примитивны. Нужны новые технологии, значительно более сложные и дорогие. Известно однако, что стоимость — убывающая функция масштабов производства. Рано или поздно общество будет вынуждено перейти к массовому решению проблемы интенсификации интеллекта. ***Переход к массовому тиражированию когнитивных информационных технологий позволит снизить их стоимость, что создаст предпосылки для повсеместного использования когнитивной письменности.*** Этот момент обозначит вход в райский сад общедоступной суперинтеллектуализации, где, как мы надеемся, почти каждый сможет почувствовать себя “интеллектуальным суперменом”.

“КАСТРИРОВАННЫЙ” ИНТЕЛЛЕКТ

Однажды меня попросили нарисовать схему сложного процесса, спроектированного коллективом разработчиков. Каждый из них прекрасно знал свой кусок работы, но никто не владел картиной в целом. Я переговорил со специалистами, выяснил все необходимое и подготовил черновик схемы. Поскольку под рукой не было подходящего плоттера, схема была выполнена вручную чертежником. В итоге получилась бумажная простыня внушительных размеров — около двух метров в длину и полметра в высоту. Схема вышла неплохая — наглядная, удобная, отражающая все, что нужно, и не содержащая ничего лишнего.

Далее события развивались так. Заказчик решил “загнать” схему в компьютер: вызвал грамотного пользователя и дал ему поручение. Тот живо изуродовал схему: разрезал ее на двадцать частей, пронумеровал листы, обозначил переходы с листа на лист, снабдил их указателями для поиска нужной линии, ввел в персоналку, отпечатал на принтере формата А4 и переплел листы в аккуратную книжечку.

Увидев результат, я обомлел: наглядной и понятной схемы больше не было — целостный образ процесса бесследно исчез! Вместо него я обнаружил чудовище, представляющее собой набор невразумительных обрубков, сделанных по принципу: умрешь — не поймешь! Чтобы понять исходную бумажную простыню, требовалось десять минут, чтобы разобраться в двадцати обрубках — не меньше часа. Это значит, что производительность умственного труда при изучении схемы упала в шесть раз! Тот, кто знаком с проблемой, знает, что подобные перлы когнитивной безграмотности отнюдь не единичны — они встречаются сплошь и рядом. Парадокс в том, что люди уменьшают производительность умственного труда неосознанно, бездумно, по инерции, даже не догадываясь, какое когнитивное “преступление” они совершают.

При решении сложных интеллектуальных проблем человеку необходим целостный и одновременно детальный образ проблемы. Такой образ должен быть достаточно большим — его нельзя получить ни на экране современного персонального компьютера, ни на бумажных листах малого формата. Вспомним сказанное в гл. 5: чтобы увеличить продуктивность мозга, нужно строить такие диосцены, которые позволяют использовать богатейшие ресурсы симультанного восприятия. Если “прохлопать ушами” эту возможность, то мощные резервы человеческого интеллекта остаются невостребованными, и персонал обрекается на “частичную слепоту”. К великому сожалению, именно такова обычная практика массовой компьютеризации. Образно говоря, современное информационное общество, в котором доминируют компьютеры с небольшим экраном и принтеры формата А4 — это общество “кастрированного” интеллекта.

Еще одна проблема. Общеизвестно, что обучение по книгам отнимает у человека значительную часть его индивидуальной жизни, так как учебные и научные тексты слишком трудны для понимания. Установлено, что объем знаний, которые нужно усвоить, превышает возможности учащихся, состояние их здоровья ухудшается [10]. Несмотря на это, современные учебники считаются научными. Так ли это? Ведь “научность без доступности теряет смысл... Непонятные сведения загромождают ум, ибо человек не может ими пользоваться. Непонимание учебного материала вызывает чувство бесполезности учебы” [11]. Еще в начале века Н. Рубакин пи-

сал: поскольку нет восприятия, постольку нет и содержания. Не понимая этого, авторы книг тратят впустую $^{9}/_{10}$ своих сил, труда и времени.

Если наша мысль верна, придется признать, что ***текстовые учебники, издаваемые на всех континентах в сотнях миллионов экземпляров, устарели (не по содержанию, а по форме представления знаний), ибо в значительной мере игнорируют принципы симультанизации и когнитивного диоинтерфейса, которые мы считаем научно обоснованными.*** На наш взгляд, учебник только тогда можно назвать научным, если научное содержание облечено в научно-обоснованную форму. Но сегодня этого, как правило, нет. Нынешняя форма представления учебных материалов слишком часто тормозит или даже препятствует усвоению знаний и, следовательно, является скорее "антинаучной" (точнее, донаучной).

Если принять наше предложение и признать принцип когнитивного диоинтерфейса критерием научности, в итоге получим, что знаковое обеспечение науки во многих случаях почти полностью лишено научного обоснования. Налицо парадокс: наука является объективным знанием в знаковой форме, однако эта форма не удовлетворяет критерию научности. Таким образом, в самом сердце науки сохраняется донаучный стиль мышления. Данная работа представляет собой осторожную и вместе с тем решительную попытку поставить под сомнение господствующие, повсеместно распространенные, но устаревшие стереотипы научного мышления и предложить альтернативный подход под названием "проектоника" *(designomics)*.

ЧТО ТАКОЕ ПРОЕКТОНИКА?

На протяжении всей книги мы подчеркивали, что научной основой выдвигаемых положений и методов является когнитивная эргономика. В этом есть определенная условность, так как наука о человеческих факторах (эргономика) не является наукой в строгом смысле слова, а представляет собой весьма полезное, но явно недостаточное объединение частных теорий, экспериментальных фактов и соображений здравого смысла. Эргономика во многом опирается на психологическую науку, а последняя в ее нынешнем виде представляет собой совокупность конкурирующих концепций, плохо связанных между собой, постоянно вступающих в "драку" с нейробиологией и неспособных образовать целостную и непротиворечивую картину.

Впрочем, дело даже не в эргономике или каких-то частных недостатках психологии и когнитивной науки. Причина гораздо глубже и состоит в том, что современное человекознание (включая когнитивную науку, биологию, нейробиологию, психологию, эргономику, философию, антропологию, социологию, культурологию и т. д.) не располагает удовлетворительной теорией человека, человеческого мозга и интеллекта. Кроме того, в литературе все чаще звучит критика "психологических концепций, сложившихся в докомпьютерную эпоху".

Учитывая эти и другие соображения, автор пришел к выводу о необходимости разработки нового направления междисциплинарных исследований — проектоники.

Само собой разумеется, что проектоника как законченная система знания пока еще не существует. Речь идет всего лишь о проекте будущей науки, точнее, о едва намеченном замысле. С учетом этих оговорок про-

ектонику можно определить как теорию интенсификации человеческого интеллекта. Вероятно, в ее состав должна также войти прикладная теория проектирования систем “человек—диоинформация” (человек—знание), примеры которых показаны на рис. 139. Предлагаемую книгу можно рассматривать как самое предварительное изложение идей проектоники. Тезисы, изложенные в данной главе (после уточнения, доработки, развития и детализации) могли бы послужить ориентировочной основой для построения теоретической части проектоники. Многочисленные примеры, разбросанные на страницах книги, призваны подтвердить правильность и перспективность выдвигаемых теоретических положений.

ПРОЕКТОНИКА И ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ

Укажем различия между проектоникой и искусственным интеллектом. Как известно, “человеческий мозг является обязательным объектом исследования в искусственном интеллекте” [12]. С какой целью искусственный интеллект занимается изучением мозга? Ответ очевиден: узнав секреты естественного интеллекта и мозга, их можно использовать для создания искусственных умных машин, т. е. “автоматизированных систем, выполняющих те же функции, что и творческая личность, во всяком случае, в их простейших проявлениях” [13].

Легко заметить, что уже сама постановка вопроса о целях искусственного интеллекта как научной теории содержит пробел. В самом деле, изучая человеческий мозг, эта теория даже не пытается использовать с таким трудом полученные ценнейшие сведения о мозге для повышения интеллектуальной производительности... самого человеческого мозга. Справедливости ради следует сказать, что наиболее дальновидные специалисты по искусственному интеллекту хорошо понимают нелепость такого положения, признавая, что “в действительности именно человек является основным объектом изучения” [12]. В частности, Жан-Луи Лорьер полагает, что со временем “программы искусственного интеллекта будут иметь самостоятельную ценность независимо от современных компьютеров” [12]. Но что значит самостоятельную ценность? Для каких полезных целей могут быть использованы программы искусственного интеллекта, если брать их “независимо от современных компьютеров”? Лорьер, как впрочем и другие специалисты по искусственному интеллекту, даже не пытается ответить на вопрос.

Этот недостаток устраняет проектоника, которая ставит во главу угла вопрос о наращивании умственной продуктивности именно человеческого мозга, используя для решения этой задачи достижения как человекознания, так и искусственного интеллекта.

В некотором смысле можно сказать, что искусственный интеллект и проектоника (предметом которой являются естественный интеллект и методы повышения его эффективности) — это два ствола одного дерева, растущие из общего корня. Обе теории черпают исходные данные из одного и того же источника, тщательно анализируя добытые наукой факты о работе человеческого мозга и интеллекта. Однако используют эти факты существенно по-разному. Искусственный интеллект хочет, чтобы поумнели машины, а проектоника — чтобы поумнел человек.

Искусственный интеллект стремится улучшить представление и обработку знаний в компьютере, проектоника пытается поставить и решить ту же проблему по отношению к человеческому мозгу. Искусственный интеллект уподобляет машину человеку, пытаясь наделить ее "интеллектом". Проектоника, наоборот, ради исследовательских целей уподобляет человека машине, желая понять законы и алгоритмы функционирования мозга как биокомпьютера и, самое главное, сделать эти алгоритмы более продуктивными. Само собой разумеется, что две теории не противоречат друг другу и являются взаимодополняющими.

ОСОБЕННОСТИ ПРОЕКТОНИКИ

А теперь поднимем уровень обсуждения. Сверхзадача развиваемой теории состоит в том, чтобы прояснить вопрос: что нужно для эффективного и гармоничного управления деятельностью человека и жизнью общества? Ответ гласит: ***управление человеком и обществом всегда осуществляется через воздействие на мозг человека, поэтому для более эффективного управления нужны более эффективные знаки, знаковые системы и комплексы.***

На наш взгляд, почти все неприятности предыдущего этапа развития цивилизации объясняются тем, что процесс создания знаковой среды обитания (и, следовательно, процесс конструирования человеческого мозга) протекал стихийно, методом проб и ошибок, т. е. крайне неэффективно. Причина в том, что традиционная наука, являясь весьма удачным инструментом для проектирования предметной среды обитания людей, почти не имеет концептуальных средств для системного проектирования знаковой среды обитания. Теория проектирования высокоэффективных систем "человек—знание" представляет собой белое пятно на карте науки. Вследствие этого знаковое обеспечение современной цивилизации в значительной степени лишено научного обоснования.

Отсюда вытекает, что традиционные методы управления развитием цивилизации следует признать устаревшими. Они не учитывают то фундаментальное влияние, которое знаки оказывают на человеческий мозг и человеческое поведение, что во многих случаях делает развитие событий непредсказуемым и опасным. Иными словами, традиционный путь не дает гарантий предсказуемого, гармоничного, успешного и устойчивого развития мирового сообщества народов и в этом смысле исчерпал себя. ***Чтобы сделать человеческую деятельность гарантоспособной, необходимо перейти от стихийного знакотворчества к научно обоснованному проектированию знаковой среды обитания людей.*** В первую очередь это относится к разработке знакового обеспечения научной, технической, производственной, управленческой, учебной, медицинской, воспитательной, пенитенциарной, государственной и других важнейших видов деятельности.

В качестве первого шага можно взять уже имеющийся задел, созданный в рамках эргономики при проектировании средств отображения информации, т. е. перенести опыт, накопленный при создании систем "человек—техника", применив его к проектированию систем "человек—знание". Однако системы "человек—знание" отличаются качественным своеобразием, так что простой перенос идей невозможен. В особенности это касается знакового обеспечения сложных творческих задач, решае-

мых большими коллективами ученых, специалистов и программистов при разработке крупномасштабных технических и социальных проектов и технологий. В этом случае проблема приобретает качественно новый междисциплинарный характер, акцент переносится на проблему интенсификации интеллекта, которая требует принципиально новых подходов и методов. Таковы объективные предпосылки для выделения проектоники в самостоятельную область научных исследований.

МИРОИНФОРМАЦИЯ И МИРОИНТЕЛЛЕКТ

Каким образом можно повысить умственную продуктивность не только отдельного человека, но и всего человеческого общества?

Тезис 42. *Мироинформация* (мировая диоинформация) — полный объем диоинформации, созданной всем человечеством, включая компьютерную диоинформацию, профессиональную и учебную литературу, книги и документацию. Суть в том, что мировая диоинформация является мощным средством познания окружающего мира и важнейшим элементом социальной памяти человечества, имеющим уникальные когнитивные свойства. Поэтому она во многих случаях оказывает определяющее влияние на такие глобальные интеллектуальные характеристики, как интеллектуальная производительность и интеллектуальные возможности населения.

Тезис 43. *Мироинтеллект* (мировой разум) — совокупный интеллект коллективного мозга всего человечества.

Тезис 44. (Принцип детерминации мироинтеллекта). Среди многих факторов, влияющих на развитие и интенсификацию мироинтеллекта, мироинформация является главным, ключевым.

Чтобы убедиться в этом, полезно задать вопрос: что произойдет, если диоинформация внезапно и навсегда исчезнет с лица нашей планеты? Ответ очевиден: погаснут экраны дисплеев, замрут принтеры и плоттеры, факсы, ксероксы и печатные станки. Исчезнет письменность, и люди навсегда разучатся читать и писать. Учреждения и банки, заводы и стройки — все остановится и канет в небытие. Цивилизация исчезнет. Это означает, что современная интеллектуальная жизнь и современный интеллект перестанут существовать.

Из принципа детерминации мироинтеллекта вытекают важные следствия. Прежде всего отметим, что сегодня, как и тысячи лет назад, процесс когнитивного развития мироинформации не имеет научной основы и осуществляется стихийно, методом проб и ошибок, что серьезно тормозит развитие мироинтеллекта.

Чтобы устранить недостаток и серьезно улучшить работу ума, необходимо рассматривать мироинформацию как:

- одно из важнейших понятий современной науки;
- самостоятельный объект научного исследования;
- объект управления, причем целью управления является улучшение (развитие и интенсификация) мироинтеллекта, достигаемое путем улучшения когнитивного качества (эргономизации) мироинформации.

Решение этой проблемы является одной из перспективных задач проектоники.

СТРАТЕГИЧЕСКАЯ ИНТЕЛЛЕКТУАЛЬНАЯ ИНИЦИАТИВА

Развитие идей проектоники создает условия и стимулы для осуществления крупномасштабной реформы интеллектуального труда "Стратегическая интеллектуальная инициатива". Реформа должна быть комплексной и всеобщей, охватывая интеллектуальный труд в области науки, техники, образования, управления, экономики, политики, идеологии, а также в армии, службах безопасности и т. д. Она должна проводиться на основе метода когнитивно-эргономического проектирования мироинформации, совершенствования диоязыков, развития когнитивных информационных технологий.

Цель реформы — качественным образом улучшить мощь и могущество мироинтеллекта, резко увеличить его возможности, чтобы привести их в соответствие с принципиально новыми задачами немыслимой прежде сложности, которые встают сегодня перед мировой цивилизацией и требуют неотложного решения.

Для предварительного обсуждения можно предложить программу реформы из 20 пунктов, имея в виду, что этот перечень заведомо не полон и должен быть скорректирован и расширен по результатам обсуждения и критики.

1. Развертывание фундаментальных и прикладных исследований для организации научного обеспечения реформы. Резкое расширение программы исследований человеческого интеллекта и мозга. Тщательно продуманная координация этих исследований с изучением системы "мозг — мироинтеллект — мироинформация".
2. Формирование новой научной и учебной дисциплины, ориентированной на решение проблемы интенсификации интеллекта.
3. Формирование нового (когнитивного) мировоззрения в области интеллектуализации и информатизации общества.
4. Внесение соответствующих изменений в учебные программы системы образования, включая подготовку кадров государственной службы и международных организаций.
5. Подготовка специалистов и создание организационных структур, способных обеспечить практическую реализацию реформы интеллектуального труда и добиться существенного роста эффективности интеллекта на индивидуальном, локальном и глобальном уровнях.
6. Разработка нового поколения когнитивных информационных технологий, цель которых — максимизация продуктивности человеческого мозга.
7. Разработка когнитивных диоязыков, когнитивных компьютеров, программного обеспечения, сетей и средств телекоммуникации, обеспечивающих переход к массовому использованию когнитивной письменности в интересах улучшения интеллекта.
8. Создание нового полиграфического оборудования для выпуска когнитивных книг и периодических изданий, основанных на принципах когнитивной письменности для улучшения работы ума.

9. Разработка когнитивных компьютерных издательских систем, включая новое поколение когнитивных текстовых, графических и формульных редакторов для усиления социального интеллекта.
10. Переподготовка авторов книг и редакционно-издательских работников, обучение навыкам эргономизации и создания когнитивной научной, учебной и деловой литературы.
11. Использование когнитивных форм обучения в школах, лицеях, колледжах и университетах, выпуск когнитивных учебников и учебных пособий для интенсификации интеллекта.
12. Переподготовка персонала, в частности, специалистов по информатике, которые должны овладеть когнитивной культурой.
13. Эргономизация диодокументации на любые продукты и изделия, включая программное обеспечение. Создание новых (когнитивных) форм документации, обеспечивающих улучшение работы ума.
14. Изменение способов представления информации в науке, технике, образовании, управлении, на производстве и в других сферах профессиональной деятельности, использование когнитивных форм представления диоинформации для облегчения умственной работы.
15. Организация консультационных пунктов для ученых, специалистов, соискателей, работников образования, помогающих овладеть методами когнитивного представления научных проблем, научных работ, докладов, диссертаций, учебников и учебных пособий.
16. Внедрение когнитивной письменности в органах государственной власти: Администрации Президента, Правительстве, Государственной Думе, Совете Федерации, министерствах и ведомствах, региональных органах власти и управления.
17. Создание когнитивных зданий, кварталов, городов и территорий для интеллектуализации населения.
18. Переработка существующих национальных и международных стандартов, определяющих порядок производства и визуального представления любых форм научной, профессиональной, служебной и учебной диоинформации, включая компьютерную, книги и документацию; создание когнитивных стандартов.
19. Аттестация и сертификация когнитивного качества информационных товаров и услуг, в первую очередь диоязыков, информационных технологий, учебников, руководств и т. д. Организация международной системы когнитивной аттестации и сертификации.
20. Разработка и внедрение новых форм интеллектуальной деятельности в науке, технике, управлении, экономике, политике, экологии, других сферах деятельности, чтобы кардинально увеличить мощь и могущество человеческого интеллекта и тем самым обеспечить выживание человечества и устойчивое будущее цивилизации.

* * *

Интеллект, понимаемый как полная совокупность возможностей мозга, есть единственное средство, озаряющее человеческий путь во мраке мироздания. Это единственный инструмент, которым располагает человек, чтобы обезопасить себя от нежелательных последствий своей планетарной деятельности, единственное средство, способное спасти цивилизацию от нарастающего системного кризиса (экологического, демографического, энергетического и т. д.). Однако нынешний уровень

мироинтеллекта слишком низок, слаб и недостаточен для решения задачи. Поэтому нужно кардинальным образом усилить его возможности. Другого пути просто нет.

Отсюда вытекает лозунг стратегической интеллектуальной инициативы: “Устойчивое развитие и процветание цивилизации — через интенсификацию интеллекта!”

ДОРОГА В БУДУЩЕЕ
(Вместо заключения)

ИНТЕЛЛЕКТУАЛЬНЫЕ ТРУДНОСТИ КАК ГЛОБАЛЬНАЯ ПРОБЛЕМА

Подведем окончательные итоги. В этой книге автор попытался показать, что нынешние методы интеллектуальной работы, используемые во всем мире, устарели и нуждаются в радикальном обновлении. Есть множество препятствий, тормозящих работу мозга. Если их убрать, скорость мышления *значительно* возрастет. Исходя из этого, поставлена амбициозная задача: *улучшить человеческий интеллект,* т. е. усовершенствовать интеллектуальную деятельность в науке, технике, бизнесе, образовании и т. д.

Предлагаемая идея опирается на критерий Декарта, из которого вытекает важное следствие: *чтобы улучшить работу ума, надо минимизировать интеллектуальные усилия, затрачиваемые на получение нужного результата*. Человек прилагает интеллектуальные усилия, чтобы преодолеть интеллектуальные трудности. Отсюда вывод: чтобы минимизировать усилия, надо выявить трудности и по возможности их устранить.

Говоря о трудностях, автор имеет в виду микроскопические интеллектуальные затруднения, такие, например, как нарушение правила главного маршрута (см. гл. 6). Каждая отдельная трудность сама по себе ничтожна и не приносит заметного вреда. Проблема в том, что их очень много. Суммируясь, они превращаются в бедствие.

На нашей планете миллионы интеллектуальных работников и миллиарды учащихся. Все они испытывают множество интеллектуальных трудностей, часто неосознанных. Таким образом, трудности воздействуют на огромное число людей, затрагивая почти все человечество. В связи с этим *мы рассматриваем интеллектуальные трудности как глобальную проблему, которая наносит людям громадный ущерб*. Этот ущерб можно и нужно исключить или уменьшить. Но как это сделать?

Общий ответ таков: необходимо тщательно и скрупулезно выявлять мельчайшие неудобства и препятствия, мешающие людям думать, и последовательно, шаг за шагом, их устранять. Это большая, кропотливая работа. Дело усугубляется тем, что существует вредная привычка игнорировать проблему интеллектуальных трудностей, рассматривать ее как недостойную внимания мелочь.

Вот пример. Авторы трудных, “уму непостижимых” учебников не задумываются о том, сколько слез прольют читатели. Печальная правда такова: по отношению к читателям авторы учебной, научной и профессиональной литературы часто ведут себя как интеллектуальные террористы. Чтобы поправить дело, *нужно в корне изменить принципы создания подобных книг*. Нужно помочь авторам, вооружить их хорошим методом, позволяющим улучшить когнитивно-эргономические характеристики книг, чтобы они стали удобными для *быстрого* понимания, помогали читателю не разбазаривать, а экономить драгоценное время.

ВЫЗОВ ИНТЕЛЛЕКТУАЛЬНОГО ТЕРРОРИЗМА

Термин интеллектуальные трудности плохо запоминается и производит впечатление чего-то мелкого. Он не способен зажечь сердца, вызвать эмоции и показать планетарный масштаб проблемы. Поэтому мы заменили его на яркое и броское название “интеллектуальный терроризм”.

Интеллектуальный терроризм ослабляет умственный потенциал человечества, серьезно тормозит развитие цивилизации. Он уменьшает творческую продуктивность мозга, мешает использовать богатейшие ресурсы симультанного восприятия, обрекая интеллектуальных работников и учащихся на “частичную слепоту” (см. гл. 5). Он многократно увеличивает сложность интеллектуальных задач и резко замедляет их решение. В результате умственная работа, которую можно выполнить за месяцы, нередко затягивается на годы. Это прискорбное замедление касается почти всех видов интеллектуальной работы в науке, технике, образовании, управлении, государственной службе и т. д.

БЕССИЛИЕ ИНТЕЛЛЕКТА

У медали есть и другая сторона. В последнее время развитие цивилизации сталкивается с серьезными трудностями, глобальные проблемы обостряются. Попытки их решения не приводят к ощутимому успеху. Все отчетливее проявляет себя неспособность человеческого разума найти решение многих жгучих проблем современности. К числу последних относятся: непрерывные военные конфликты, огромные военные расходы, расползание ядерного оружия, преступность, перенаселенность, нищета, социальные взрывы, религиозный экстремизм, загрязнение среды, утоньшение озонового слоя, опасные процессы в биосфере, рост концентрации парниковых газов, глобальное потепление и возможное повышение уровня океана, исчерпание невозобновимых ресурсов, астероидная опасность и пр.

Для решения названных проблем необходимы беспрецедентные усилия. Роль интеллекта в судьбах мира становится решающей — без целенаправленного вмешательства разума, разума всего человечества, решить этот клубок проблем за приемлемое время вряд ли удастся. Вместе с тем приходится констатировать, что нынешний интеллект человечества слишком слаб и явно недостаточен для решения столь сложных задач. В результате ситуация нередко выходит из-под контроля, вызывая серьезные негативные последствия локального или глобального характера. Цена ошибок, бездействия и слабости интеллекта стала недопустимо высокой.

ЦЕЛЬ — ЗНАЧИТЕЛЬНОЕ УЛУЧШЕНИЕ ИНТЕЛЛЕКТА

Можно предположить, что *улучшение интеллекта может внести весомый вклад в решение глобальных проблем, преодоление системного кризиса цивилизации.* С учетом этих и других соображений в книге выдвинута “Стратегическая интеллектуальная инициатива”, цель которой — существенное усиление человеческого интеллекта. Она позволит кардинально увеличить интеллектуальную мощь общества, ускорить решение многих жизненно важных проблем, которые сегодня не поддаются решению.

Материалы и предложения, изложенные в книге, — результат 40-летних интенсивных исследований, которые показали, что человеческий интеллект *действительно* можно улучшить. Применяя описанные в книге и другие аналогичные методы *на практике*, автор многократно убеждался, что они *работают*, причем работают эффективно. Размыш-

ления об этом и привели к выводу о необходимости крупномасштабной реформы интеллектуального труда.

К сожалению, сегодня проблема улучшения интеллекта находится в тени. Современное общество фактически игнорирует ее, не рассматривая вопрос об улучшении интеллекта в числе приоритетных. Еще вчера такое положение, вероятно, было оправданно, так как надежного метода улучшения интеллекта просто не было. Но ситуация изменилась — в этой книге удалось сделать важный шаг к его созданию. Опираясь на достижения многих ученых, автор разработал теорию эргономичных алгоритмов и на конкретных примерах показал, что такие алгоритмы действительно позволяют улучшить работу ума (см. гл. 6—18). Конечно, это всего лишь частный случай. Однако есть уверенность, что предлагаемый подход окажется эффективным и в других случаях. Например, в гл. 19 автор подверг критике когнитивно-эргономические основания математики и показал возможность ее эргономизации, т. е. улучшения работы ума при решении математических проблем и задач.

Проблема улучшения интеллекта *созрела для практического решения.* Пришла пора поместить ее в центр внимания, привлечь к обсуждению не только ученых, но и общественность, а также политиков, так как успех крупномасштабных преобразований во многом зависит от общественного одобрения и политической воли.

И последнее. Вести работу по улучшению интеллекта чрезвычайно трудно. Но трудность эта особого рода. Она связана с необходимостью болезненной ломки устаревших представлений, которые до сих пор владеют умами многих уважаемых ученых. Речь идет о серьезной переоценке ценностей, так как предлагаемый *принцип эргономизации науки* требует изменения взглядов на саму сущность науки, пересмотра критериев научности. Повторим сказанное в гл. 20: **данная работа представляет собой осторожную и вместе с тем решительную попытку поставить под сомнение господствующие, повсеместно распространенные, но устаревшие стереотипы научного мышления и предложить альтернативный подход под названием “проектоника” (теория интенсификации интеллекта).**

Но главная цель книги лежит не в области теории, а в области *практики*. Нужно во много раз увеличить продуктивность человеческого ума в реальной жизни — в условиях каждодневной умственной работы в организациях, фирмах, офисах, научных центрах, учебных заведениях и т. д. Это можно и обязательно нужно сделать.

СПИСОК ЛИТЕРАТУРЫ

ИНТЕЛЛЕКТУАЛЬНЫЙ ТЕРРОРИЗМ: ФАНТАЗИЯ ИЛИ РЕАЛЬНОСТЬ?

1. *Лорьер Ж.-Л.* Системы искусственного интеллекта. М.: Мир, 1991. С. 6, 7.
2. Интеллектуальная культура специалиста. Новосибирск: Наука, 1988.
3. Междисциплинарный подход к исследованию научного творчества. М.: Наука, 1990.
4. *Видинеев Н. В.* Природа интеллектуальных способностей человека. М.: Мысль, 1989.
5. *Пиаже Ж.* Психология интеллекта // Ж. Пиаже. Избранные психологические труды. М.: Просвещение, 1969.
6. *Венда В. Ф.* Системы гибридного интеллекта: эволюция, психология, информатика. М.: Машиностроение, 1990.
7. *Альтшулер Г. С.* Найти идею. Введение в теорию решения изобретательских задач. Новосибирск: Наука, 1991.
8. *Скотт Д. Г.* Сила ума. Описание пути к успеху в бизнесе. Киев: Век, 1991.
9. *Клейн Ф.* Лекции о развитии математики в XIX столетии: В 2 т. Т. 1. М.: Наука, 1989. С. 424.
10. Психологические проблемы автоматизации научно-исследовательских работ. М.: Наука, 1987. С. 10.
11. *Трофимов Ю. Л.* Техническое творчество в САПР. (Психологические аспекты). Киев: Вища школа, 1989. С. 6.
12. *Декарт Р.* Избр. произв. М.: Госполитиздат, 1950. С. 80, 89.
13. *Катасонов В. Н.* Метафизическая математика XVII в. М.: Наука, 1993. С. 32.

ГЛАВА 1

1. *Martin J., McClure C.* Diagramming Technique for Analysts and Programmers. N. J.: Prentice Hall, Inc., 1985. P. 81–120, 208–350.
2. *Martin J.* Recommended Diagrammed Standards for Analysts and Programmers. N. J.: Prentice Hall, Inc., 1985.
3. *Martin J., McClure C.* Action Diagramms: Clearly Structured Specifications, Programs and Procedures. Second Edition. N. J.: Prentice Hall, 1989.
4. *Martin J.* Rapid Application Development. N.-Y.: Macmillan Publishing Co., 1991. P. 399–439, 611–640.
5. *Фаулер М., Скотт К.* UML в кратком изложении. Применение стандартного языка объектного моделирования. М.: Мир, 1999. С. 146–158.
6. *Nassi I., Schneiderman B.* Flowchart Techniques for Structured Programming // ACM SIGPLAN Notices. 1973. Vol. 8. No 8. P. 12–26.
7. *Belady L. A., Evangelisti C. J., Power L. R.* GREENPRINT: a Graphic Representation of Structured Programs // IBM Syst. J. 1980. Vol. 19. No 4. P. 542–553.
8. Modern Software Engineering. Foundations and Current Perspectives. Edited by A. Ng. Peter, T. Yeh Raymond. N.-Y.: Van Nostrand Reinhold, 1990.
9. *Пароджанов В. Д.* Перспективы информационных технологий и повышение продуктивности интеллектуального труда // НТИ. Сер. 1. 1993. № 5.
10. *Солсо Р. Л.* Когнитивная психология. М.: Тривола, 1996.
11. *Симонов П. В.* Предисловие // *Д. Норман.* Память и научение. М.: Мир, 1985. С. 5.

ГЛАВА 3

1. *Сергеев Б. Ф.* Ступени эволюции интеллекта. Л.: Наука, 1986. С. 189.
2. *Зенкин А. А.* Когнитивная компьютерная графика и научное творчество // Будущее искусственного интеллекта. М.: Наука, 1991. С. 274.

3. *Кукушкин В. Д., Неволин И. Ф., Бушуев В. С.* Организация умственного труда. Ч. 1. М.: МИСИС, 1976. С. 60.
4. Кибернетика и логика. Математико-логические аспекты становления идей кибернетики и развития вычислительной техники. М.: Наука, 1978. С. 192, 207, 208.
5. *Martin J., McClure C.* Diagramming Technique for Analysts and Programmers. Prentice Hall, 1985. P. 1–3.
6. *Паронджанов В. Д.* Знаковая революция как движущая сила НТР // Теоретические вопросы истории техники и научно-технического прогресса. М.: Наука, 1994.
7. Кибернетика и вычислительная техника. Вып. 1. М.: Наука, 1985. С. 207.
8. *Громов Г. Р.* Очерки информационной технологии. М.: Инфоарт, 1993. С. 143–158, 265–299.
9. Программные средства вычислительной техники. Толковый терминологический словарь-справочник. М.: Изд-во стандартов, 1990. С. 21.
10. Краткий экономический словарь. М.: Политиздат, 1987. С. 331.
11. Научно-технический прогресс. Словарь. М.: Политиздат, 1987. С. 293, 349.
12. Толковый словарь по искусственному интеллекту. М.: Радио и связь, 1992. С. 36.
13. *Джорджеф М. П., Лэнски Э. Л.* Процедурные знания // ТИИЭР. 1986. Т. 74. № 10. С. 101.
14. *Иванов В. Н.* Социальные технологии в современном мире. Москва—Нижний Новгород: Изд-во Волго-Вятской академии госслужбы, 1996.
15. Социальные технологии: Толковый словарь: Центр социальных технологий. Москва—Белгород: Луч, 1995.
16. *Martin J.* Rapid Application Development. N.-Y.: Macmillan Publishing Co., 1991. P. 2–6, 36–40.
17. *Woodward J. F.* Science in Industry: Science of Industry. An Introduction to the Management of Technology-Based Industry. Aberdeen: Aberdeen University Press, 1982. P. 13.

ГЛАВА 5

1. *Хухо Ф.* Нейрохимия. Основы и принципы. М.: Мир, 1990. С. 9–14.
2. *Митькин А. А.* Системная организация зрительных функций. М.: Наука, 1988. С. 46, 117, 136, 153.
3. *Parondjanov V. D.* Intensification of the Students' Intellect and the Theory of Intensive Distance Education // Distance Learning and New Technologies in Education: Proc. of the First International Conference on Distance Education in Russia. M.: Association for International Education, 1994. P. 415, 416.
4. *Глезер В. Д.* Зрение и мышление. СПб.: Наука, 1993. С. 14, 253.
5. *Венда В. Ф.* Средства отображения информации (Эргономические исследования и художественное конструирование). М.: Энергия, 1969. С. 165, 167, 168.
6. *Шнейдерман Б.* Психология программирования. Человеческие факторы в вычислительных и информационных системах. М.: Радио и связь, 1984. С. 12, 22–46.
7. *Кертис Б. и др.* Психология программных систем: о необходимости междисциплинарной комплексной программы исследований // ТИИЭР. 1986. Т. 74, № 8. С. 43.
8. *Martin J.* Rapid Application Development. N.-Y.: Macmillan Publishing Co., 1991. P. 607.
9. *Martin J., McClure C.* Diagramming Technique for Analysts and Programmers. N. J.: Prentice Hall, Inc., 1985. P. 26, 27.
10. Modern Software Engineering. Foundation and Current Perspectives. N. Y.: Van Nostrand Reinhold, 1990.
11. *Боэм Б. У.* Инженерное проектирование программного обеспечения. М.: Радио и связь, 1985. С. 488.

ГЛАВА 6

1. *Shu N. C.* Visual Programming. N.-Y.: Van Nostrand Reinhold Comp., 1988.
2. ГОСТ 19.701–90. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. М.: Изд-во стандартов, 1991.
3. *Вельбицкий И.* Знакомьтесь, Р-технология // НТР: проблемы и решения. 1987. № 13. С. 5.
4. *Шнейдерман Б.* Психология программирования. Человеческие факторы в вычислительных и информационных системах. М.: Радио и связь, 1984. С. 90, 92.

5. *Лингер Р., Миллс Х., Уитт Б.* Теория и практика структурного программирования. М.: Мир, 1982. С. 124–126, 139–146.
6. *Йодан Э.* Структурное проектирование и конструирование программ. М.: Мир, 1979. С. 185–196.

ГЛАВА 13

1. *Кемп П., Арнс К.* Введение в биологию. М.: Мир, 1988. С. 612, 613.
2. Руководство по профилактической медицине. М.: Новая слобода, 1993. С. 40.
3. Философский словарь. М.: Политиздат, 1991. С. 114, 115.
4. Психология. Словарь. М.: Политиздат, 1990. С. 101–103.
5. Математическая энциклопедия. М.: Сов. энциклопедия, 1977. Т. 1. С. 202–210.

ГЛАВА 16

1. *Bohm C., Jacopini G.* Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules // Comm. ACM. 1965. Vol. 9, N 5. P. 366–371.
2. *Дал У., Дейкстра Э., Хоор К.* Структурное программирование. М.: Мир, 1975. С. 25–28.
3. *Лингер Р., Миллс Х., Уитт Б.* Теория и практика структурного программирования. М.: Мир, 1982. С. 100, 102, 120, 121, 123, 141, 142.
4. *Майерс Г.* Надежность программного обеспечения. М.: Мир, 1980. С. 134, 135, 137, 138, 150.
5. *Хьюз Дж., Мичтом Дж.* Структурный подход к программированию. М.: Мир, 1980. С. 24, 73, 80.
6. *Брусенцов Н. П.* Микрокомпьютеры. М.: Наука, 1985.
7. *Сауле Я. В.* Перспективы построения программного обеспечения // Программирование. 1985. № 6. С. 34.
8. Знакомьтесь, Р-технология // НТР: проблемы и решения. 1987, № 13, 7–20 июля. С. 4, 5.
9. Толковый словарь по вычислительным системам. М.: Машиностроение, 1991. С. 463.
10. *Очков В. Ф., Пухначев Ю. В.* 128 советов начинающему программисту. М.: Энергоатомиздат, 1992. С. 21.
11. *Котов В. Е., Сабельфельд В. К.* Теория схем программ. М.: Наука, 1991. С. 71, 76–78.
12. *Касьянов В. Н.* Оптимизирующие преобразования программ. М.: Наука, 1988. С. 35.
13. *Евстигнеев В. А.* Применение теории графов в программировании. М.: Наука, 1985. С. 5.
14. *Янов Ю. И.* О локальных преобразованиях схем алгоритмов // Проблемы кибернетики. 1968. Вып. 20. С. 201.
15. *Ершов А. П.* Современное состояние схем программ // Проблемы кибернетики. 1973. Вып. 27. С. 87–110.
16. *Венда В. Ф.* Инженерная психология и синтез систем отображения информации. М.: Машиностроение, 1982. С. 300–302.
17. *Криницкий Н. А.* Алгоритмы вокруг нас. М.: Наука, 1984. С. 102.
18. *Питерс Л. Дж.* Методы отображения и компоновки программных средств // ТИИЭР. 1980. Т. 68, № 9. С. 60.
19. *Рейуорд-Смит В.* Дж. Теория формальных языков. Вводный курс. М.: Радио и связь, 1988. С. 28, 30, 44.
20. *Левкин Г. Н., Левкина В. Е.* Demo Turbo C. 98 тем для начинающего программиста. Витебск: Прок, 1992. С. 35.
21. *Уэйт М., Прата С., Мартин Д.* Язык Си. М.: Мир, 1988. С. 237, 239, 240.
22. *Дейкстра Э.* Дисциплина программирования. М.: Мир, 1978.

ГЛАВА 17

1. *Ершов Ю. Л., Палютин Е. А.* Математическая логика. М.: Наука, 1979. С. 12, 13.
2. *Клини С. К.* Введение в метаматематику. М.: ИЛ, 1957. С. 59–61.
3. *Колеватов В. А.* Социальная память и познание. М.: Мысль, 1984. С. 133.
4. *Зенкин А. А.* Когнитивная компьютерная графика. М.: Наука, 1991.
5. *Кондаков Н. И.* Логический словарь-справочник. М.: Наука, 1976. С. 101, 285.
6. *Поспелов Г. С.* Искусственный интеллект основа новой информационной технологии. М.: Наука, 1988. С. 41.
7. *Котов В. Е., Сабельфельд В. К.* Теория схем программ. М.: Наука, 1991. С. 67–82.
8. *Касьянов В. Н.* Оптимизирующие преобразования программ. М.: Наука, 1988. С. 35, 228.

9. *Ершов А. П.* Введение в теоретическое программирование. М.: Наука, 1977. С. 226–281.
10. *Андерсон Р.* Доказательство правильности программ. М., 1988. С. 152.
11. *Грис Д.* Наука программирования. М.: Мир, 1984. С. 303.
12. *Хлебцевич Г. Е., Цыганкова С. В.* Визуальный стиль программирования: понятия и возможности // Программирование. 1990. № 4. С. 78.
13. *Ершов А. П.* Операторные алгоритмы. III (Об операторных схемах Янова) // Проблемы кибернетики. Вып. 20. М.: Наука, 1968. С. 187, 189.
14. *Бульонков М. А., Кочетов Д. В.* Схема эффективной специализации императивных программ // Программирование. 1995. № 5. С. 33.
15. *Непейвода Н. Н.* Выводы в форме графов // Семиотика и информатика. Вып. 26. М.: ВИНИТИ, 1985.

ГЛАВА 18

1. *Зараковский Г. М., Зинченко В. П.* Анализ деятельности оператора // Эргономика. Принципы и рекомендации. Вып. 1. М.: ВНИИТЭ, 1970.
2. *Галактионов А. И.* Инженерная психология // Тенденции развития психологической науки. М.: Наука, 1989. С. 136–143.
3. Человеческий фактор: В 6 т. М.: Мир, 1991. Т. 3, 4.
4. *Хоггер К.* Введение в логическое программирование. М.: Мир, 1985. С. 14.
5. *Осуга С.* Обработка знаний. М.: Мир, 1989. С. 179.
6. *Лорьер Ж.-Л.* Системы искусственного интеллекта. М.: Мир, 1991. С. 420, 421, 443.
7. *Доорс Дж. и др.* Пролог — язык программирования будущего. М.: Мир, 1990.
8. Практикум по инженерной психологии и психологии труда / Т. П. Зинченко, Г. В. Суходольский, М. А. Дмитриева и др. Л.: ЛГУ, 1983. С. 146.
9. *Разумов А. Н., Косолапов О. А., Пикалов Д. А.* Психофизиологическая подготовка членов экипажей многоместных летательных аппаратов к взаимодействию в аварийных ситуациях // Военно-медицинский журнал. 1991. № 12. С. 47, 48.
10. *Зайдельман Я. Н., Лебедев Г. В., Самовольнова Л. Е.* Три кита школьной информатики // Информатика и образование. 1993. № 3. С. 19; № 4. С. 14.
11. Деятельность: теория, методология, проблемы. М.: Политиздат, 1990.
12. Философские проблемы деятельности (Материалы круглого стола) // Вопр. философии. 1985. № 2, 3, 5.
13. *Игнатенко Е. И.* Чернобыльская авария и ликвидация ее последствий // Информационный бюллетень, 1989. М.: ЦНИИ атоминформ, 1990.
14. Чернобыльская катастрофа: Причины и последствия (Эксперт. заключение). В 4 ч. Ч. 1: Непосредственные причины аварии на Чернобыльской АЭС. Дозиметрический контроль. Меры защиты и их эффективность. Минск: Тест, 1993. С. 3–122.
15. Всемирный день окружающей среды. Информационные материалы 1990. М.: ВИНИТИ, 1991. С. 32, 37.
16. *Авиженис А., Лапри Ж.-К.* Гарантоспособные вычисления: от идеи до реализации в проектах // ТИИЭР. 1986. Т. 74, № 5. С. 8.
17. *Пароджанов В. Д.* Кризис цивилизации и нерешенные проблемы информатизации // НТИ. Сер. 2. 1993. № 12.
18. *Болошин И. А.* Еще раз о кризисе цивилизации и нерешенных проблемах информатизации // НТИ. Сер. 2. 1994. № 9.
19. *Перминов О. Н.* Программирование на языке Паскаль. М.: Радио и связь, 1988. С. 4, 5.
20. *Дейкстра Э.* Дисциплина программирования. М.: Мир, 1978. С. 9.
21. Лекции лауреатов премии Тьюринга. М.: Мир, 1993. С. 392, 393.

ГЛАВА 19

1. *Барабашев А. Г.* Будущее математики. Методологические аспекты прогнозирования. М.: МГУ, 1991. С. 19, 20.
2. *Клайн М.* Математика. Поиск истины. М.: Мир, 1985. С. 269.
3. *Диофант.* Арифметика. М.: Наука, 1974.
4. *Никифоровский В. А.* В мире уравнений. М.: Наука, 1987. С. 47–57, 104–124.
5. *Кликс Ф.* Пробуждающееся мышление. История развития человеческого интеллекта. Киев: Вища школа, 1985. С. 236, 237.
6. *Стройк К. Л.* Краткий очерк истории математики. М.: Наука, 1990. С. 83–118, 139–143, 213, 214.

7. *Кукушкин В. Д., Неволин И. Ф., Бушуев В. С.* Организация умственного труда. М.: МИСИС, 1976. С. 134–136, 141.
8. *Crowe M. J.* A History of Vector Analysis. The Evolution of the Idea of a Vectorial Systems. N.-Y., 1985. P. 217.
9. *Кузнецова Н. И.* Наука в ее истории. Методологические проблемы. М.: Наука, 1982. С. 103–105.
10. *Нейгебауэр О.* Точные науки в древности. М.; 1968. С. 67.
11. *Алиев Т. М., Вигдоров Д. И., Кривошеев В. П.* Системы отображения информации. М.: Высшая школа, 1988. С. 10, 57–64.
12. ACM SIGGRAPH Computer Graphics. 1987. Vol. 21. No 6.
13. *Зенкин А. А.* Когнитивная компьютерная графика и научное творчество // Будущее искусственного интеллекта. М.: Наука, 1991. С. 270–274.
14. *Гантмахер Ф. Р.* Теория матриц. М.: Наука, 1988. С. 141, 142.
15. *Пароджанов В. Д.* Операторная теория сопровождающих матриц и линейные переключающие схемы. Рукопись статьи депонирована в ЦНТИ "Поиск". Реферат рукописи опубликован в журнале "Промышленно-технический опыт", 1982, № 5.

ГЛАВА 20

1. *Выготский Л. С.* Собр. соч. в 6 т. Т. 3. М.: Педагогика, 1983. С. 80.
2. *Полторацкий А., Швырев В.* Знак и деятельность. М.: Политиздат, 1970. С. 6.
3. Роль сенсорного притока в созревании функций мозга. М.: Наука, 1987.
4. *Сахаров Д. А.* Предисловие редактора перевода / Г. Шеперд. Нейробиология. В 2 т. Т. 1. М.: Мир, 1987. С. 6.
5. *Блум Ф., Лейзерсон А., Хофстедтер Л.* Мозг, разум и поведение. М.: Мир, 1988. С. 73.
6. *Вартанян Г. А., Пирогов А. А.* Нейробиологические основы высшей нервной деятельности. Л.: Наука, 1991. С. 153–156.
7. Научное знание: логика, понятия, структура. Новосибирск: Наука, 1987. С. 154.
8. *Born-Andersen N.* Are "Human Factors" Human? // Computer j. British Computer Society. 1988. Vol. 31, N 5. P. 386–390.
9. Visualization. Using Computer Graphics to Explore Data and Present Information / J. R. Brown, R. Earnshow, M. Jern, J. Vince. John Wiley & Sons, Inc., 1995.
10. Педагогика, 1996. № 5. С. 20, 21.
11. *Микк Я. А.* Оптимизация сложности учебного текста. М.: Просвещение, 1991. С. 3, 9.
12. *Лорьер Ж.-Л.* Системы искусственного интеллекта. М.: Мир, 1991. С. 13, 18.
13. Представление и использование знаний. М.: Мир, 1989. С. 5.

С автором можно связаться
по тел.: (095) 331-50-72, (095) 334-34-13, факс: (095) 336-54-00

Владимир Даниелович ПАРОНДЖАНОВ

КАК УЛУЧШИТЬ РАБОТУ УМА
Алгоритмы без программистов — это очень просто!

Гл. редактор *Ю. В. Луизо*
Зав. редакцией *Г. Г. Кобякова*
Художники *Н. В. Пьяных, С. А. Ульянов*
Компьютерная подготовка оригинал-макета *Ю. С. Лобанов, Т. А. Лобанова*
Технический редактор *Л. А. Зотова*
Корректоры *Л. И. Трифонова, Е. И. Борисова*

Лицензия ИД № 03590 от 19.12.2000 г.
Гигиеническое заключение № 77.99.2.953.П.16308.12.00 от 01.12.2000 г.

Подписано в печать 24.08.2001. Формат 60 × 90 $^{1}/_{16}$.
Бумага офсетная. Гарнитура Таймс. Печать офсетная.
Усл. печ. л. 22,5. Тираж 3000 экз. Заказ № 899. Изд. № 188

Издательство “Дело”
117571, Москва, пр-т Вернадского, 82
Коммерческий отдел — тел.: 433-2510, 433-2502
E-mail: delo@ane.ru
Internet: http://www.delo.ane.ru

Отпечатано в Московской типографии № 6
Министерства Российской Федерации по делам печати, телерадиовещания
и средств массовых коммуникаций
109088, Москва, Ж-88, Южнопортовая ул., 24

ОТЗЫВЫ

о книге В. Пароңджанова “Как улучшить работу ума”

В высшей степени интересная и прекрасно написанная книга.

***Виталий Кутепов**,*
профессор

Автор затрагивает вопросы, которые волнуют многих, и предлагает неожиданные ответы. Очень нужная книга.

***Сергей Падалко**,*
профессор

Оригинальность идеи и глубина ее развития буквально потрясают.

***Виктор Взятышев**,*
профессор

Хорошая книга. Много новых идей, чрезвычайно полезных для практики.

***Павел Пархоменко**,*
член-корреспондент Российской академии наук

Графический язык высокого уровня, непосредственно преобразующий чертеж алгоритма в компьютерную программу — вещь очень современная. Такой язык и разработал Владимир Пароңджанов. Он называется “Дракон”. Есть совершенно бесспорная сторона в открытии Пароңджанова. “Дракон” — это эргономичный стандарт для графического представления информации. Это безусловно первый и единственный такой стандарт. Блок-схемы во всех имеющихся на сегодня книгах (кроме книг Пароңджанова) составлены очень плохо. Пароңджанов учит правильному составлению блок-схем. В этом он безусловно новатор. Насколько нам известно, нет другой литературы, где тому же самому можно научиться столь просто и даже увлекательно.

Учительская газета