ЗАЧЕМ НУЖНА ИНФОРМАЦИОННАЯ ТЕХНОЛОГИЯ «ГРАФИТ-ФЛОКС»?

(Вводный раздел)

> Индустрии программирования необходимо чудо – чудо, которое воплотило бы в жизнь мечту о быстрой и легкой разработке программ.
>
> *Том Мануэль* [22]

В НПЦ автоматики и приборостроения разработана новая информационная технология ГРАФИТ- ФЛОКС, предназначенная для значительного повышения производительности труда больших коллективов, участвующих в производстве алгоритмов и программ. Данная технология создана на основе обобщения опыта проектирования орбитального корабля «Буран». И опробована на практике в ходе разработки системы управления доразгонного модуля проекта «Морской старт» (Sea launch) [1].

В перспективе технология ГРАФИТ- ФЛОКС может послужить основой для построения *Единой технологии разработки алгоритмов и программ для встроенных ЭВМ* ракетно-космической отрасли Российской Федерации. Проектирование *Единой технологии* будет проводиться на основе широких консультаций с представителями Российского Космического агентства, Министерства обороны и заинтересованных предприятий.

КАК ПОВЫСИТЬ ПРОИЗВОДИТЕЛЬНОСТЬ ТРУДА ПРИ РАЗРАБОТКЕ СЛОЖНЫХ ПРОГРАММНЫХ КОМПЛЕКСОВ?

Важные задачи

При создании алгоритмов и программного обеспечения ракетно-космических комплексов и других систем управления реального времени важными задачами являются:

- повышение производительности труда постановщиков задач, алгоритмистов, комплексников, программистов, стендовиков, испытателей и других участников технологической цепочки производства и сопровождения программного обеспечения;
- сокращение сроков разработки и отработки, уменьшение стоимости систем управления и смежных систем;
- улучшение качества алгоритмов, повышение надежности программного обеспечения;
- минимизация суммарных затрат на программное обеспечение за время жизненного цикла согласно принципу «затраты – результат» с учетом требований рыночной экономики.

Драматическое противоречие

Особую остроту эти задачи приобретают сегодня, в эпоху массовой компьютеризации и неуклонного роста сложности проектируемых ракетно-космических систем. Разработка подобных систем неизбежно влечет за собой усложнение интеллектуальных задач, увеличение нагрузки на мозг разработчиков. В свою очередь, увеличение нагрузки на нервную систему, интенсификация и рост напряженности умственного труда приводят к перегрузке мозга и, как следствие, к сбоям в его работе, увеличивая вероятность заблуждений, просчетов, ошибок, недоработок и упущений. В конечном итоге это может привести и нередко приводит к снижению качества работ, многократным переделкам конструкции и программного обеспечения, нарушению графика и перерасходу средств [36].

Стремительное и повсеместное усложнение всех аспектов профессиональной деятельности, острая потребность в овладении большим объемом разноплановых знаний вступают в драматическое противоречие с ограниченными возможностями человеческого мозга перерабатывать поступающую информацию [28].

Что важнее: компьютеры или человеческий мозг?

Уже давно признано: при разработке сложных систем узким местом стали не столько вычислительные, сколько человеческие ресурсы [15]. Поэтому задача экономии *человеческих*, а не *машинных* ресурсов стала центральной для технологии программирования [11]. По мере роста мощности и снижения удельной стоимости компьютеров все более негативное влияние на сроки и стоимость исследований, разработок и других 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 [38].

Мировой опыт показывает: большинство организаций расходует слишком много средств на информационный продукт, а получает незначительную отдачу из-за низкой творческой производительности персонала [45]. В этих условиях требование повысить производительность именно интеллектуального работника (т.е. человека, а не машины) становится все более актуальным [33].

Чтобы вникнуть в суть проблемы, необходимо учесть, что интегральная производительность систем «персонал-компьютеры» зависит от двух независимых показателей: производительности компьютеров и продуктивности собственно персонала, т.е. человеческого мозга. Первая быстро растет (за счет увеличения мощности компьютеров, совершенствования программ и передачи им все новых интеллектуальных функций). Вторая, наоборот, все более отстает от растущих требований и нередко превращается в основной тормоз повышения эффективности организаций [38].

Человеческий фактор и эргономика

Согласно современным взглядам центральным звеном информационной технологии является *человек* (а не компьютер, как считалось ранее) [9]. Тезис о приоритете человека в информационной технологии является отражением известного положения, согласно которому человек понимается как центральный фактор при создании любой техники и технологии [48].

Наука о человеческих факторах называется эргономикой [52]. На начальном этапе своего развития эргономика занималась изучением и оптимизацией деятельности операторов (летчиков, космонавтов и т.д.) в системе «человек-машина».

Однако в последнее время ситуация изменилась. В поле зрения эргономики все чаще попадает более сложный пласт проблем, связанных со стремлением повысить творческую продуктивность ученых и специалистов, занятых разработкой больших технических систем [16, 25, 32, 40, 47, 50, 53, 61].

Под *когнитивным фактором* часто понимают познавательные, мыслительные, творческие аспекты деятельности исследователей, разработчиков, программистов и других высококвалифицированных специалистов [32, с.21]. Когнитивные проблемы – важная часть эргономики. Чтобы вычленить когнитивную группу среди других эргономических вопросов иногда употребляют термины «когнитивная эргономика» и «когнитивно-эргономические проблемы» [14].

Можно ли тестировать программу в течение миллиона лет?

В основе технологии ГРАФИТ- ФЛОКС лежит *принцип эргономизации алгоритмов*, позволяющий повысить производительность труда при их разработке [32].

При проектировании ракетно-космической техники нередко приходится создавать крупномасштабные алгоритмы, содержащие десятки и сотни тысяч 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.

Тестирование программ выполняется в конце многих этапов разработки и по этой причине является, хотя и необходимой, но чрезвычайно дорогой операцией. Известно, что тестирование – один из наиболее трудоемких этапов создания программ. По некоторым оценкам, его трудоемкость составляет от 30 до 60% общей трудоемкости [20].

Неприятность в том, что – вопреки широко распространенному мнению – при тестировании принципиально невозможно осуществить стопроцентную проверку программ, т.е. полный перебор всех маршрутов при всех сочетаниях исходных данных, так как подобная операция заняла бы миллионы и миллиарды лет [6, с. 31].

Предостережение В. Липаева

Анализируя проблему тестирования, В. Липаев пишет: «Комбинаторный характер исходных данных и накопленной при обработке информации, а также множество условных переходов создает огромное количество различных маршрутов исполнения для каждого комплекса программ, которое обычно на несколько порядков больше числа команд в программе. Такое число вариантов исполнения программы не может быть проверено полностью из-за ограничений на длительность отладки и объем приемочных испытаний.

Некоторые маршруты обработки информации характеризуются малой вероятностью исполнения. Опыт отладки и эксплуатации сложных комплексов программ показывает, что при отладке невозможно проверить все варианты обработки информации, и даже после нескольких лет эксплуатации встречаются непроверенные сочетания исходных данных, при которых работающий комплекс программ дает неверные результаты. Подобные искажения результатов с точки зрения пользователя рассматриваются как сбои или отказы» [19].

О чем говорили классики?

Отмеченный недостаток тестирования (невозможность полного перебора всех вариантов) выявил и обосновал знаменитый швейцарский программист, автор языка Паскаль Никлаус Вирт, который еще в 1973 году сформулировал так называемое *основное правило*: «Экспериментальное тестирование программ может служить для доказательства наличия ошибок, но никогда не докажет их отсут-

ствия» [6, с.31].Ту же мысль высказал и другой классик программирования, голландский ученый Эдсгер Дейкстра: «тестирование программы может вполне эффективно служить для демонстрации наличия в ней ошибок, но, к сожалению, непригодно для доказательства их отсутствия» [12, с.40].

Отсюда вытекает, что наряду с тестированием необходимо применять и другие методы выявления ошибок в программах. Одним из таких методов является верификация.

Чем хороша верификация?

Как известно, наименьший ущерб приносят ошибки, которые удается обнаружить сразу, до генерации кода и исполнения программы на компьютере –- в ходе *верификации*. При верификации главная работа по поиску ошибок выполняется благодаря интеллектуальным усилиям человека, а компьютер играет хотя и важную, но вспомогательную роль. Чтобы облегчить верификацию, алгоритмы нередко переводят в графическую форму, более удобную для понимания.

В настоящее время создано около двух десятков графических языков (компьютерных чертежей), которые используются на этапе постановки задачи и проектирования программ: схемы «сущность-связь», схемы потоков данных, схемы действий, схемы декомпозиции процессов, схемы SADT и т.д. [7,13, 23, 54 – 58]. При верификации человек тщательно изучает компьютерные чертежи спецификаций и алгоритмов, представленные на бумаге или экране и выявляет многие ошибки.

Минимизация интеллектуальных усилий

Верификацию, как и любую другую деятельность, необходимо грамотно проектировать. Критерием ее эффективности служит минимизация средних интеллектуальных усилий мозга, затрачиваемых на выявление одной ошибки. С точки зрения эргономики, нейронная конструкция мозга такова, что он может эффективно проводить визуальную проверку (верификацию) 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 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 [32, с.94].

Программные чертежи нужны не только для верификации

А.Масловский как-то заметил: «В практике создания автоматизированных систем управления известно много случаев неудачной реализации и даже полного крушения довольно крупных проектов... Спросите у любого системного аналитика, в чем основная причина подобных неудач, и он наверняка ответит, что дело в отсутствии удобного языка представления проектных решений и замыслов» [24]. О каких языках идет речь?

В последнее время все больше специалистов приходят к выводу, что наибольшие удобства при анализе сложных проблем, понимании трудных вопросов, взаимопонимании между специалистами и проектировании программных комплексов обеспечивают графические (визуальные) языки.

Характеризуя эффективность использования графических языков, Джеймс Мартин и Карма Мак-Клюр указывают: 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 [57, p.1 – 3].

ЭРГОНОМИЗАЦИЯ ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

Что такое эргономичный алгоритм?

Эргономичный алгоритм – алгоритм, удовлетворяющий критерию высокой понимаемости [32, с.95]. 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 выявить при верификации, тем больше вероятность, что вновь созданный алгоритм окажется правильным, безошибочным, надежным.

Кроме того, эргономичные алгоритмы удобнее для изучения, их проще объяснить другому человеку. Сегодня разработаны методы, позволяющие значительно повысить эргономичность алгоритмов [32].

КАКИМ ТРЕБОВАНИЯМ ДОЛЖНА УДОВЛЕТВОРЯТЬ ПРОГРАММА?

На первом этапе развития программирования к программам предъявлялся сравнительно небольшой набор требований. Считалось, что программа должна решать поставленную задачу без ошибок за минимальное время и занимать минимальное количество ячеек памяти.

На втором этапе заговорили о производительности труда и экономике. Стало ясно, что «эффективность программирования играет существенную роль в национальной экономике. Даже незначительное в процентном отношении повышение производительности труда в этой сфере деятельности может привести к существенной экономии» [18]. К сожалению, как отмечают В. Липаев и А. Петров, в нашей стране «разработчики программ, как правило, не знают даже основ экономики» [21].

Производительность труда программистов: кризис жанра

Проблема повышения производительности труда в программировании исключительно сложна. Традиционные методы в значительной степени исчерпали свои потенциальные возможности. В этой области наблюдается своеобразный кризис жанра. Как сообщают ученые, «несмотря на значительные усилия и средства, затраченные на развитие индустриального программирования, резкого повышения производительности труда программистов не произошло и, по мнению

большинства специалистов, в настоящее время не ожидается кардинального изменения этой тенденции» [44]. А вице-президент фирмы «Прайм» А. Эдмондс высказывается еще более категорично: «Соотношение цены и характеристик аппаратуры улучшалось большими скачками, но увеличение производительности труда программистов почти равно нулю [10]. Интегральная оценка американских специалистов такова: «За двадцатилетний период с 1965 по 1985г. потребность в программном обеспечении выросла в сто раз, в то время как производительность труда программистов увеличилась лишь вдвое» [58, p.23].

В последнее время появился ряд новых методов: CASE-технологии (CASE означает «автоматизированное проектирование систем и программного обеспечения»), интегрированные CASE-технологии (обозначаемые как I-CASE), методология RAD (Rapid Application Development – быстрая разработка приложений), объектно-ориентированное программирование и т.д. [4, 7, 13, 54 – 59]. Эти средства позволили в ряде случаев существенно поднять производительность. Если же говорить в целом, проблема роста производительности по-прежнему остается чрезвычайно острой и актуальной.

Автоматизация и эргономизация как факторы,влияющие на производительность труда

Среди методов, повышающих производительность труда, наиболее важными являются автоматизация и эргономизация.

Под *автоматизацией программирования* понимают автоматизацию всех работ, связанных с созданием и сопровождением программного обеспечения на всех этапах жизненного цикла от анализа проблемы и постановки задачи до снятия программного комплекса с эксплуатации.

Эргономизация подразумевает прежде всего улучшение эргономичности языковых средств проектирования, моделирования, программирования и сопровождения. Эргономизация опирается на следующий постулат.

> Язык – важнейший интеллектуальный инструмент разработчиков и программистов. Чем лучше эргономические характеристики языка, тем лучше работает мозг, тем выше производительность умственного труда при создании алгоритмов и программ [32, с.16].

Необходимость эргономизации языков программирования была понята не сразу. Одним из первых осознал проблему Эдсгер Дейкстра. Он подверг критике ранние языки за то, что написанные на них программы появляются в виде, рассчитанном на механическое исполнение в компьютере, но «совершенно непригодном для человеческого восприятия» [12, с.9].

Соглашаясь с ним, Никлаус Вирт подчеркивает: представление сложной программы в виде лишенной структуры линейной последовательности команд было самой неподходящей формой для человеческого восприятия и выражения [6, с.27].

Постепенно стало ясно, что «программист мыслит категориями, которые дает ему в распоряжение язык программирования», причем «царящий в существующих языках беспорядок» непосредственно отражается на стиле и эффективности труда [43]. По мнению экспертов, влияние языка «независимо от нашего желания сказывается на нашем способе мышления» [12, с.9]. Язык оказывает глубокое воздействие «на навыки мышления и изобретательские способности» [43].

Развитие этих идей постепенно привело к выводу, что эргономизация алгоритмических языков является мощным средством улучшения качества мышления и повышения продуктивности труда разработчиков и программистов. Современные методы эргономизации алгоритмических языков описаны в работе [32].

Два противоположных требования

Уже говорилось, что неуклонный рост сложности проектируемых ракетно-космических систем влечет за собой увеличение нагрузки на мозг разработчиков, которая порою становится запредельной, что увеличивает вероятность заблуждений, просчетов и ошибок.

Налицо противоречие. В самом деле, чтобы избежать перегрузок нервной системы и связанных с ними негативных последствий, необходимо уменьшить нагрузку на мозг творческих работников. С другой стороны, лавинообразное усложнение задач, непрерывное увеличение их количества, а также сильное давление экономических требований предъявляет к человеческому мозгу постоянно растущие требования, направленные на повышение его интеллектуальной производительности. Спрашивается: можно ли одновременно выполнить два противоположных требования — облегчить работу мозга и увеличить его интеллектуальную 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 [36, с. 338].

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 [12]. Академик А.Ершов подчеркивает: программисты составляют первую большую группу людей, работа которых ведет к пределам человеческого знания... и затрагивает глубочайшие тайны человеческого мозга [53, с.11]. Психолог П. Ярошевский отмечает: успехи кибернетики, передача машинам поддающихся формализации умственных операций, которые раньше поглощали значительную часть интеллектуальных усилий разработчика, резко повышают требования к формированию его способностей производить такие действия, которые не могут совершаться компьютерами [47, с.10].

Переломная веха в развитии технологии программирования

Таким образом, чтобы преодолеть «кризис производительности» необходимо использовать два рычага. С одной стороны, надо повышать степень автоматизации, с другой – создавать более комфортные (более эргономичные) условия для работы человеческого мозга, которые позволяют существенно повысить его умственную продуктивность.

Вспомним еще раз, что сегодня узким местом являются не машинные, а человеческие ресурсы, т.е. главная задача заключается в увеличении производительности мозга (а не компьютера). Следует специально подчеркнуть, что рост продуктивности мозга можно достичь только эргономическими (а отнюдь не техническими) методами. Если допустить, что человеческий фактор является главным звеном любой информационной технологии, придется признать, что именно эргономика выходит на передний план при решении вопросов продуктивности (мозга) разработчиков и программистов.

Таким образом, происходит важная смена ориентиров, перенос внимания на когнитивно-эргономические методы. Этот факт можно рассматривать как переломную веху в развитии технологии программирования.

Можно утверждать, что эргономизация технологии (наряду с активным использованием других средств и методов) открывает новые перспективы для повышения производительности труда больших коллективов, состоящих из программистов и непрограммистов, которые совместными усилиями создают крупномасштабные программные комплексы.

Принцип эргономизации – один из ключевых принципов, положенных в основу информационной технологии ГРАФИТ- ФЛОКС.

ТЕХНОЛОГИЯ «ГРАФИТ-ФЛОКС»

Недостатки традиционного подхода

Прежде чем перейти к изложению принципов построения технологии ГРАФИТ-ФЛОКС, необходимо отметить, что традиционные языки и технологии программирования для встроенных (бортовых и наземных) ЦВМ в нашей отрасли имеют существенные недостатки.

- Слабо учитывается критерий экономической эффективности [21]. Известно, что при программировании основные усилия тратятся не на кодирование, а на понимание проблемы и сопровождение программ [8, 10]. Поэтому качественное программное обеспечение должно обладать помимо надежности и эффективности ещё и такими важнейшими свойствами, как понимаемость и сопровождаемость [49]. Однако современные языковые средства обеспечения понимаемости и сопровождаемости являются недостаточно эффективными.
- Автоматизированная технологическая цепочка создания программ является разорванной. Проектирование концепций и алгоритмов и разработка программ рассматриваются как два различных этапа, причём для автоматизации указанных этапов используются различные инструментальные комплексы, не объединённые в интегрированную систему. По этой причине переход от первого этапа ко второму требует значительных затрат ручного труда алгоритмистов и программистов.
- Отсутствует системный подход: разработка программ и формализация профессиональных знаний рассматриваются как две самостоятельные и почти не связанные между собой задачи, при решении которых используются различные языковые средства. Это приводит к дублированию работ и удорожанию проектов.
- Заказчик в большинстве случаев лишён возможности осуществлять содержательный и детальный контроль качества алгоритмов и программ. Вследствие использования неудачных языковых средств тексты программ и документация на программы настолько сложны, что трудоёмкость их содержательной проверки на порядок превышает реальные возможности заказчика. Поэтому заказчик в подавляющем большинстве случаев вынужден отказываться от содержательного контроля качества программ и ограничиваться частичным тестовым контролем по инструкциям разработчика.
- Отсутствует эргономически оптимальный единый набор языковых средств представления знаний, обеспечивающих эффективное междисциплинарное научное и научно-техническое общение между специалистами разного профиля, принимающими участие в работах по созданию программного обеспечения. Сегодня различные участники технологической цепочки создания программ (постановщики задач, математики, механики,

управленцы, прибористы, комплексники, программисты) нередко используют различные и почти несовместимые средства представления своих профессиональных знаний (т.е. говорят «на разных языках»). Это затрудняет процесс интеллектуального взаимопонимания и интеллектуального взаимодействия между ними, что увеличивает затраты на создание программных комплексов и заметно удлиняет сроки опытно-конструкторских работ.

■ Технология программирования, основанная на концепции жизненного цикла, является неполной, так как не охватывает задачу обучения персонала (как у разработчика, так и у заказчика). При выборе языковых средств для технологии программирования разработчики последней обычно не задумываются об отдаленных последствиях, связанных с необходимостью решать проблему обучения персонала. Между тем острота этой проблемы в условиях рыночной экономики и большой текучести кадров на госпредприятиях будет возрастать.

■ Обучение эксплуатационного персонала в войсках требует неоправданно больших трудозатрат. Причина в том, что разработка языковых средств представления знаний о программах не опирается на последние достижения эргономики и когнитивной науки и заметно отстаёт от роста сложности программных комплексов.

■ Недооценка познавательных (когнитивных) проблем, связанных с разработкой программ, отсутствие эффективной теоретической модели интеллектуального взаимодействия между специалистами (участниками технологической цепочки создания и эксплуатации программ) нередко приводят к недостаточному учёту человеческого фактора в технологии создания программного обеспечения. В этом состоит одна из важнейших причин того обстоятельства, что предыдущие попытки создать *Единую технологию программирования*, приемлемую для всех предприятий ракетно-космической отрасли и заказчика, оказались неудачными.

Общие сведения о технологии «ГРАФИТ-ФЛОКС»

Информационная технология ГРАФИТ-ФЛОКС подразумевает устранение отмеченных недостатков. Она предназначена для решения всех или большинства вопросов, связанных с созданием и эксплуатацией программного обеспечения для встроенных (управляющих) ЦВМ на протяжении жизненного цикла. Особое внимание уделяется следующим вопросам:

- систематизация, формализация, эргономизация и дружелюбное представление профессиональных знаний разработчиков алгоритмов и программ,
- организация чётко согласованной высокопродуктивной коллективной интеллектуальной деятельности разработчиков,
- быстрое и эффективное обучение персонала.

Технология ГРАФИТ- ФЛОКС имеет следующие особенности.

■ Используется принцип строгого разграничения процедурных и декларативных знаний. Для этих целей языковые средства данной технологии расщепляются на два относительно самостоятельных языка: процедурный язык ГРАФИТ [2, 3] и декларативный язык ФЛОКС. Каждый исходный мо-

дуль знания делится на две части: процедурную и декларативную. Все декларативные части объединяются в единую базу данных ФЛОКС.

■ Благодаря использованию новых, более лёгких и эффективных языковых средств представления знаний специалисты различного профиля (участники технологической цепочки создания программ) во многих случаях получают возможность формализовать свои профессиональные знания самостоятельно, по принципу «сделай сам» (т.е. без помощи профессиональных программистов и инженеров по знаниям). Вследствие этого значительно возрастает доля формализованных знаний в общем объёме проекта, что позволяет увеличить объём машинных и человеческих проверок.

■ Наряду с известными критериями качества программ вводится дополнительный критерий минимизации интеллектуальных усилий персонала. Согласно этому критерию при сопоставлении различных зрительно воспринимаемых форм представления знаний следует отдавать предпочтение (при прочих равных условиях) таким формам, которые позволяют увеличить скорость безошибочного восприятия, понимания и усвоения знаний человеком. Выигрыш в том, что при выполнении этого требования обеспечивается почти полная «прозрачность» содержания проектов для всех видов визуальной верификации, а скорость и качество верификации увеличиваются. Это объясняется тем, что контроль со стороны заказчика, руководителя, коллег и самоконтроль автора осуществляются ценою минимальных интеллектуальных усилий [38, с. 8, 9].

В остальном технология ГРАФИТ- ФЛОКС опирается на известные методы: системно-структурный анализ, методологию RAD, объектно-ориентированное программирование и т.д. Однако все эти методы преобразуются таким образом, чтобы удовлетворить эргономическому критерию минимизации интеллектуальных усилий и за счет этого задействовать дополнительные резервы повышения производительности труда разработчиков, эксплуатационников и заказчиков.

Что лучше: графика или текст?

С точки зрения эргономики, наиболее важной характеристикой алгоритма является его *понимаемость*, т.е. свойство алгоритма минимизировать интеллектуальные усилия, необходимые для его понимания [49]. Наиболее мощным средством для улучшения понимаемости является визуализация алгоритма, т.е. преобразование алгоритма из текстовой формы в графическую [32]. Проблемы визуализации и эргономизации знаний в последнее время активно обсуждаются в литературе [13, 23, 24, 26 – 42, 46, 62]. Бурными темпами развивается визуальное программирование [7, 13, 23, 29, 30, 32, 34, 51, 54 – 61].

Общепризнано, что человеческий мозг в основном ориентирован на визуальное восприятие и люди получают информацию при рассмотрении графических образов быстрее, чем при чтении текста [5]. Данный вывод вступает в острое противоречие с традиционными стереотипами программистского мышления, согласно которым наилучшей формой записи программ является текст.

Мировая практика показывает: при организации работы больших коллективов разработчиков, в которых конечный результат создаётся совместными усилиями программистов и непрограммистов, наивысшую производительность коллективно-

го труда обеспечивает отказ от текстовых форм и переход к графической записи алгоритмов [54 – 59].

«Безответственное руководство»

В 1989г. журнал «Форчун» попытался выяснить, почему так трудно писать программы: «Программное обеспечение – это "материя чистой мысли", бестелесная и умозрительная. Поэтому проектировщики программ 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 новых подходов думают иначе. Комментируя указанную статью, видный американский специалист Джеймс Мартин пишет: «важно понять, что эта народная мудрость сегодня уже не верна». Он подчеркивает, что в новейших технологиях программирования, например, в методологии RAD, применяются «точные и детальные чертежи и схемы (аналогичные тем, что рисуют конструкторы электронного оборудования) с помощью технологии I-CASE, причём 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» [54, p.38].

Процедурные и декларативные знания

Человеческие знания, выраженные с помощью письменного языка, можно разбить на две части: процедурные и декларативные.

Процедурные знания – сведения о последовательности действий и о выборе пути при разветвлении процессов.

Декларативные знания – сведения не о действиях, а о свойствах объектов. Иначе говоря, декларативные знания – это описания объектов, их свойств и характеристик.

Язык ДРАКОН – новое средство для повышения производительности труда

В технологии ГРАФИТ-ФЛОКС используется новое средство, обеспечивающее значительный рост производительности труда при создании сложных программных комплексов – язык ДРАКОН.

В чем его особенность? Чтобы уяснить суть дела, обратимся к анализу традиционных языков, таких как Ассемблер, Фортран, Бейсик, Паскаль, Си, Си++ и т.д. Во всех этих языках имеются средства для записи процедурных и декларативных знаний, которые *образуют одно целое*. Никто не говорит: это процедурная часть Паскаля, а это – декларативная. Поэтому в программах процедурные и декларативные части пишут подряд, вперемешку. Это значит, что перечисленные языки представляют собой «аморфную массу», в которой отсутствуют эргономически эффективные средства, позволяющие чётко разграничить процедурные и декларативные знания, провести между ними бросающуюся в глаза наглядную эргономическую границу.

Два подъязыка: ГРАФИТ и ФЛОКС

Язык ДРАКОН устроен принципиально иначе. Он «разрезан» на два относительно самостоятельных и концептуально законченных подъязыка:

- процедурный язык ГРАФИТ, предназначенный для записи процедурных знаний,
- декларативный язык ФЛОКС, служащий для записи декларативных знаний.

Каждый язык (ГРАФИТ и ФЛОКС) имеет свой синтаксис, свою семантику, свое описание и комплект документации.

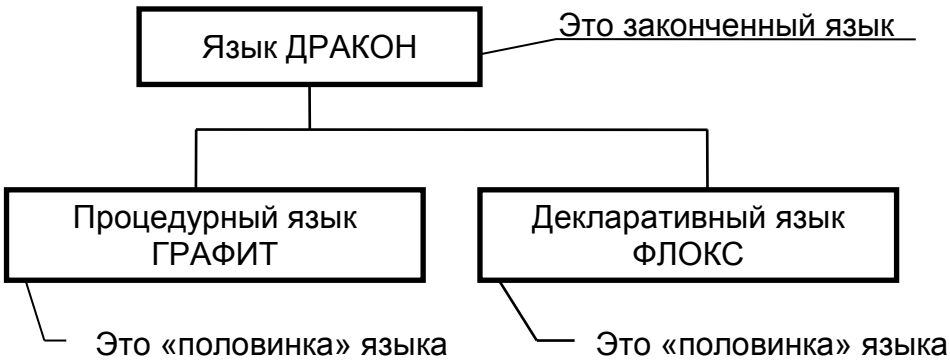
Алгоритм на языке ДРАКОН включает два документа:

- чертёж алгоритма на языке ГРАФИТ (*графит-алгоритм*),
- таблицу понятий, использованных в графит-алгоритмах, которая записана на языке ФЛОКС (*флокс-таблицу*).

Как связаны три понятия: ГРАФИТ, ФЛОКС и ДРАКОН?

Таким образом, графит-алгоритм и флокс-таблица – не законченные продукты, а полуфабрикаты. Для генерации объектного кода нужны оба компонента. Отсутствие одного из них делает создание объектного кода невозможным.

Отсюда вытекает, что ГРАФИТ и ФЛОКС – не самостоятельные языки, а «половинки» языка. Мысленно соединяя обе половинки, получаем законченный продукт – язык ДРАКОН.



Ещё одна особенность. Задав синтаксис и семантику языков ГРАФИТ и ФЛОКС, мы тем самым полностью определяем синтаксис и семантику языка ДРАКОН. Это означает, что язык ДРАКОН не имеет собственного описания. Поэтому, строго говоря, термин «ДРАКОН» является избыточным и обычно используется только в педагогических целях. В документации на технологию ГРАФИТ-ФЛОКС термин «ДРАКОН» не употребляется.

Подведём предварительные итоги. В технологии ГРАФИТ- ФЛОКС процедурные знания выражают на языке ГРАФИТ, декларативные – на языке ФЛОКС. Разделение языка ДРАКОН на две почти самостоятельные «половинки» позволяет получить значительные эргономические преимущества и за счет этого существенно повысить производительность труда.

Программироние без программистов

В 1982 г. Джеймс Мартин выдвинул идею «программирование без программистов» [53а], согласно которой алгоритмы и прикладные программы должны разрабатывать не программисты, а специалисты в предметной области, т. е люди, которые являются «первоисточниками» соответствующих знаний. При этом, по мнению Мартина, достигается двоякий выигрыш:

- из трудового процесса исключается целый класс лишних работников (прикладных программистов), которые, выполняя посреднические функции между специалистом и компьютером, представляет собой лишнее звено в цепочке «специалист – программист – компьютер»,
- устраняются ошибки типа «испорченный телефон», возникающие в процессе общения специалистов в предметной области и программистов.

В настоящее время создан ряд систем, в которых реализованы различные варианты воплощения этой идеи. К сожалению, они имеют недостатки, препятствующие ее широкому распространению. По этому поводу Пол Литвин с досадой пишет: «Много лет говорилось о том, что разработка прикладных программ должна быть доступна не только профессиональным программистам. Однако на деле ни один из созданных наборов инструментальных средств не приблизил нас к решению данной проблемы» [21а].

Информационная технология ГРАФИТ-ФЛОКС представляет собой очередную попытку реализовать метод «программирование без программистов» в сочетании с принципом эргономизации и визуальной алгоритмизации. Следует подчеркнуть, что о полном исключении прикладных программистов речь не идет, так как последние выполняют важную функцию оптимизации полученных программ с целью экономии машинных ресурсов (если в этом есть необходимость).

Уместно отметить следующие особенности технологии ГРАФИТ-ФЛОКС.

- Разработку графит-алгоритмов и флокс-таблиц могут осуществлять *разные* специалисты.
- Заполнение разных граф флокс-таблиц могут осуществлять *разные* специалисты.
- Благодаря этому достигается наиболее эффективное разделение труда. Каждый работник вводит в систему только *свои собственные знания*, т.е. сведения о тех вопросах, которые лучше его никто не знает, где он является специалистом номер 1.
- В результате система ГРАФИТ-ФЛОКС получает знания не из вторых рук, а непосредственно из первоисточников, что существенно уменьшает вероятность ошибок и повышает продуктивность совместной работы больших коллективов, состоящих из непрограммистов (специалистов-предметников) и программистов.

ЗАЧЕМ НУЖЕН ЯЗЫК «ГРАФИТ»?

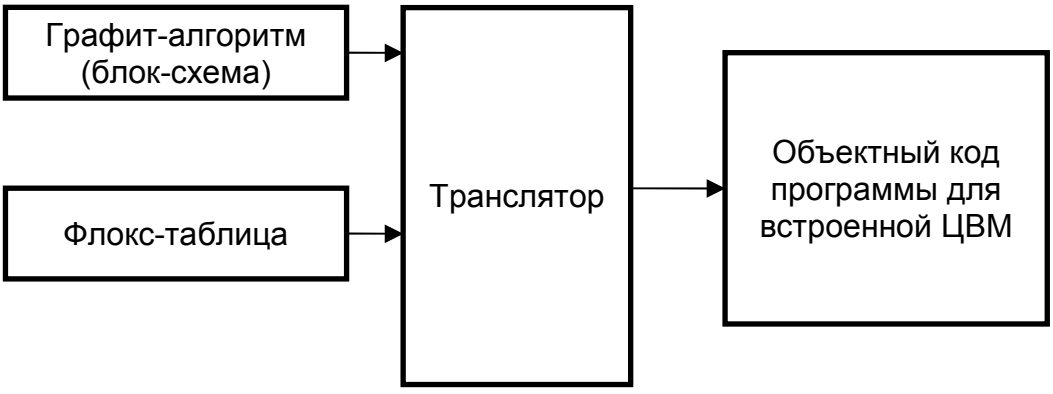
Общие сведения о языке ГРАФИТ

Строгое описание языка ГРАФИТ дано в документах [2, 3]. Наша цель – дать общее представление об этом языке.

ГРАФИТ – визуальный (графический) язык для записи алгоритмов. При проектировании языка ГРАФИТ за основу взяты блок-схемы алгоритмов и программ, описанные в ГОСТ 19.701-90 [49а]. Указанные блок-схемы были значительно модифицированы с учётом принципов формализации, структуризации и эргономизации. Таким образом, несмотря на некоторое сходство с блок-схемами, язык ГРАФИТ является оригинальной разработкой.

На языке ГРАФИТ программы не пишут, а рисуют на экране персонального компьютера с помощью специальной программы – визуального ГРАФИТ-редактора. В результате получается «картинка» (блок-схема), которая с одной стороны, удовлетворяет требованиям математической строгости, с другой – является удобной для человеческого восприятия и лёгкой для понимания (эргономичной). Это позволяет значительно улучшить наглядность и доходчивость алгоритмов и программ, устранить чрезмерную и неоправданную нагрузку на мозг разработчиков.

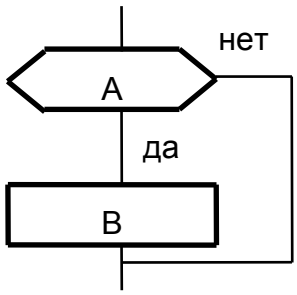
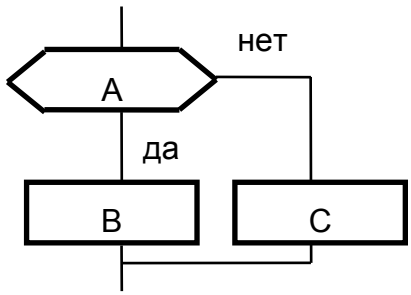
Поскольку ГРАФИТ – формальный язык, указанная «картинка» – после добавления флокс-таблиц – автоматически транслируется в объектный код встроенной (бортовой или наземной) ЦВМ.



Упрощенная схема генерации объектного кода

Как выглядит оператор языка ГРАФИТ?

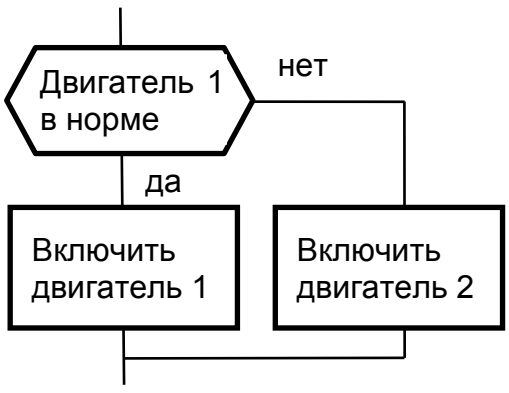
Принцип записи операторов на языке ГРАФИТ можно понять из примера

Оператор обычного языка	Оператор языка ГРАФИТ
Если А то В	
Если А то В иначе С	

Перейдём от условных обозначений к содержательным согласно таблице:

Условное обозначение	Содержательное обозначение
А	Двигатель 1 в норме
В	Включить двигатель 1
С	Включить двигатель 2

С учетом этих обозначений использованный выше пример принимает вид:

Программа на обычном языке	Программа на языке ГРАФИТ
Если (Двигатель 1 в норме = 1) **то** Включить двигатель 1; **иначе** Включить двигатель 2;	

Как выглядит упрощённый алгоритм на языке ГРАФИТ?

Рассмотрим три картинки на стр. 37, 38 и 39. На первой из них показан упрощённый алгоритм, на второй – «почти настоящий», на третьей – реальный алгоритм.

Начнём по порядку (см. стр. 37). Упрощённый алгоритм называется ОТКЛЮЧЕНИЕ.УСИЛИТЕЛЕЙ.МОЩНОСТИ. Сначала из БЦВМ по линии связи к абоненту выдаётся команда ОТКЛЮЧИТЬ.ФИДЕР.УМ1. После этого выдерживается пауза длительностью 32 секунды. Затем выдаётся вторая команда ОТКЛЮЧИТЬ.ФИДЕР.УМ2 . Далее следует ещё одна пауза, после чего снимаются (устанавливаются в состояние "0") два логических признака:

```
ВКЛЮЧЕН.ФИДЕР.УМ1 := 0
ВКЛЮЧЕН.ФИДЕР.УМ2 := 0
```

Как выглядит «почти настоящий» графит-алгоритм?

В графит-алгоритмах в начале каждого идентификатора имеется так называемый классификатор, содержащий пять символов. Добавив классификаторы к упрощённому алгоритму, получим «почти настоящий» алгоритм, изображённый на стр. 39.

Первые два символа классификатора означают:

АП – Алгоритм Процедура
КС – Команда Силовая
ПЛ – Признак Логический

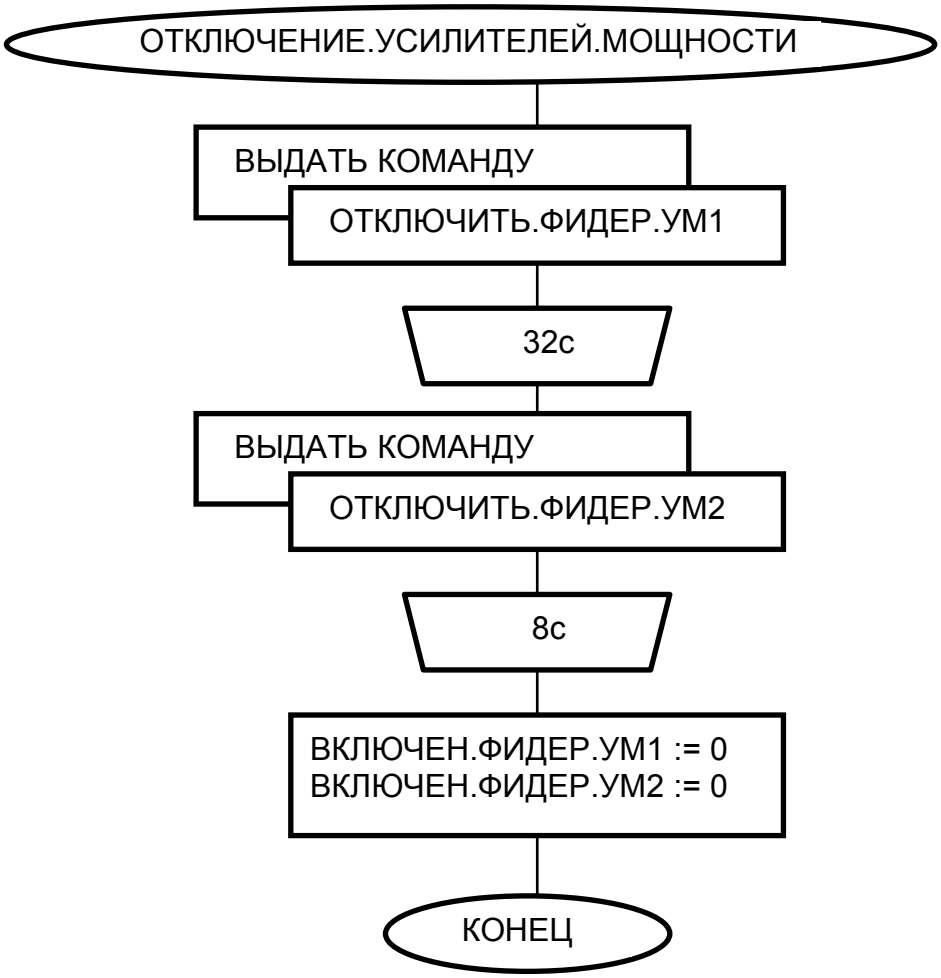
Остальные три символа классификатора следует понимать так:
1 – означает, что алгоритм выполняет бортовая (а не наземная) ЦВМ.
У – обозначает один из нескольких функциональных трактов.
Ф – означает, что алгоритм предназначен для ракетно-космического проекта ФРЕГАТ.

Как выглядит реальный графит - алгоритм?

В реальных алгоритмах длина идентификатора не должна превышать 16 символов. Чтобы превратить «почти настоящий» алгоритм в реальный, необходимо проверить длину всех идентификаторов и уменьшить её до 16 символов. После выполнения этой операции получим алгоритм, представленный на стр. 40.

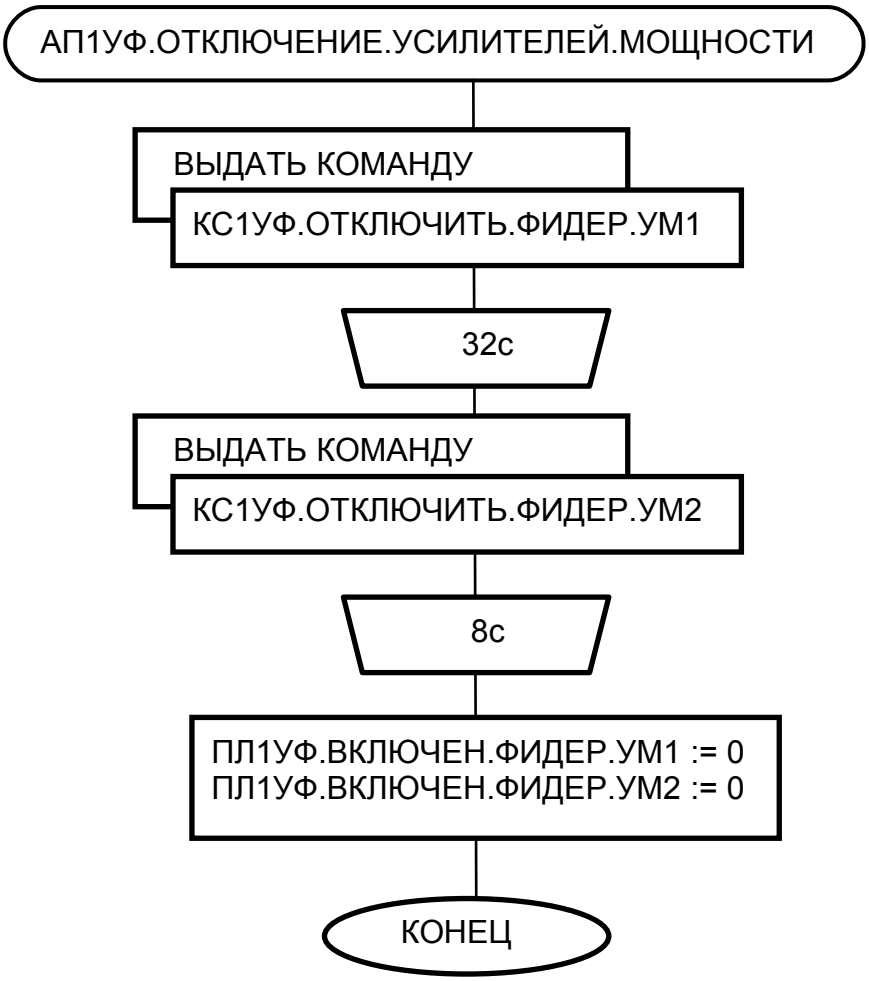
Обратите внимание: в реальном графит-алгоритме используются зарезервированные слова – ВЫДАТЬ (выдать команду), КОНЕЦ и С (секунда).

И последнее. Чтобы понять смысл реального алгоритма в терминах предметной области рекомендуется следующий приём. При чтении идентификаторов нужно пропускать первые пять символов (которые образуют классификатор) и начинать чтение после первой точки (см. стр. 40).



Упрощенный алгоритм на языке ГРАФИТ.

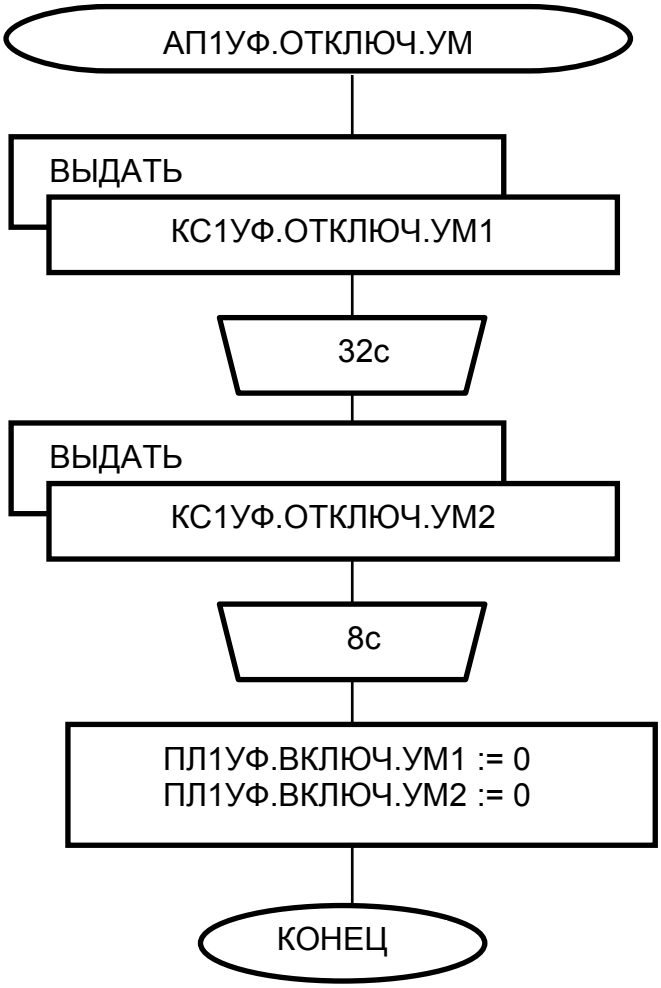
(Во всех идентификаторах опущены первые шесть символов)



«Почти настоящий» алгоритм на языке ГРАФИТ.

Примечание. Отличие от реального алгоритма состоит в следующем:

- на данном рисунке длина идентификаторов превышает 16 символов, что запрещено;
- вместо реального оператора ВЫДАТЬ сделана более подробная запись ВЫДАТЬ КОМАНДУ.



Реальный алгоритм на языке ГРАФИТ

ЗАЧЕМ НУЖЕН ЯЗЫК «ФЛОКС»?

Вторая «половинка» языка

Мы уже знаем, что ГРАФИТ – не законченный язык, а «половинка» языка. Поэтому графит-алгоритм содержит лишь половину сведений, необходимых для трансляции и генерации объектного кода. Другая половина содержится во флокс-таблице и записывается на языке ФЛОКС.

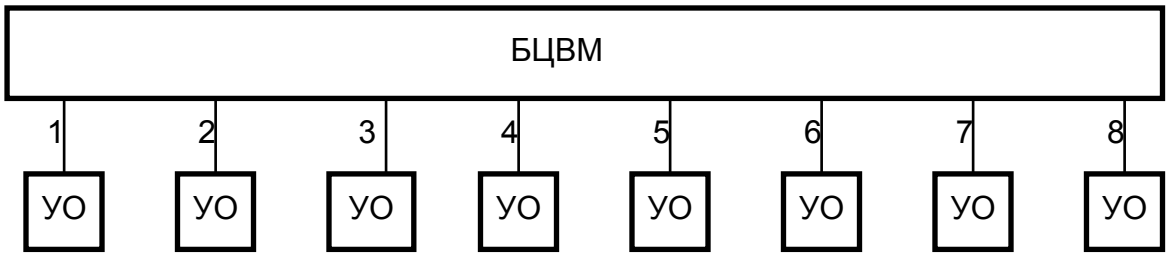
Рассмотрим вопрос подробнее. Из примера на стр. 40 видно, что идентификаторы, записанные в графит-алгоритмах, *не определены*. Изюминка в том, что определения идентификаторов специально удалены из графит-алгоритмов и помещены во флокс-таблицу.

О каких определениях идёт речь? Чтобы определить идентификатор

КС1УФ.ОТКЛЮЧИТЬ.ФИДЕР.УМ1

необходимо, в частности, ответить на четыре вопроса, которые анализируются ниже.

1. БЦВМ может посылать команды в восемь устройств обмена УО.



В какое из этих устройств обмена следует направить команду ОТКЛЮЧИТЬ.ФИДЕР.УМ1 ?

2. В каждом устройстве обмена имеется несколько 32-разрядных ячеек памяти. В какую из них следует записать передаваемую команду?

3. В какие биты выбранной 32-разрядной ячейки следует поместить передаваемую команду?

4. Каков двоичный код передаваемой команды, т.е. какие из выбранных битов следует установить в "0", а какие – в "1"?

Ответы на эти вопросы в своей совокупности представляют собой определение идентификатора. Определения записываются на языке ФЛОКС, который представляет собой вторую «половинку» языка.

Упрощённая последовательность операций

Последовательность операций при разработке алгоритмов с помощью языков ФЛОКС и ГРАФИТ представлена в таблице

Номер шага	Название операции
Шаг 1	Определить набор понятий, необходимых для разработки комплекса алгоритмов
Шаг 2	Для каждого понятия записать идентификатор
Шаг 3	Для каждого идентификатора составить флокс-описание Набор флокс-описаний поместить во флокс-таблицу
Шаг 4	Ввести флокс-таблицу в базу данных ФЛОКС
Шаг 5	Разработать графит-алгоритмы
Шаг 6	Используя графит-алгоритм и базу данных ФЛОКС, произвести трансляцию и получить объектный код программы

Порядок работы при проектировании алгоритмов по технологии ГРАФИТ-ФЛОКС

Как видно из таблицы, работа выполняется за шесть шагов (которые могут частично перекрываться).

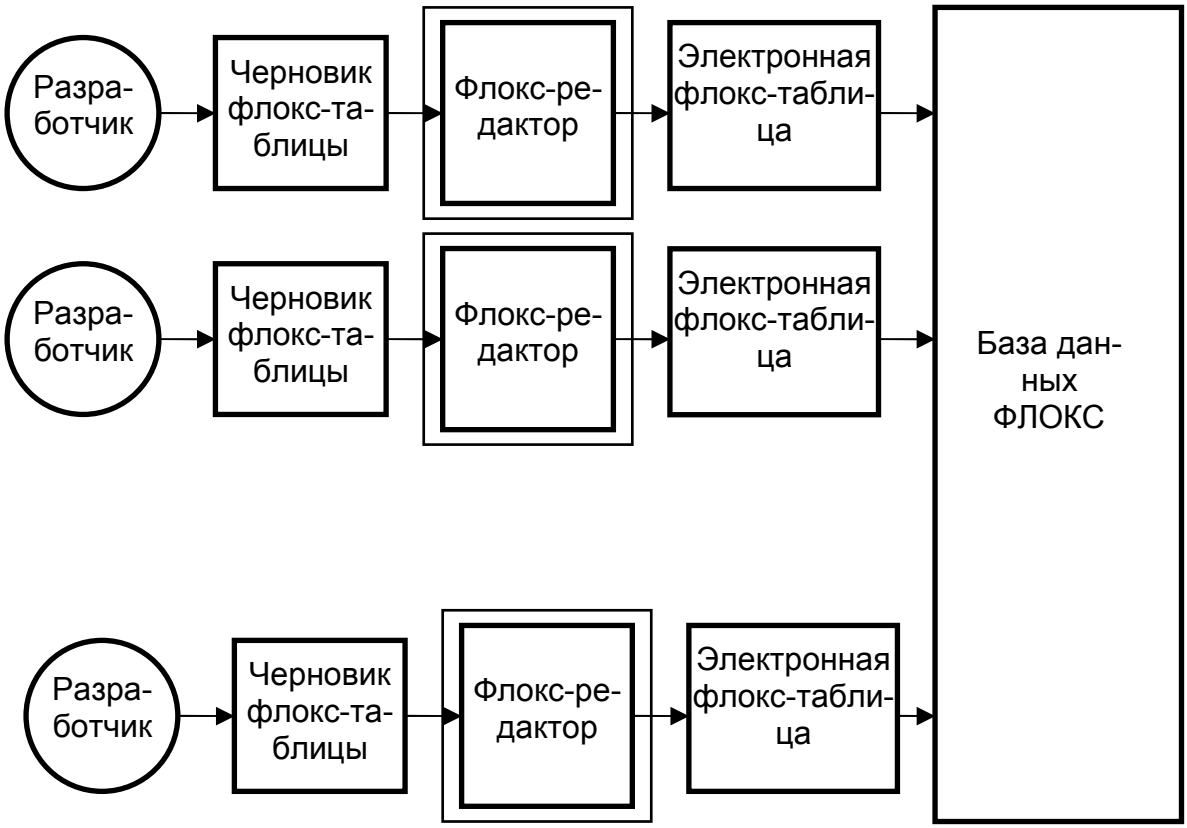
Шаг 1. Определяют набор понятий, необходимых для разработки проекта. Понятиями считаются алгоритмы, параметры, команды, сигналы и массивы. Например, при разработке доразгонного модуля проекта «Морской старт» использованы 4000 понятий.

Шаг 2. Для каждого понятия записывают формальное имя – флокс-идентификатор (далее – идентификатор).

Шаг 3. На языке ФЛОКС для каждого идентификатора записывают флокс-описание, которое содержит атрибуты, необходимые для определения идентификатора, а также другие полезные сведения об этом понятии.

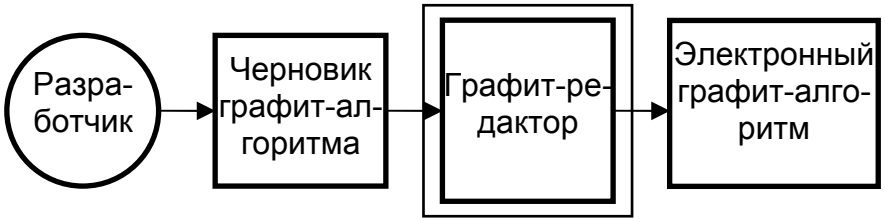
Примечание. Идентификатор – составной элемент флокс-описания. Несколько флокс-описаний образуют флокс-таблицу.

Шаг 4. Флокс-таблицы помещают в базу данных ФЛОКС. Ввод флокс-таблиц в инструментальную ЦВМ осуществляют с помощью программы «флокс-редактор».



Шаг 5. Используя язык ГРАФИТ, разрабатывают графит-алгоритмы.

В графит-алгоритмах разрешается употреблять только те идентификаторы, которые определены в базе данных ФЛОКС. Ввод графит-алгоритмов в инструментальную ЦВМ производят с помощью графит-редактора.



Шаг 6. Производится трансляция графит-алгоритма. Встретив в алгоритме идентификатор, транслятор обращается к базе данных ФЛОКС, находит нужный идентификатор, извлекает соответствующее флокс-описание, выбирает из него нужные атрибуты, необходимые для трансляции, производит трансляцию и формирует объектный код графит-алгоритма.

Если искомый идентификатор отсутствует в базе данных ФЛОКС, трансляция графит-алгоритма останавливается и выдаётся сообщение об ошибке.

Разделяй и властвуй!

А теперь рассмотрим всё сказанное с точки зрения критерия эргономизации. Вспомним, что одна из важнейших целей технологии ГРАФИТ-ФЛОКС – значительное повышение производительности труда. Эргономизация профессиональ-

ных языков – мощное средство для увеличения продуктивности (мозга) разработчиков больших алгоритмических комплексов. Каким образом эта идея реализуется на практике?

В технологии ГРАФИТ-ФЛОКС предусмотрены специальные эргономические приёмы, позволяющие облегчить труд разработчиков, упростить сложные алгоритмы, сделать их более лёгкими для понимания.

С одним из таких приёмов мы уже знакомы. Каждый алгоритм делится на две части:

- Первая часть содержит графическую схему алгоритма, из которой удалены определения идентификаторов. Эту часть записывают на языке ГРАФИТ.
- Вторая часть содержит определения идентификаторов и дополнительную информацию (комментарии). Эту часть пишут на языке ФЛОКС.

При обычном (традиционном) подходе обе части записывают вперемешку в одном и том же документе. В результате возникает ситуация «все в одной куче!», которая затрудняет чтение и понимание алгоритма.

В технологии ГРАФИТ-ФЛОКС подобные трудности исключены. Деление алгоритма на две части позволяет получить значительные эргономические преимущества. Первая часть (графит-алгоритм) содержит наиболее важную информацию, очищенную от «мелкого мусора». Она позволяет читателю взглянуть на алгоритм через «волшебное увеличительное стекло» и увидеть самую суть алгоритма: последовательность выполняемых действий и условия, при которых они выполняются (или не выполняются).

Благодаря исключению второстепенных деталей разработчик попадает в идеальные условия. Он уже не путается в мелочах, а концентрирует внимание на главных, основополагающих (для данного алгоритма) вопросах. Благодаря этому существенно облегчается верификация, резко увеличивается вероятность выявления ошибок при визуальной проверке алгоритма «за столом» и за экраном видеотерминала.

Большая урна для мелкого мусора

«Мелкий мусор» – это второстепенные подробности, которые записываются в определении идентификаторов. Они обязательно присутствуют в каждом алгоритме. С эргономической точки зрения, такие подробности представляют собой «визуальные помехи», которые загромождают чертёж программы. Они мешают читать алгоритм, ибо отвлекают внимание читателя в тот самый момент, когда он хочет сосредоточиться на «стратегических» проблемах.

С другой стороны, без «мелкого мусора» обойтись нельзя. Он содержит множество числовых данных и иных деталей, без которых математическая модель алгоритма оказывается не полной, ущербной, что исключает возможность трансляции и получения объектного кода.

Мы уже знаем, что «мелкий мусор», удалённый из графит-алгоритмов, помещают в специальное хранилище – базу данных ФЛОКС. Образно говоря, база данных ФЛОКС – это большая урна для мелкого мусора.

Практика показывает, что деление алгоритма на две части:

- процедурную (записываемую на языке ГРАФИТ) и
- декларативную (записываемую на языке ФЛОКС),

представляет собой чрезвычайно плодотворную идею. Она заметно облегчает и упрощает работу с алгоритмами, ощутимо повышает производительность труда при их разработке. Во многих случаях она позволяет производить изменения в алгоритмах, не затрагивая базу данных, и наоборот, модифицировать записи в базе данных, не изменяя алгоритмы.

Проблема-невидимка

У медали есть и другая сторона. Анализ показывает, что «мелкий мусор» представляет собой проблему исключительной важности, которая зачастую выпадает из поля зрения разработчиков. Недооценка этой проблемы является источником многих коварных ошибок. Трудность в том, что «мелкий мусор» образует большие массивы информации, жизненно важные для чёткого функционирования программных комплексов. Проверка правильности этой информации требует от разработчиков огромных трудозатрат, намного превышающих их реальные возможности. Таким образом, проблема «мелкого мусора» – это проблема-невидимка, которая наносит удар из-за угла и доставляет разработчикам множество затруднений, ощутимо снижая производительность труда. Язык ФЛОКС создан для того, чтобы уменьшить или устранить эти трудности.

В общем виде проблему «мелкого мусора» можно сформулировать так. Каким образом следует описать декларативные знания на декларативном языке ФЛОКС, чтобы минимизировать количество ошибок и обеспечить максимальную производительность труда при коллективной разработке больших программных комплексов?

Что такое первичные документы?

Первичные документы – документы, в которых разработчики, комплексники и другие соисполнители проектов описывают исходные данные для работ, производимых смежными подразделениями и программистами на последующих этапах разработки.

Существующая организация работ с первичными документами имеет ряд недостатков.

- Отсутствует стандартизация документов, что затрудняет унификацию алгоритмов обработки первичных документов и препятствует построению единой базы данных для их хранения.
- Для обозначения одних и тех же объектов в разных первичных документах нередко используются разные имена, что порождает путаницу и обилие ненужных синонимов. Это мешает взаимопониманию между участниками разработки и влечёт за собой неоправданные затраты ручного труда и машинного времени на перевод с одного «языка» на другой.
- Некоторые первичные документы не формализованы и потому не пригодны для *непосредственного* ввода данных в компьютер. Они подразуме-

вают предварительную ручную обработку, что снижает результирующую производительность труда.

- Существующие формы представления информации в первичных документах и текстах программ слишком сложны для понимания.
- Низкая понимаемость указанных документов уменьшает вероятность обнаружения ошибок при верификации, что снижает надёжность и качество разработки, увеличивает сроки отработки программотехнических комплексов.
- В процессе естественной смены кадрового состава новички испытывают серьёзные трудности, так как «в чужой программе и чужой документации очень трудно разобраться». Такие же проблемы возникают при необходимости подменить разработчика на время отпуска, болезни, командировки, увольнения и т.д.

Язык ФЛОКС специально создан для формализации и эргономизации первичных документов. Он позволяет нейтрализовать или устранить недостатки, описанные выше. Он сводит многообразие первичных документов к фиксированному набору стандартных форматов, которые имеют общее название – *флокс-описание*.

Что такое флокс-описание?

Флокс-описание служит для описания понятий. Если в проекте используется, скажем, 4000 понятий, значит необходимо создать 4000 флокс-описаний и поместить их в базу данных ФЛОКС.

Каждое флокс-описание содержит:

- идентификатор,
- паспорт,
- полное название.

Идентификатор – формальное имя понятия. Чтобы наполнить имя реальным смыслом, чтобы «привязать» его к ракетно-космическому проекту, нужно его *определить*. Определением идентификатора служит паспорт. *Паспорт* – совокупность буквенно-цифровой и кодовой информации о данном понятии, его паспортные данные.

Вопрос. Чем отличается паспорт от обычного определения?

Ответ. Обычное определение понятия содержит очень краткие, минимально необходимые сведения об этом понятии. В отличие от него паспорт флокс-описания содержит не только минимально необходимые, но и многие другие сведения, которые могут понадобиться как при машинной обработке, так и при человеческом восприятии и понимании.

Обоснование структуры флокс-описания

Трёхэлементная структура флокс-описания – важная особенность языка ФЛОКС. Выбор именно такой структуры опирается на достижения современной логики и психологии познавательных процессов.

С точки зрения логической *теории имён*, все элементы флокс-описания (идентификатор, паспорт, полное название) являются собственными именами, определяющими одно и то же понятие, т.е. *равнозначными по значению.*

С точки зрения логической *теории определений*, ситуация иная: идентификатор и полное название – это *определяемое* (дефиниендум), а паспорт – *определение* (дефиниенс).

С точки зрения *психологии познавательных процессов*, три элемента флокс-описания характеризуют три этапа познавательного процесса, отражающего движение мысли вглубь изучаемого объекта (понятия).

Сначала разработчик создаёт *полное название*, которое записывается на естественном языке и символизирует начальный этап постижения объекта.

На втором этапе происходит формализация понятия, которая заключается в замене неформализованного полного названия на формальный *идентификатор*. На третьем (последнем) шаге разработчик *определяет* идентификатор, т.е. заполняет паспорт понятия. На этом этапе указываются «мельчайшие подробности» и количественные характеристики информационного объекта.

Таким образом, все три элемента флокс-описания являются необходимыми и полезными. Все они должны быть введены в базу данных ФЛОКС и зафиксированы в документации. Отсутствие хотя бы одного из них вносит неоправданные затруднения в деятельность разработчиков, снижает их интеллектуальную продуктивность.

ЕЩЁ РАЗ О ПРОИЗВОДИТЕЛЬНОСТИ ТРУДА

Повышение эффективности коллективного труда

Разработка программного обеспечения сложных ракетно-космических проектов имеет коллективный характер. При коллективной работе важную роль играет интеллектуальное взаимопонимание и интеллектуальное взаимодействие между специалистами. К сожалению, в течение длительного времени связь между взаимопониманием и производительностью труда считалась не очень существенной и недооценивалась. Однако в последнее время ситуация изменилась. Становится все более очевидным, что эта связь имеет чрезвычайно важный, более того, фундаментальный характер [32].

В связи с этим можно сделать следующие замечания.

- Процесс достижения интеллектуального взаимопонимания между специалистами, соисполнителями сложных проектов – один из наиболее сложных видов умственного труда.
- Производительность этого труда недопустимо мала и разительно отстаёт от растущих потребностей.
- Чтобы переломить неблагоприятные тенденции, необходимо резко поднять уровень интеллектуального взаимопонимания и эффективность интеллектуального взаимодействия между специалистами.
- Для решения проблемы взаимопонимания нужны новые языковые и инструментальные средства.

Технология ГРАФИТ - ФЛОКС специально ориентирована на решение этой задачи.

«Синхронизация мышления» всех участников разработки

Многие языки программирования обеспечивают *стандартизацию имён* (например, за счёт введения единообразных синтаксических правил для записи меток). Однако традиционный подход к стандартизации имён имеет ряд недостатков.

- Стандартизация имён обычно распространяется на один язык. Если при проектировании используется несколько языков, имена одного из них могут оказаться «неподходящими» для другого.
- Используемые имена далеко не всегда являются уникальными, что создаёт препятствия для машинного контроля.
- Традиционная стандартизация имён является неоправданно узкой: она ограничивается рамками программирования и не преследует цель унифицировать и стандартизировать имена, используемые непрограммистами и конечными пользователями.

Чтобы устранить недостаток, необходимо, в частности, «засинхронизировать мышление» всех участников разработки с тем, чтобы их мысленные образы, отображающие те или иные понятия, не противоречили друг другу (т.е. чтобы они одинаково понимали обсуждаемые вопросы).

Технология ГРАФИТ - ФЛОКС решает проблему следующим образом. На самом раннем этапе разработки создаётся единая система понятий, снабжённых идентификаторами и флокс-описаниями. Крайне важно, что все имена и почти все атрибуты флокс-описаний создаются не программистами, а теми специалистами, кто знает «физику процесса», т.е. обладает профессиональными знаниями в данной предметной области (назовём их *экспертами*). Именно эксперты (комплексники, разработчики, алгоритмисты) вводят стандартные имена (идентификаторы и флокс-описания) в базу данных. Благодаря этому всем участникам разработки, отработки и испытаний *принудительно навязывается* единая терминология – одинаковые комплексы имён и кодов. Тем самым осуществляется принудительная «синхронизация мышления» участников разработки.

Чтобы подобная операция имела успех, нужно выполнить два условия:

- имена и флокс-описания должны быть эргономичными (т.е. дружелюбными, удобными, доходчивыми, наглядными). В противном случае человек не станет ими пользоваться и постарается придумать другие, более удобные для себя имена, что приведёт к разнобою терминологии и к ошибкам,
- система имён и флокс-описаний должна быть полной, удовлетворяющей все мыслительные потребности разработчиков (иначе разработчик постарается заполнить пробел своим собственным «доморощенным» именем, что также приведёт к разнобою терминологии).

Единый банк эргономичных понятий – залог успеха

Правила создания флокс-описаний тщательно продуманы и обеспечивают «дружелюбный» (эргономичный) характер последних. Это создаёт необходимый интеллектуальный комфорт для мыслительной деятельности разработчиков и большие удобства в работе.

Благодаря эргономизации описанная выше *единая система понятий и единая терминология* принимается всеми участниками разработки. В итоге достигается единообразное использование терминов и кодов на всех этапах разработки, отработки и испытаний, включая расшифровку телеметрической информации.

КАКИМИ УЧЕБНЫМИ ПОСОБИЯМИ МОЖНО ПОЛЬЗОВАТЬСЯ ПРИ ИЗУЧЕНИИ ТЕХНОЛОГИИ «ГРАФИТ-ФЛОКС»?

Информационная технология ГРАФИТ-ФЛОКС тесно связана с теорией эргономичных алгоритмов и теоретической концепцией языка ДРАКОН, которые подробно обоснованы в работе [32]. Описанная выше система ГРАФИТ-ФЛОКС воплощает один из возможных подходов к построению языка ДРАКОН. Существуют и другие подходы. Вообще говоря, ДРАКОН – это не один язык, а семейство универсальных языков, предназначенных для создания наглядных эргономичных алгоритмов в любой предметной области, для формализации процедурных (технологических) знаний специалистов любого профиля, для описания структуры человеческой деятельности, для системы образования и многих других задач.

Изучение языка ДРАКОН включено в программу бакалавриата высшей школы.

Министерство общего и профессионального образования Российской Федерации утвердило «Примерную программу дисциплины "информатика" » [46], предназначенную для специальностей:

510000 – Естественные науки и математика
540000 – Образование
550000 – Технические науки
560000 – Сельскохозяйственные науки

Концепция языка ДРАКОН и теория эргономичных визуальных алгоритмов положены в основу двух разделов этой программы:

- Раздел 3. Алгоритмы и алгоритмизация. Визуализация алгоритмов.
- Раздел 4. Автоформализация профессиональных знаний.

В официальном документе Министерства общего и профессионального образования, в частности, говорится:

- Язык ДРАКОН – «один из самых легких языков представления знаний и самый первый язык, с которого нужно начинать обучение алгоритмическому мышлению и программированию» [46, с.16];
- «Благодаря своей человечности (эргономичности) язык ДРАКОН относительно легко устраняет барьеры взаимного непонимания ... между работниками различных специальностей» [46, с.16];
- «ДРАКОН создает универсальную языковую основу для процедурного интеллектуального взаимодействия между людьми, в частности, между участниками многопрофильных проектов» [46, с.16];
- «Бакалавр любой специальности должен уметь формализовать свои процедурные профессиональные знания самостоятельно, т.е. без помощи профессиональных программистов или когнитологов (инженеров по знаниям). Программа предусматривает приобретение навыков автоформализации знаний на языке ДРАКОН» [46, с.16].

В соответствии с программой [46] написана монография [32], которую рекомендуется использовать как учебное пособие для ВУЗов при изучении указанных разделов.

Обсуждается вопрос о необходимости изучения языка ДРАКОН не только в высшей, но и в средней школе [27, 34, 40]. Создан пробный учебник информатики для 3 – 6 классов средней школы на основе языка ДРАКОН [30]. Учебник оформлен как научно-популярное издание и может быть полезен не только детям, но и взрослым, желающим получить представление о визуальных (образных) методах представления знаний.

Книги [32, 30] можно рекомендовать в качестве учебных пособий для специалистов, приступающих к изучению информационной технологии ГРАФИТ- ФЛОКС.

Литература к вводному разделу

1. **Балтрушайтис В.В**. «ГРАФИТ- ФЛОКС» – технология разработки программного обеспечения бортовых вычислительных машин. – В кн.: Системы и комплексы автоматического управления в космонавтике и народном хозяйстве. Междунар. научно-техн. конф., посвящённая 90-летию со дня рождения акад. Н.А. Пилюгина. Тезисы докл. – М.: НПЦ АП, 1998. – С.79 – 81.

2. **Балтрушайтис В.В., Косточкин Г.Н.** Система проектирования алгоритмов системы управления. Описание языка управления ГРАФИТ-Ш. – М.: НПЦ АП, 1997. (Документация НПЦ АП).

3. **Балтрушайтис В.В., Косточкин Г.Н.** Система проектирования алгоритмов системы управления. Описание языка испытаний ГРАФИТ-И. – М.: НПЦ АП, 1997. (Документация НПЦ АП).

4. **Буч Г.** Объектно-ориентированное проектирование с примерами применения. – Киев.: Диалектика, Москва: ИВК, 1992.

5. **Вельбицкий И.В., Ковалёв А.Л., Лизенко С.Л.** Графический интерфейс представления алгоритмов и программ // Управляющие системы и машины, 1998, N4. – С.42.

6. **Вирт Н.** Систематическое программирование. Введение. – М.: Мир, 1977.

7. **Гейн К., Сарсон Т.** Системный структурный анализ: средства и методы. – М.: Эйтекс, 1992.

8. **Гласс Р., Нуазо Р.** Сопровождение программного обеспечения. – М.: Мир, 1983.

9. **Гончаров С.С., Ершов Ю.Л., Свириденко Д.И.** Методологические аспекты семантического программирования. – В кн.: Научное знание: логика, понятия, структура. – Новосибирск: Наука, 1987. – С.154.

10. **Громов Г.Р.** Национальные информационные ресурсы. Проблемы промышленной эксплуатации. – М.: Наука, 1985. – С.181, 184.

11. **Громов Г.Р.** Очерки информационной технологии. – М.: Инфоарт, 1993. – С.144.

12. **Дейкстра Э.** Дисциплина программирования. – М.: Мир, 1978.

13. **Калянов Г.Н.** CASE. Структурный системный анализ (автоматизация и применение). – М.: Лори, 1996.

14. **Каптелинин В.Н.** Психологические проблемы разработки пользовательских интерфейсов // Психологический журнал. – Т.13, N5, 1992. – С.38.

15. **Кауфман В.Ш.** Предисловие редактора перевода. – В кн.: Дж. Хьюз, Дж. Мичтом. Структурный подход к программированию. – М.: Мир, 1980. – С.5.

16. **Кертис Б., Солоуэй Э.М., Брукс Р.Е. и др.** Психология программных систем. О необходимости междисциплинарной комплексной программы исследований. – ТИИЭР. – Т.68, N8, 1986.

17. **Лекции лауреатов премии Тьюринга** за первые двадцать лет. 1966 – 1985. – М.: Мир, 1993.

18. **Леман М.М.** Программы, жизненные циклы и эволюция программного обеспечения // ТИИЭР. – т.68, 1980, N9. – С.26.

19. **Липаев В.В.** Надёжность программного обеспечения АСУ. – М.: Энергоиздат, 1981. – С.18.

20. **Липаев В.В.** Предисловие к русскому изданию. – В кн. Г. Майерс. Искусство тестирования программ. – М.: Финансы и статистика, 1982. – С.7.

21. **Липаев В.В., Потапов А.И.** Оценка затрат на разработку программных средств. – М.: Финансы и статистика, 1988. – С.3.

21а. **Литвин П.** OBJECT VISION – «разумные» формы документов // Мир ПК, 1991, N4. – С.42.

22. **Мануэль Т.** Объектно-ориентированное программирование: рождение новой звезды. – Электроника, 1989, N9. – С.49.

23. **Марка Д., Мак Гоуэн Д.** Методология структурного анализа и проектирования SADT. – М.: Метатехнология, 1993.

24. **Масловский Е.К.** Предисловие редактора перевода. – В кн.: Р. Бар. Язык Ада в проектировании систем. – М.: Мир, 1988. – С.5.

25. **Моляко В.А.** Психология конструкторской деятельности. – М.: Машиностроение, 1983.

26. **Ойхман Е.Г., Попов Э.В.** Реинжиниринг бизнеса: реинжиниринг организаций и информационные технологии. – М.: Финансы и статистика, 1997.

27. **Паронджанов В.Д.** Визуализация школьного курса информатики с помощью языка ДРАКОН // Педагогическая информатика, 1994, N3. – С.7–12.

28. **Паронджанов В.Д.** Возможна ли новая революция в образовании? // Высшее образование в России, 1997, N2. – С.9 – 18.

29. **Паронджанов В.Д.** Графический синтаксис языка ДРАКОН // Программирование, 1995, N3. – С.45 – 62.

30. **Паронджанов В.Д.** Занимательная информатика. Волшебный Дракон в гостях у Мурзика. – М.: Росмэн, 1998. – 152с. Илл.: 192.

31. **Паронджанов В.Д.** Знаковая революция как движущая сила НТР. – В кн.: Теоретические вопросы истории техники и научно-технического прогресса. – М.: Наука, 1994. – С.270 – 292.

32. **Паронджанов В.Д.** Как улучшить работу ума. (Новые средства для образного представления знаний, развития интеллекта и взаимопонимания). – М.: Радио и связь, 1998. – 352с. Илл.: 154.

33. **Паронджанов В.Д.** Какая фундаментальная идея победит в XXI веке? // Технология программирования девяностых годов. Междунар. конф.-ярмарка. Киев, 14 – 17 мая 1991. Тез. докладов. – Киев.: Инст. кибернетики АН УССР, 1991. – С.37 – 39.

34. **Паронджанов В.Д.** Каким будет школьный алгоритмический язык XXI века? // Информатика и образование, 1994, N3. – С.77 – 92

35. **Паронджанов В.Д.** Кризис цивилизации и нерешённые проблемы информатизации // Научно-техническая информация. Серия 2, 1993, N12. – С.1 – 9.

36. **Паронджанов В.Д.** Неожиданные уроки космонавтики XX века. Новая роль человеческого фактора и когнитивная революция в информационных технологиях. – В кн.: Человек – земля – космос. Труды первой международной авиакосмической конференции. Москва, 28 сентября – 2 октября 1992г. Том 2. Крылатые космические системы. – М.: Российская инженерная академия, 1995. – С.337 – 345.

37. **Паронджанов В.Д.** Общедоступный визуальный язык «Дракон» для улучшения понимания школьниками технологических процессов на уроках технологии. – В кн.: Проблемы, перспективы, опыт апробации и внедрения программы «Технология». Тезисы докладов II Международной конф., 9 – 12 октября 1995г. – М.: ИНТО, МИПКРО, 1995. – С.60, 61.

38. **Паронджанов В.Д.** Перспективы информационных технологий и повышение продуктивности интеллектуального труда // Научно-техническая информация. Серия 1, 1993, N5. – С.7 – 10.

39. **Паронджанов В.Д.** Развитие системного обеспечения вычислительных комплексов при отработке систем управления ракет-носителей и космических аппаратов. – В кн.: XXII научные чтения по космонавтике, посвящённые памяти академика С.П. Королёва и других выдающихся отечественных учёных – пионеров освоения космического пространства. Тезисы докл. – М.: ИИЕТ РАН, 1998. – С.14, 15.

40. **Паронджанов В.Д.** Техноязык «Дракон» – новое средство для улучшения интеллектуальной деятельности и проектирования космической техники. – В кн.: Системы и комплексы автоматического управления в космонавтике и народном хозяйстве. Международная научно-техн. конф., посвящённая 90-летию со дня рождения акад. Н.А. Пилюгина. Тезисы докл. – М.: НПЦ АП, 1998. – С.5, 6.

41. **Parondzhanov V.D.** Visualization of the Students' Intellect and the Theory of Intensive Distance Education. – In: Distance Learning and New Technologies in Education. Proceedings of the First Intern. Conf. on Distance Education in Russia. ICDED'94. 5 – 8 July, 1994. – Moscow: Association for Intern. Education, 1994. – P.415, 416.

42. **Parondzhanov V.D.** Visualization of Educational Literature Leads to Increasing of the Students' Brain Productivity. – В кн.: XXXI научная конференция факультета физико-математических и естественных наук, посвящённая 35-летию Российского университета дружбы народов. Часть 1. Математические секции. Тезисы докл. – М.: РУДН, 1995. – С.15.

43. **Перминов О.Н.** Программирование на языке Паскаль. – М.: Радио и связь, 1988. – С.4,5.

44. **Полищук В.Б.** Интеграция данных и программ в средах разработки программного обеспечения и решения задач на основе парадигмы объектно-ориентированного программирования. – В кн.: Технология программирования девяностых годов. Международная конференция-ярмарка. Киев, 14 – 17 мая, 1991. Тез. докладов. – Киев: Инст. кибернетики, 1991. – С.39.

45. **Поппель Г., Голдстайн Б.** Информационная технология – миллионные прибыли. – М.: Экономика, 1990. – С.37.

46. **Примерная программа дисциплины «Информатика».** Издание официальное. – М.: Госкомвуз, 1996.

47. **Психологические проблемы** автоматизации научно-исследовательских работ. – М.: Наука, 1987.

48. **Романов Г.М., Туркина Н.В., Колпащиков Л.С.** Человек и дисплей. – Л.: Машиностроение, 1986. – С.60.

49. **Саркисян А.А.** Повышение качества программ на основе автоматизированных методов. – М.: Радио и связь, 1991. – С.17 – 19.

49а. **Схемы алгоритмов, программ, данных и систем.** Условные обозначения и правила выполнения. ГОСТ 19.701-90. Единая система программной документации. – М.: Госстандарт СССР, 1991.

50. **Тихомиров О.К.** Психология компьютеризации. – Киев, 1988.

51. **Хлебцевич Г.Е., Цыганкова С.В.** Визуальный стиль программирования: понятия и возможности // Программирование, 1990, N4.

52. **Человеческий фактор.** В 6-и томах. – М.: Мир, 1991, 1992.

53. **Шнейдерман Б.** Психология программирования. Человеческие факторы в вычислительных и информационных системах. – М.: Радио и связь, 1984.

53a. **Martin J.** Application Development without Programmers. – Prentice Hall Inc., Englewood Cliffs, N.J., 1982.

54. **Martin J.** Rapid Application Development. – N.-Y.: Macmillan Publishing Co., 1991.

55. **Martin J.** Recommended Diagrammed Standards for Analysts and Programmers. – N.-Y.: Prentice Hall, Inc., 1985.

56. **Martin J., McClure C.** Action Diagramms: Clearly Structured Specifications, Programs and Procedures. Second Edition. – N.-Y.: Prentice Hall, 1989.

57. **Martin J., McClure C.** Diagramming Technique for Analysts and Programmers. – N.-Y.: Prentice Hall Inc., 1985.

58. **Modern Software Engineering.** Foundation and Current Perspectives. Edited by A. Ng. Peter, T. Yeh. Raymond. – N.-Y.: Van Nostrand Reinhold, 1990.

59. **Robinson J.R.** Radical Systems Development. An Introduction to Rapid Applications Development. – Los Angeles.: Opamp Technical Books, 1995.

60. **Schu N.C.** Visual Programming. – N.-Y.: Van Nostrand Reinhold Comp., 1988.

61. **Shneiderman B.** Designing the User Interface: Strategies for Effective Human - Computer Interaction. – Reading, VA: Addison-Wesley, 1987.

62. **Visualization.** Using Computer Graphics to Explore Data and Present Information / J.R. Brown, R. Earnshow, M. Jern, J. Vince. – John Wiley & Sons, Inc., 1995.