Научные труды
3 Центрального научно-исследовательского института
Министерства обороны РФ

Материалы межведомственной конференции «Современные автоматизированные системы управления реального времени как прикладное развитие научных достижений кибернетики»

(К 100-летию со дня рождения И.А. Полетаева)

Паронджанов В.Д. Визуальный алгоритмический язык ДРАКОН в ракетной технике и медицине // Современные автоматизированные системы управления реального времени как прикладное развитие научных достижений кибернетики» (К 100-летию со дня рождения И.А. Полетаева). Материалы межведомственной конференции 24 марта 2016 г. — ФГБУ «3 ЦНИИ» Минобороны РФ, 2016. — 218 с. — С. 57-78.

Рассматривается применение визуального алгоритмического языка ДРАКОН в ракетной технике (в качестве метода «Программирование без программистов») и в медицине (для предотвращения врачебных ошибок и обеспечения безопасности пациентов).

В.Д. ПАРОНДЖАНОВ

ВИЗУАЛЬНЫЙ АЛГОРИТМИЧЕСКИЙ ЯЗЫК ДРАКОН В РАКЕТНОЙ ТЕХНИКЕ И МЕДИЦИНЕ

В Научно-производственном центре автоматики и приборостроения им. акад. Н.А. Пилюгина создана инновационная визуальная технология программирования, центральным элементом которой является язык ДРАКОН.

ДРАКОН — визуальный алгоритмический язык программирования и моделирования. Был разработан при создании системы управления орбитального корабля «Буран» многоразовой транспортной космической системы «Энергия — Буран». Разработка языка велась с 1986 года при участии НПЦАП и Института прикладной математики РАН им. М. В. Келдыша. Язык построен путём формализации, эргономизации и неклассической структуризации блок-схем алгоритмов и программ, описанных в ГОСТ 19.701-90 и ISO 5807-85, а также для разработки программ реального времени [1-4].

Основной задачей разработчиков было создание единого языка, который своей доступностью и мощностью способен заменить специализированные языки: ПРОЛ2 (для разработки бортовых комплексных программ Бурана), ДИПОЛЬ (для создания наземных программ Бурана) и ЛАКС (для моделирования).

Работы по созданию ДРАКОНа были в основном закончены в 1996 году (спустя 3 года после закрытия программы «Буран»), когда была разработана автоматизированная система проектирования программных средств ГРАФИТ-ФЛОКС (рис. 1).

ДРАКОН можно определить как общедоступный визуальный язык, предназначенный:

— для систематизации, структуризации, наглядного представления и формализации процедурных (императивных) знаний;

— для описания структуры человеческой деятельности, потоков работ (workflows) и бизнес-процессов;

— для проектирования, программирования, моделирования и обучения [2, с. 31].

Правила языка ДРАКОН по созданию дракон-схем (drakon-charts) оптимизированы для восприятия и понимания алгоритмов человеком. Язык предлагается в качестве инструмента усиления человеческого интеллекта [2-4].

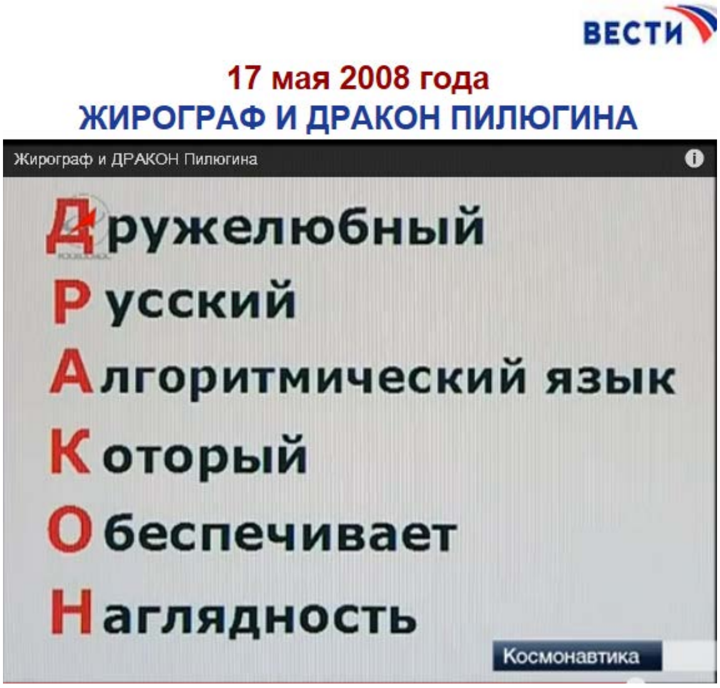
СТРУКТУРНАЯ СХЕМА ДРАКОН-ТЕХНОЛОГИИ

На рис. 2 показана структура ДРАКОН-технологии. Алгоритмы изображаются графически, в виде блок-схем языка ДРАКОН (дракон-схем). Сложные структуры данных невозможно изобразить на блок-схемах. Поэтому данные выносятся за пределы блок-схем и описываются вне блок-схем в табличной форме, в виде так называемых флокс-таблиц. Флокс-таблицы помещаются в реляционную базу данных ФЛОКС. Исходный код любой ДРАКОН-программы содержит две части:

— блок-схему (алгоритм),

— флокс-таблицу (данные).

На рис. 3 и 4 представлена функциональная схема ДРАКОН-технологии. Рис. 3 показывает последовательность работ, выполняемых инженерами по созданию исходного кода ДРАКОН-программ.



17 мая 2008 года

ЖИРОГРАФ И ДРАКОН ПИЛЮГИНА

Жирограф и ДРАКОН Пилюгина

Дружелюбный

Русский

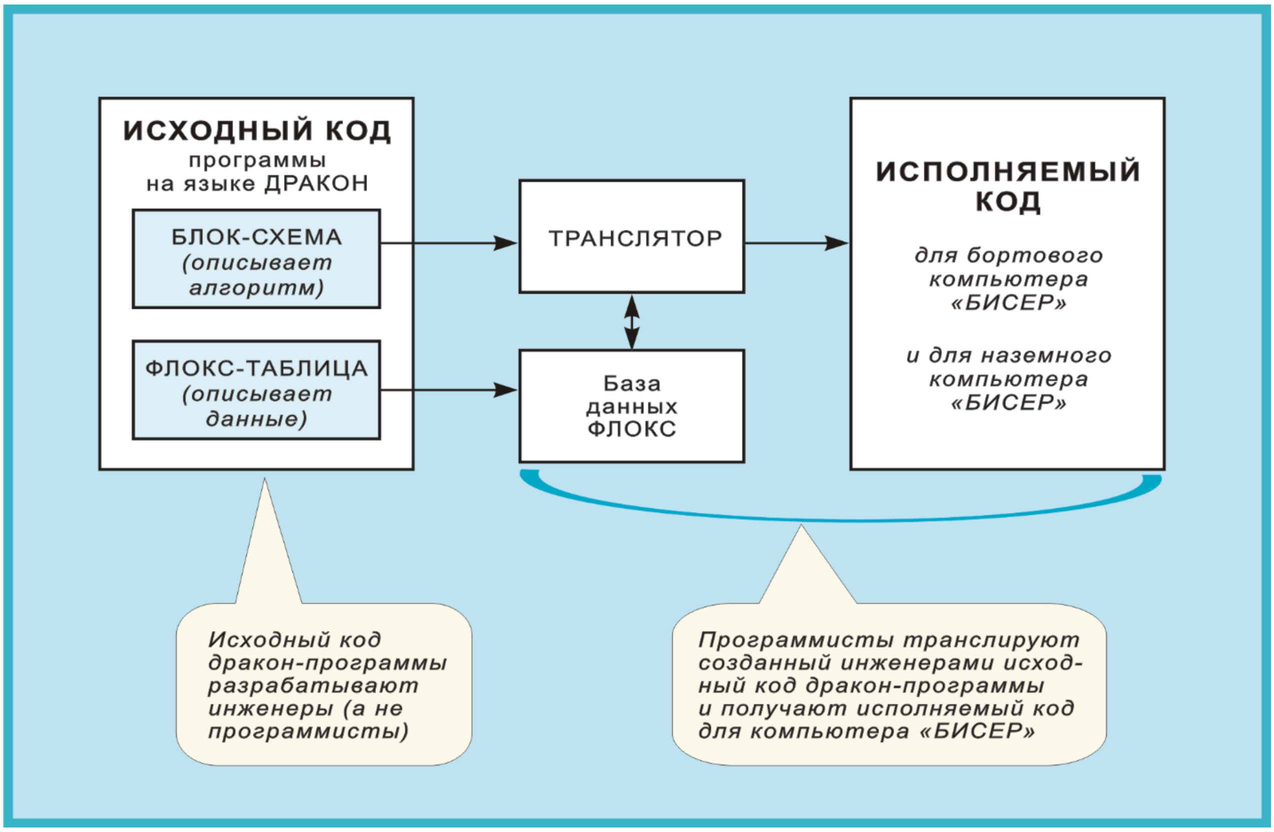
Алгоритмический язык

Который

Обеспечивает

Наглядность

Рис. 1. Кадр из фильма «Жирограф и ДРАКОН Пилюгина» [5].



Исходный код дракон-программы разрабатывают инженеры (а не программисты)

Программисты транслируют созданный инженерами исходный код дракон-программы и получают исполняемый код для компьютера «БИСЕР»

Рис. 2. Структурная схема ДРАКОН-технологии

Шаг 1. Получение и согласование исходных данных на разработку системы управления ракеты (СУ).

Шаг 2. Разбиение СУ ракеты на функциональные тракты и разработка приборного состава СУ.

Шаг 3. Определение набора понятий, необходимых для разработки СУ и программного обеспечения. Понятиями считаются алгоритмы, параметры, команды, сигналы и массивы. Например, при разработке разгонного блока космических аппаратов ДМ-SL проекта «Морской старт» использованы 4000 понятий.

Шаг 4. Для каждого понятия записывают уникальное формальное имя – флокс-идентификатор (далее – идентификатор).

Шаг 5. На языке ФЛОКС для каждого идентификатора записывают паспорт идентификатора (т.е. данные, необходимые при трансляции исходного кода ДРАКОН-программы). Идентификатор и связанные с ним данные образуют флокс-описание. Несколько флокс-описаний образуют флокс-таблицу.

Шаг 6. Инженеры разрабатывают флокс-таблицы с помощью программы «флокс-редактор».

Шаг 7. Инженеры разрабатывают блок-схемы алгоритмов (дракон-схемы) с помощью программы «графит-редактор». В блок-схемах разрешается употреблять только те идентификаторы, которые определены во флокс-таблицах. В противном случае транслятор сообщает об ошибке.

Шаг 8. Инженеры передают исходный код каждой дракон-программы программистам в виде двух файлов: блок-схемы и флокс-таблицы.

Особенность языка ДРАКОН в том, что он содержит две части (два набора правил):

— для записи алгоритмов (эта часть условно называется «процедурный язык ГРАФИТ»),

— для записи данных (эта часть условно называется «декларативный язык ФЛОКС»).

Понятно, что ГРАФИТ и ФЛОКС не являются самостоятельными языками; они являются двумя «половинками» языка программирования ДРАКОН.

На рис. 4 показана последовательность работ, выполняемых программистами по созданию исполняемого кода и формированию модели памяти, предназначенной для загрузки в бортовой или наземный компьютер «Бисер», разработанный в НПЦАП.

Компилятор ДРАКОНа содержит три блока: транслятор, анализатор и кодогенератор. Компилятор преобразует исходный код в код ассемблера. Встретив в алгоритме идентификатор, компилятор обращается к базе данных ФЛОКС, находит нужный идентификатор и заменяет идентификатор на соответствующие ему данные.

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

КТО ДОЛЖЕН ФОРМАЛИЗОВАТЬ ЗНАНИЯ

Язык ДРАКОН опирается на принцип: «Кто обладает знаниями, тот и должен их формализовать». Этот принцип вступает в противоречие с обычной практикой, согласно которой формализацию знаний для исполнения на компьютере выполняют специалисты по программированию.

Чтобы пояснить суть дела, зададим вопрос. Кто обладает прикладными знаниями при создании системы управления космической ракеты: инженеры или программисты? Конечно, инженеры.

Именно инженеры (инженеры-комплексники) изучают исходные данные на разработку системы управления космической ракеты. Именно они совместно с математиками разбивают систему управления на функциональные тракты и разрабатывают приборный состав СУ ракеты-носителя или разгонного блока космического аппарата. Именно инженеры определяют состав и объем проверок на комплексном стенде НПЦАП, на техническом и стартовом комплексах космодрома. Именно инженеры (а отнюдь не программисты) досконально знают «физику процесса». По этой причине, никто лучше инженера не знает и не может знать алгоритмы и структуры данных прикладных задач, решаемых системой управления.

А раз так, именно инженеры (а отнюдь не программисты) должны формализовать свои собственные профессиональные знания и превратить их в прикладные программы.

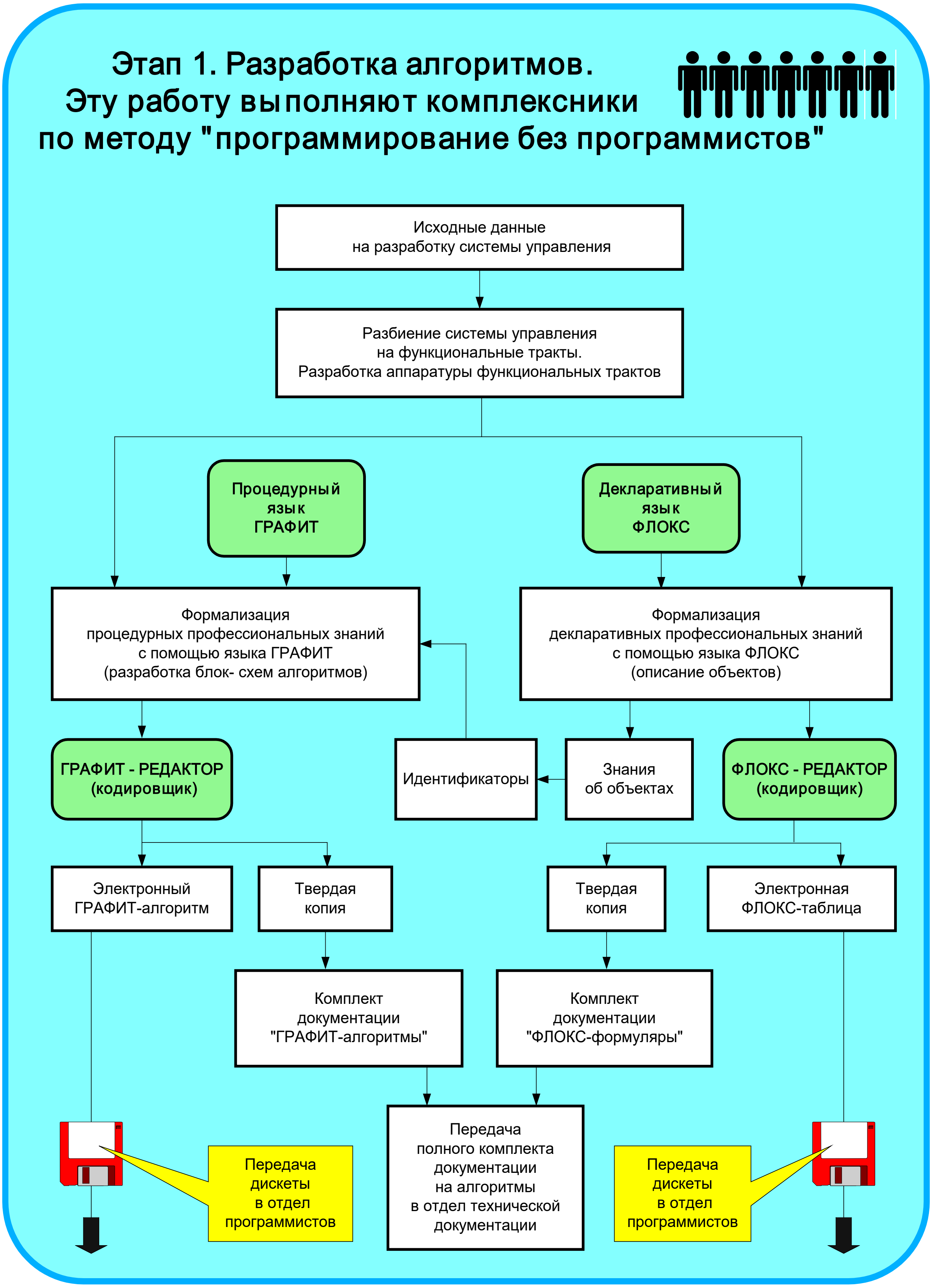


Рис. 3. На первом этапе ДРАКОН-технологии инженеры-комплексники разрабатывают исходный код ДРАКОН-программ и передают его программистам

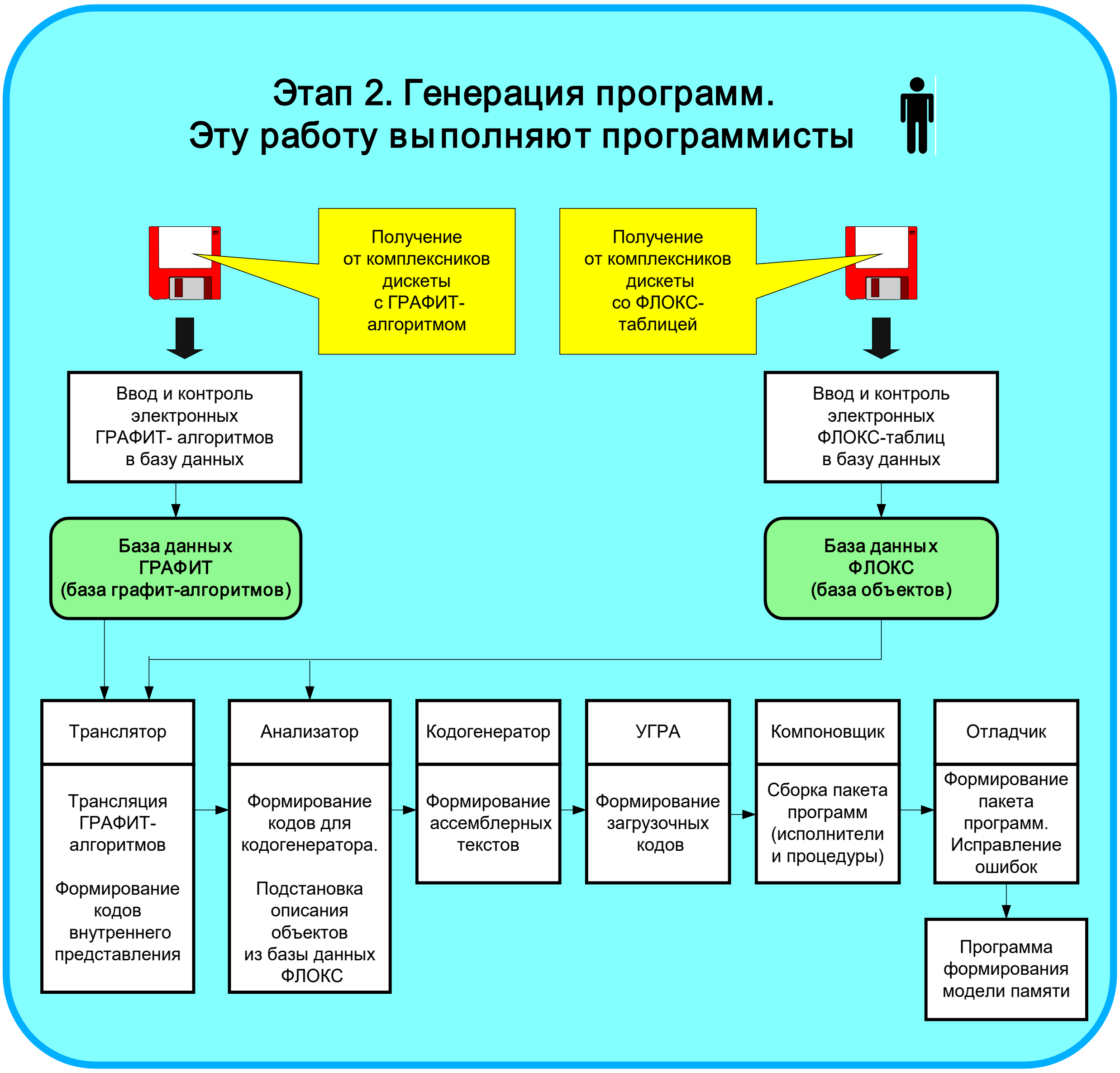


Рис. 4. На втором этапе ДРАКОН-технологии к работе подключаются программисты. Они преобразуют исходный код ДРАКОН-программ в исполняемый код компьютера «Бисер», формируют модели памяти, загружают модели памяти в оперативную память и флеш-память компьютера «Бисер» и т.д.

На этом пути возникает серьезное препятствие. Дело в том, что традиционные средства математической формализации знаний (в виде математического и программного обеспечения ЭВМ) оказываются чрезвычайно трудными — непосильными для инженера. К счастью, эту преграду удалось преодолеть при создании ДРАКОНа с помощью когнитивно-эргономических методов [4, с. 253, 495, 496].

ЭРГОНОМИЗАЦИЯ ЯЗЫКА ДРАКОН

Эргономичный алгоритм – алгоритм, позволяющий минимизировать интеллектуальные усилия, необходимые для его понимания человеком и облегчить выявление ошибок при зрительном восприятии текста алгоритма. 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 [2, с. 105].

Эргономизация языка программирования подразумевает облегчение восприятия и понимания человеком средств проектирования, программирования и сопровождения программ. Она опирается на постулат *(принцип Дракона)*:

> «Язык – важнейший интеллектуальный инструмент разработчиков и программистов. Чем удобнее язык, чем лучше он адаптирован к решаемой задаче, тем лучше работает человеческий мозг, выше производительность труда».

Разработчики языка ДРАКОН исходили из следующих соображений. Процесс достижения интеллектуального взаимопонимания между соисполнителями сложных проектов – один из наиболее сложных видов умственного труда. Производительность этого труда недопустимо мала и разительно отстает от растущих потребностей, связанных с ростом объемов и сложности бортового и наземного программного обеспечения. Чтобы переломить неблагоприятную тенденцию, необходимо поднять уровень взаимопонимания и эффективность взаимодействия между специалистами. Язык ДРАКОН и ДРАКОН-технология специально ориентированы на решение данной проблемы за счет предоставления персоналу новых языковых и инструментальных средств.

Эргономичный язык ДРАКОН [4, с. 8, 54, 487] позволил устранить возникшие перед инженерами трудности понимания без потери математической строгости. Благодаря этому упомянутый выше *принцип Дракона* получил постоянную прописку в НПЦАП при разработке систем управления ракет-носителей и разгонных блоков космических аппаратов.

ПРОГРАММИРОВАНИЕ БЕЗ ПРОГРАММИСТОВ

В НПЦАП язык ДРАКОН позволил реализовать метод «Программирование без программистов» при разработке исходного кода программ. Данный метод можно считать одной из основных идей ДРАКОН-технологии.

Автор термина «Программирование без программистов» — американский ученый Джеймс Мартин. В 1982 году Мартин опубликовал книгу под названием «Разработка прикладных программ без программистов» (Applications development without programmers) [6].

Книга Мартина дала начало новому направлению исследований, которое обычно для краткости называют «Программирование без программистов» (хотя фактически речь идет только о программировании без *прикладных* программистов»).

В работе Мартина говорится о непроцедурных языках. Разработчики ДРАКОНа приняли у Мартина идею «Программирование без программистов». Вместе с тем, они решительно отказались от сделанного им выбора в пользу непроцедурных языков. И сделали диаметрально противоположный выбор. ДРАКОН — процедурный, императивный язык программирования.

Инженеры, сотрудники комплексных отделов НПЦАП, успешно освоили инструментальные средства языка ДРАКОН (флокс-редактор для описания данных и графит-редактор для описания алгоритмов). Эти средства стали их рабочими инструментами при разработке визуальных алгоритмов и программ.

Благодаря систематическому использованию флокс-редактора и графит-редактора инженеры-комплексники:

— освоили метод формализации при разработке исходного кода ДРАКОН-программы;

— научились использовать формальный метод при практической разработке алгоритмов и программ для *Системы управления бортовыми системами ракеты*, а также для всех видов наземных испытаний СУ ракеты. Имеются в виду:

- регламентные испытания,
- проверка электрических цепей ракеты (защитные операции),
- автономные испытания (проверочные включения),
- комплексные испытания,
- контрольный набор стартовых готовностей,
- предстартовая подготовка и пуск [7].

Главное достижение в том, что сотрудники комплексных отделов передают работу программистам в математически строгой форме – в виде исходного кода ДРАКОН-программ *(исходники)*, полностью готовые для трансляции в исполняемые *(executable)* коды.

Таким образом, ДРАКОН предоставил инженерам новые интеллектуальные инструменты, усиливающие их творческие возможности. И позволяющие решать задачи, которые ранее считались *непосильными* для инженеров.

ДРАКОН — это усилитель естественного человеческого интеллекта. Или, что одно и то же, средство для улучшения работы ума [2-4].

ПРИМЕРЫ ДРАКОН-ОПЕРАТОРОВ РЕАЛЬНОГО ВРЕМЕНИ

Предположим, управляющий компьютер должен выдать серию электрических команд, которые по линиям связи передаются в исполнительные органы и вызывают срабатывание электромеханических или иных реле. В результате происходит открытие трубопровода, включение насоса и другие операции, необходимые для функционирования управляемого объекта.

Будем считать, что управляющий компьютер должен:

— выдать команду ОТКРЫТЬ.ТРУБОПРОВОД;

— подождать 2 минуты;

— выдать две команды: ВКЛЮЧИТЬ.НАСОС и ОТКРЫТЬ.ЗАСЛОНКУ;

— подождать 45 секунд;

— выдать команду ПОДАЧА.ТОПЛИВА;

— подождать 3 минуты;

— выдать команду ПУСК.АГРЕГАТА.

Соответствующий алгоритм представлен на рис. 5 (слева). Задержка выдачи команд реализуется с помощью иконы «пауза». Внутри последней указывается время необходимой задержки. Например, 2мин (2 минуты), 45с (45 секунд) и т. д.

Верхний оператор «пауза» на рис. 5 работает так. После выдачи команды ОТКРЫТЬ.ТРУБОПРОВОД в управляющем компьютере запускается программный счетчик времени на 2 минуты. По истечении этого времени компьютер выдает в линию связи команды ВКЛЮЧИТЬ.НАСОС и ОТКРЫТЬ.ЗАСЛОНКУ.

Изменим задачу. Предположим, разработчик управляемого объекта хочет указать время выдачи команд не по принципу «задержка после предыдущей команды», а по принципу секундомера. Это значит, что все времена отсчитываются от единого начального момента (совпадающего с пуском секундомера).

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

— включить «секундомер», то есть обнулить и запустить 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.

Описанный алгоритм изображен на рис. 5 (справа). В нем используются операторы «пуск таймера» и «синхронизатор», совместная работа которых обеспечивает нужный эффект.

Оператор «пуск таймера» порождает, обнуляет и запускает таймер и присваивает ему имя *A*. Оператор «синхронизатор» задерживает выполнение размещенного справа от него визуального оператора до наступления момента, указанного в иконе «синхронизатор».

Например, синхронизатор *A* = 2мин45с на рис. 5 (справа) задерживает выдачу команды ПОДАЧА.ТОПЛИВА до момента, когда таймер *A* отсчитает 2 минуты 45 секунд.

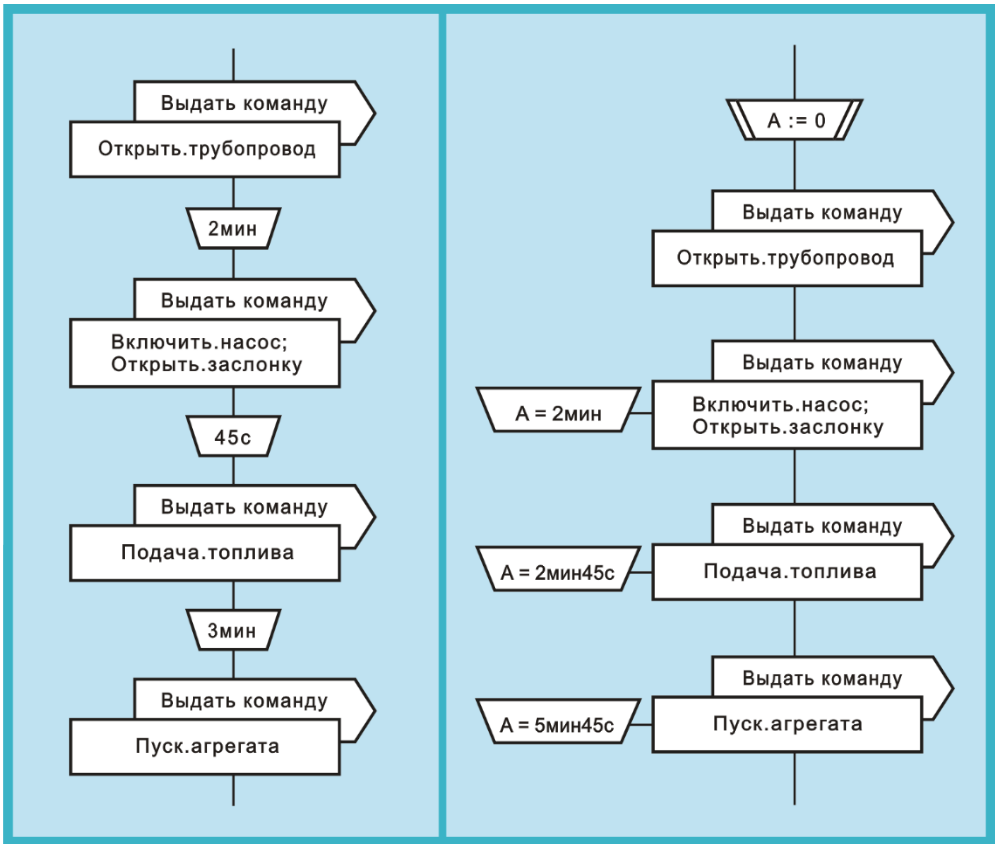


Рис. 5. Примеры операторов реального времени

Сравнивая алгоритмы на рис. 5, можно заметить, что они почти эквивалентны. Почему почти?

Чтобы разобраться, рассмотрим идеальный случай. Предположим, что время, необходимое для выдачи одной команды равно нулю. Имеется в виду, что перечисленные ниже команды выдаются за время, равное нулю:

— ОТКРЫТЬ.ТРУБОПРОВОД;

— ВКЛЮЧИТЬ.НАСОС;
— ОТКРЫТЬ.ЗАСЛОНКУ;
— ПОДАЧА.ТОПЛИВА;
— ПУСК.АГРЕГАТА.

В этом случае оба алгоритма будут выдавать команды синхронно.

Однако в действительности идеальные случаи встречаются далеко не всегда. Иногда бывает, что время выдачи одной команды больше нуля.

В таком случае алгоритмы работают по-разному. Практика показывает, что в некоторых ситуациях предпочтительным является *принцип паузы* (когда используется икона «пауза»). А в других – *принцип таймера* (когда используются иконы «пуск таймера» и «синхронизатор»). Оба инструмента оказываются в равной степени необходимыми и полезными [4, с. 209–212].

СТРУКТУРА ФЛОКС-ИДЕНТИФИКАТОРА

Рассмотрим пример. Предположим, нужно включить бортовую систему УМ. Для этого необходимо включить два фидера электропитания: УМ1 и УМ2. Соответствующая ДРАКОН-схема показана на рис. 6. Схема содержит 4 идентификатора:

АП1УП.ВКЛЮЧИТЬ.УМ
КС1УП.ВКЛЮЧИТЬ.УМ1
КС1УП.ВКЛЮЧИТЬ.УМ2
ПЛ1УП.УМ.ВКЛЮЧЕНА

Назначение идентификатора (тип данных) указывают первые два символа:

АП — Алгоритм Процедура
КС — Команда Силовая
ПЛ — Признак Логический (однобитовый)

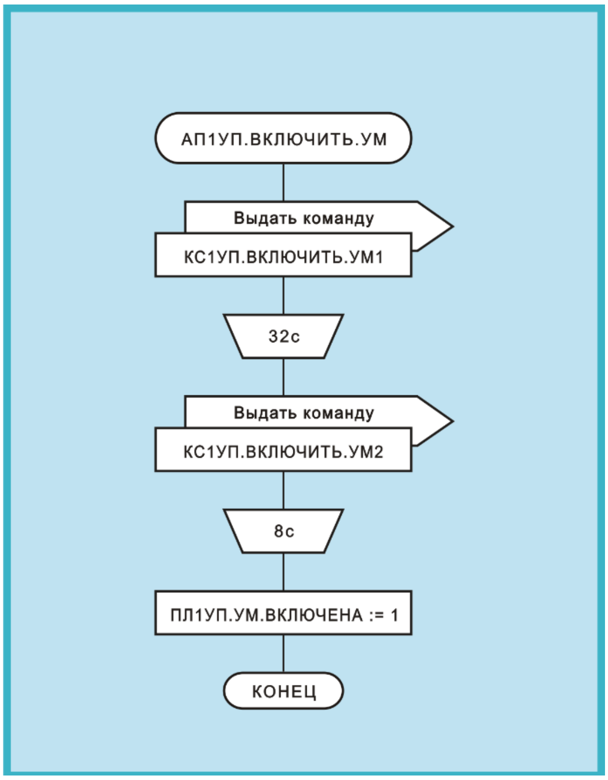


Рис. 6. Пример ДРАКОН-схемы

Все идентификаторы имеют стандартную структуру и состоят из двух полей: префикс и смысловая часть (рис. 7).

Префикс – позиционная часть идентификатора, где в каждой позиции информация записывается по формальным правилам языка ФЛОКС.

Смысловая часть имеет непозиционный характер и пишется на естественном для инженера языке.

Если два идентификатора отличаются хотя бы в одном символе, они считаются различными. Чтобы быстро уяснить физический смысл идентификатора, надо пропустить префикс и читать только смысловую часть.

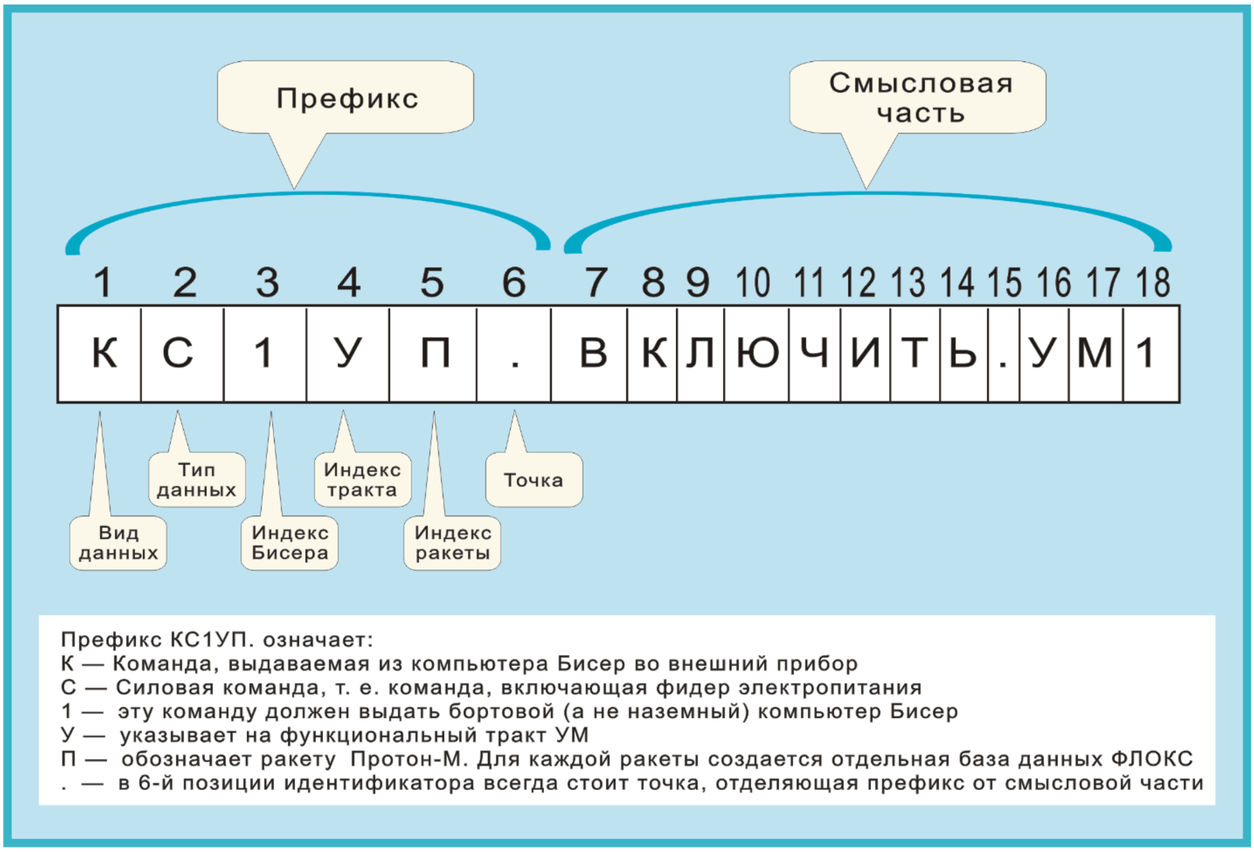
ОСОБЕННОСТИ ДРАКОН-МЕТОДОЛОГИИ

При разработке сложных программных комплексов традиционная технология работы имеет существенный недостаток. Он связан с взаимоотношениями между Заказчиком и Исполнителем. Болевая точка проблемы сконцентрирована на документе под названием «Техническое задание на разработку программного комплекса» (Software requirements specification).

Подразумевается, что Заказчик совместно с Исполнителем создают Техническое задание на разработку программы. Далее работа по разработке и отработке программного обеспечения возлагается на программистов Исполнителя. А Заказчик полностью отстраняется от разработки программ вплоть до приемо-сдаточных испытаний. Проблема в том, что созданный программистами исходный код программы (source code) требует знания языка программирования и непонятен Заказчику. Почему?

Как отмечает академик А.П. Ершов, «язык программирования кодирует объекты предметной области задачи, а наше знание об этих объектах остается за пределами программного текста» [8]. Это значит, что профессиональное знание Заказчика об объектах предметной области не отражается в тексте программы, что плохо. Поэтому понять сложную программу в отсутствие ее авторов-программистов очень трудно или даже невозможно. Приходится признать, что известные методы формализации знаний с помощью языков программирования (таких как Си, Си++, Си#, Java, Ada и т. д.) несовершенны [3, с. 44, 45]. Они не подходят для многих специалистов, чья работа связана с алгоритмами, но которые по разным причинам не умеют выражать свои профессиональные знания в форме алгоритмов и программ на указанных языках.

Вторая проблема состоит в том, что передача знаний от Заказчика к Исполнителю через Техническое задание чрезвычайно сложна, разработка ТЗ требует большой трудоемкости.



Префикс КС1УП. означает:
К — Команда, выдаваемая из компьютера Бисер во внешний прибор
С — Силовая команда, т. е. команда, включающая фидер электропитания
1 — эту команду должен выдать бортовой (а не наземный) компьютер Бисер
У — указывает на функциональный тракт УМ
П — обозначает ракету Протон-М. Для каждой ракеты создается отдельная база данных ФЛОКС
. — в 6-й позиции идентификатора всегда стоит точка, отделяющая префикс от смысловой части

Рис. 7. Структура флокс-идентификатора

АВТОФОРМАЛИЗАЦИЯ ПРОФЕССИОНАЛЬНЫХ ЗНАНИЙ ИНЖЕНЕРОВ

Техническое задание несет в себе трудно обнаруживаемые ошибки и зачастую содержит неясности и умолчания. Как отмечают эксперты, 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» [3, с. 59, 60].

Для борьбы с этими недостатками предложен ряд методов, например, быстрая разработка приложений (rapid application development), прототипирование программного обеспечения с вовлечением пользователей в процесс разработки программ (software prototyping), инженерия требований (software requirements specification) и т. д.

Интересную идею выдвинул Г.Р. Громов под названием «автоформализация профессиональных знаний» [9]. Автоформализация — это не автоматическая формализация, а самоформализация, т. е. формализация знаний, которую человек выполняет САМ. Автоформализация полезна тем, что позволяет устранить ненужных посредников и избежать ошибок типа «испорченный телефон», вызванных взаимным непониманием. Имеется в виду, что формализация профессиональных знаний Заказчика должна выполняться самим Заказчиком (а не Исполнителем). Слабое место идеи Громова в том, что он не указал средство для реализации принципа автоформализации.

В качестве такого средства можно использовать язык ДРАКОН [3, с. 194-196, 225]. В НПЦАП реализован метод автоформализации, имеющий следующие особенности.

1. Сотрудники комплексных отделов НПЦАП (инженеры) выполняют автоформализацию своих профессиональных знаний без участия программистов и разрабатывают исходный код ДРАКОН-программы самостоятельно.
2. Автоформализацию декларативных и процедурных знаний (описание данных и разработку алгоритмов) инженеры выполняют с помощью флокс-редактора и графит-редактора соответственно.

МИНИМИЗАЦИЯ НАГРУЗКИ НА ИНЖЕНЕРОВ

В 1996 году в НПЦАП было принято решение о внедрении ДРАКОН-технологии и возложении на инженеров-комплексников задачи по разработке дракон-алгоритмов и флокс-таблиц с выпуском учтенной технической документации согласно стандарту ЕСКД. Чтобы минимизировать нагрузку на инженеров, связанную с разработкой программ, было решено:

1. Для каждой вновь создаваемой или модернизируемой СУ космической ракеты инженеры должны разработать исходный код ДРАКОН-программ и передать его программистам, в том числе разработать комплект дракон-алгоритмов и комплект флокс-таблиц.
2. Все остальные работы (преобразование исходного кода в исполняемый код, формирование моделей памяти и др.) выполняют программисты. Это значит, что инженеры освобождаются от необходимости читать, понимать и анализировать ассемблерные и машинные листинги и иную подобную работу.
3. Чтобы в максимальной степени облегчить работу инженеров, язык ДРАКОН построен с учетом «кругозора» инженера и, как правило, использует хорошо знакомые инженеру термины и понятия. Памятуя замечание А.П. Ершова, язык программирования ДРАКОН дает возможность инженерам ввести в состав программного текста известные им знания «об объектах предметной области задачи», т. е. о системе управления ракеты, о приборах и бортовых системах, из которых она состоит и пр.

ОБСУЖДЕНИЕ

Существующие способы записи алгоритмов (принятые во всем мире) слишком трудны для непрограммистов и требуют неоправданно больших трудозатрат [4, с. 504, 505]. Это обстоятельство ставит непреодолимый барьер для многих инженеров Роскосмоса, работа которых связана с алгоритмами, но которые не имеют резерва времени, чтобы научиться выражать свои профессиональные знания в форме алгоритмов и программ. Эргономичный язык ДРАКОН значительно помогает решению данной проблемы.

Благодаря этому новшеству алгоритмы становятся более понятными и доступными. В итоге ТРУДНЫЕ для понимания способы записи алгоритмов заменяются на более ЛЕГКИЕ. Вследствие этого инженеры НПЦАП быстро овладевают дракон-схемами. ДРАКОН снижает риск появления ошибок, не выявленных по результатам отработки программного обеспечения. Это достигается за счет наглядности, удобочитаемости, эргономичности и строгой формализации дракон-схем и флокс-описаний.

ДРАКОН-технология коренным образом изменяет механизм профессионального взаимодействия инженеров и программистов. Секрет успеха в том, что именно инженеры (а отнюдь не программисты) обладают всей полнотой знаний об особенностях функционирования ракет-носителей и разгонных блоков космических аппаратов.

Инженеры «первичны», программисты «вторичны» в том смысле, что программисты получают знания об объекте «из вторых рук» — из рук инженеров. Однако до появления ДРАКОНа инженеры были лишены интеллектуального инструмента, который позволил бы им представить их собственные профессиональные знания в удобной, понятной и одновременно математически строгой форме. Благодаря ДРАКОНу разработка алгоритмов и исходного кода прикладных программ превращается в автоформализацию профессиональных знаний инженеров.

РЕЗУЛЬТАТЫ

Язык ДРАКОН и ДРАКОН-технология эксплуатируются в НПЦАП уже 20 лет (1996–2016 годы). За это время они помогли создать системы управления четырнадцати космических проектов:

— разгонный блок космических аппаратов ДМ-SL (проект «Морской старт»);

— модернизированная ракета-носитель тяжелого класса ПРОТОН-М;

— семейство разгонных блоков ФРЕГАТ, которое включает: просто ФРЕГАТ, ФРЕГАТ-СБ (со сбрасываемыми баками), ФРЕГАТ-УТТХ (с улучшенными тактико-техническими характеристиками), ФРЕГАТ-МТ, предназначенный для запусков из Южной Америки, ФРЕГАТ-СУМ (с модернизированной системой управления);

— разгонный блок космических аппаратов ДМ-SLБ (проект «Наземный старт»),

— разгонный блок космических аппаратов ДМ-03,

— первая ступень южнокорейской ракеты-носителя KSLV,

— три проекта семейства Ангара: ракета-носитель легкого класса АНГАРА-1.2 первого пуска, ракета-носитель легкого класса АНГАРА-1.2 с агрегатным модулем, ракета-носитель тяжелого класса АНГАРА-А5;

— разгонный блок КВТК (кислородно-водородный тяжелого класса) [5, с. 515].

Пуски ракет-носителей и разгонных блоков, при создании которых использовался язык ДРАКОН, производятся или производились с пяти космодромов мира: Плесецк, Байконур, Kuru (Французская Гвиана), плавучий космодром «Морской старт» (экваториальная зона Тихого океана), Naro (Южная Корея).

ЯЗЫК ДРАКОН ЗА РАМКАМИ НПЦАП

Язык ДРАКОН применяется не только в НПЦАП, но и за его пределами — в областях, далеких от ракетно-космической тематики. В 1996 году Государственный комитет Российской Федерации по высшему образованию включил изучение языка ДРАКОН в программу курса «Информатика» [10]. В некоторых университетах организовано обучение ДРАКОНу, например, в СибГИУ, КемГУ и др.

При этом используется качественно иной (по сравнению с НПЦАП) принцип работы. ДРАКОН можно присоединить к некоторым языкам программирования и получить так называемые гибридные языки, например: Дракон-С, Дракон-Java, Дракон-С#, Дракон-Python, Дракон-Javascript, Дракон-Tcl, Дракон-Erlang, Дракон-Lua и др.

В Интернете можно скачать 4 бесплатных редактора для работы с языком ДРАКОН. Более полно эта сторона дела отражена в Википедии в статье «ДРАКОН» и на официальном форуме языка ДРАКОН https://forum.drakon.su/viewforum.php?f=151

ЧАСТЬ 2. ЯЗЫК ДРАКОН В МЕДИЦИНЕ

ВРАЧЕБНЫЕ ОШИБКИ И БЕЗОПАСНОСТЬ ПАЦИЕНТОВ

Врачебные ошибки опасны тем, что могут привести к смерти, стойкой инвалидности пациентов или причинить иной вред. До последнего времени статистика появления ошибок не была известна. Ситуация изменилась в 2000 году, когда был опубликован 300-страничный доклад Национальной академии медицины США (National Academy of Medicine, Institute of Medicine) под названием «Человеку свойственно ошибаться: создание более безопасной системы здравоохранения» [11].

На основании тщательных исследований было установлено, что медицинские ошибки в больницах США являются причиной смерти от 44 000 до 98 000 человек в год [11, р. 1, 26, 31]. Это значит, что «в американских больницах каждые полгода погибает больше американцев, чем за всю Вьетнамскую войну» [12].

Авторы доклада признают, что врачебные ошибки занимают одно из ведущих мест в структуре смертности населения США. Даже если взять нижнюю оценку (44 000 человек), смертность из-за врачебных ошибок превышает значение восьмой ведущей причины смерти в США. По вине врачей умирает больше людей, чем от дорожно-транспортных происшествий (43 458 жертв), от рака молочной железы (42 297 жертв), от СПИДа (16 516 жертв) [11, р. 1, 26].

В докладе делается вывод, что больница гораздо опаснее самолета. Потому что риск смерти вследствие врачебной ошибки намного больше, чем риск гибели в авиационной аварии («risk of dying as a result of a medical error far surpasses the risk of dying in an airline accident») [11, р. 42].

По рекомендациям Академии медицины в конгрессе США были проведены слушания и принят закон о безопасности пациентов (Patient Safety and Quality Improvement Act of 2005), подписанный президентом Джорджем Бушем мл. 29 июля 2005 года. Таким образом, понятие безопасности пациентов приобрело не только научный, но и юридический статус.

Доклады Национальной академии медицины США [11, 13-15] впервые поставили в центр научного исследования исключительно трудную междисциплинарную проблему — проблему врачебных ошибок. В докладах предложена развернутая программа противодействия этому злу в интересах безопасности пациентов.

КРИТИКА

Указанная программа уязвима для критики. Врачебные ошибки зависят от многих причин, в том числе, от недостатков профессионального медицинского языка, который, будучи естественным языком, не приспособлен для точного и удобного описания медицинских алгоритмов и не имеет необходимых для этого специальных средств.

Алгоритмы профилактики, диагностики, лечения, скорой помощи, реанимации, реабилитации, прогноза являются научной проблемой первостепенной важности, которая имеет прямое отношение к предотвращению врачебных ошибок и безопасности пациентов. Однако эта проблема полностью упущена из виду в указанных докладах, что снижает ценность их выводов и рекомендаций.

МЕДИЦИНСКИЙ АЛГОРИТМ И ЕГО НЕДОСТАТКИ

Понятие *алгоритм* в медицинском мире не имеет единого строгого определения, допускает различные толкования и графические представления. Стандартизация отсутствует. Главный недочет большинства медицинских трактовок понятия *алгоритм* — отсутствие необходимой точности, приблизительность, недосказанность, отсутствие нужных уточнений, подробностей и условий.

В медицинских учебниках, стандартах, руководствах используются неточные, приблизительные, неудовлетворительные описания медицинских алгоритмов. Подобные описания нарушают элементарные алгоритмические и эргономические правила, провоцируют врачебные ошибки и опасны для пациентов.

Все эти погрешности зачастую исключают возможность воспроизвести алгоритм, опираясь на его описание. Это значит, что нельзя воспроизвести (повторить) процессы диагностики и лечения, используя только алгоритмы. Потому что указанные процессы описаны в алгоритмах далеко не полностью.

ТЕРМИНОЛОГИЯ

С математической точки зрения, медицинский алгоритм — это не алгоритм, а всего лишь алгоритмическое предписание (algorithmic prescription, workflow). Мы будем опираться на тезис, который сформулировал математик Н.Н. Непейвода: «Необходимо различать алгоритм и алгоритмическое предписание, имеющее внешнюю форму алгоритма, но включающее не до конца определенные шаги» [16].

На этот счет существует обширная литература, восходящая к пионерской работе Л.Н. Ланды «Алгоритмизация в обучении» [17]. Ланда первым указал различие, но мы предпочитаем более точную формулировку Непейводы.

Тезис Ланды-Непейводы создает научный фундамент для четкого разграничения медицинских и математических алгоритмов.

ЯЗЫК ДРАКОН

Согласно развиваемой идее, прежние (неточные, не полностью определенные) описания медицинских алгоритмов должны со временем отмереть, навсегда сойти со сцены. И уступить место принципиально новому типу алгоритмов — *медицинским алгоритмам высокой точности*. Последние, оставаясь алгоритмическими предписаниями, приближаются по своим свойствам (в пределах возможного) к настоящим алгоритмам. Здесь имеется в виду, что *точность алгоритма* — синоним термина «определенность (детерминированность) алгоритма».

В качестве медицинского алгоритмического языка высокой точности предлагается использовать язык ДРАКОН, который имеет существенные преимущества перед конкурирующими визуальными языками.

ОСНОВНАЯ МЫСЛЬ

Профессиональный медицинский язык (язык медицинской литературы, учебников, стандартов, руководств, клинических рекомендаций, протоколов) имеет серьезный недостаток. Он недостаточно точен и плохо приспособлен для описания сложных и разветвленных медицинских действий и решений, выполняемых при профилактике, диагностике, лечении и пр.

Чтобы устранить дефект, нужно осуществить глубокую реформу медицинского языка, расширив его возможности с помощью визуального медицинского алгоритмического языка — языка ДРАКОН. Последний предназначен для стимулирования клинического мышления врачей, повышения безопасности пациентов, предотвращения врачебных ошибок и стандартизации представления медицинских алгоритмов в медицинской литературе [18, 19].

АПРОБАЦИЯ

Литовские врачи первыми обратили внимание на возможность крупномасштабного использования языка ДРАКОН в медицине. За последнее время в Литве изданы четыре медицинских учебника на русском языке, в которых используется ДРАКОН.

1. Начальная неотложная акушерская помощь [20].
2. Специализированная реанимация новорожденного [21].
3. Неотложная медицинская помощь [22].
4. Травма [23].

Учебники апробированы в ряде стран (Литва, Казахстан, Азербайджан, Таджикистан, Туркменистан, Киргизия) в рамках курсов повышения квалификации врачей. Курсы проводили высококвалифицированные специалисты из Литовского университета медицинских наук (Lietuvos sveikatos mokslų universitetas) для местных врачей. Проведенная апробация дала положительные результаты [24].

В Российской медицине аналогичная работа пока еще находится в начальной стадии. Имеются лишь отдельные упоминания о языке ДРАКОН в научных публикациях и диссертациях [25]. Более полная информация на эту тему представлена в Википедии, в статье «Медицинский алгоритм».

Предотвращение врачебных ошибок и повышение безопасности пациентов в России, как и во всем мире, является актуальной задачей. Визуальный алгоритмический язык ДРАКОН, по нашему мнению, может содействовать ее успешному решению.

ЛИТЕРАТУРА

1. *Паронджанов В. Д.* Графический синтаксис языка ДРАКОН // Программирование, 1995, №3, С. 45—62.
2. *Паронджанов В. Д.* Как улучшить работу ума. Алгоритмы без программистов — это очень просто! — М.: Дело, 2001. — 360 с. — http://drakon.su/_media/biblioteka_1/parondzhanov_v.d._kak_uluchshit_rabotu_uma_.pdf
3. *Паронджанов В. Д.* Дружелюбные алгоритмы, понятные каждому. Как улучшить работу ума без лишних хлопот. — М.: ДМК-пресс, 2010. — 464 с. — http://drakon.su/_media/biblioteka_1/03._2010_druzheljubnye_algoritmy_1.pdf
4. *Паронджанов В. Д.* Учись писать, читать и понимать алгоритмы. Алгоритмы для правильного мышления. Основы алгоритмизации. — М.: ДМК Пресс, 2014. — 520 с. — http://drakon.su/_media/biblioteka_1/01._2012_uchis_chitat_new_end_podlinnik.pdf
5. *Бурцева Н., Петров.* Жирограф и ДРАКОН Пилюгина. Телерадиостудия Роскосмоса. (17 мая 2008). — Фильм выпущен к 100-летию со дня рождения Главного конструктора

систем управления ракет-носителей академика Н. А. Пилюгина. http://tvroscosmos.ru/frm/vestidata/2008/vesti17_5_8_2.php

6. *James Martin.* Application Development without Programmers, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982. — 368 p.
7. *Морозов В.В., Трунов Ю.В. и др.* Система управления межорбитального космического буксира «Фрегат» // Вестник «ФГУП НПО им. С.А. Лавочкина», 2014, №1. — С. 16-25.
8. *Ершов А. П.* Предварительные соображения о лексиконе программирования // Кибернетика и вычислительная техника. Вып. 1. — М.: Наука, 1985. — С. 207.
9. *Громов Г. Р.* Очерки информационной технологии. — М.: Инфоарт, 1993. — 336с. — С. 143–158, 265–299.
10. Примерная программа дисциплины «Информатика». Издание официальное. — М.: Госкомвуз, 1996. — 21 с. / См. разделы 3 и 4, а также Приложение, пункты 1-7. http://drakon.su/_media/biblioteka/progr_drakon.pdf
11. To Err is Human: Building a Safer Health System / Linda T. Kohn, Janet M. Corrigan, and Molla S. Donaldson, editors. — Committee on Quality of Health Care in America, Institute of Medicine. 2000. — 312 p.
12. *Hayward R.A., Hofer T.P.* Estimating Hospital Deaths Due to Medical Errors: Preventability Is in the Eye of the Reviewer // JAMA: the Journal of the American Medical Association. — July 25, 2001, Vol. 286, No. 4. — pp. 415–420.
13. Crossing the Quality Chasm: a New Health System for the 21st Century. — Committee on Quality Health Care in America, Institute of Medicine. 2001. — xxi + 337p. — 364 p.
14. Preventing Medication Errors: Quality Chasm Series / Philip Aspden, Julie Wolcott, J. Lyle Bootman, Linda R. Cronenwett, Editors. — Committee on Identifying and Preventing Medication Errors. Board on Health Care Services. Institute of Medicine. 2007. — 480 p.
15. Improving Diagnosis in Health Care. / Erin P. Balogh, Bryan T. Miller, and John R. Ball, Editors. — Committee on Diagnostic Error in Health Care. Board on Health Care Services. Institute of Medicine. 2015. — The National Academies of Sciences, Engineering, and Medicine. — 472 p.
16. *Непейвода Н.Н.* Алгоритм // Новая философская энциклопедия: В 4-х т. / Ин-т философии РАН. / Под. ред. Степина В.С., Гусейнова А.А. и др. — М.: Мысль, 2010. — Том 1. — 744с. — С. 76.
17. *Ланда Л. Н.* Алгоритмизация в обучении. / Под общей ред. и со вступительной статьей Б. В. Гнеденко и Б. В. Бирюкова. — М.: Просвещение, 1966. — 523 с.
18. *Паронджанов В. Д.* Можно ли улучшить медицинский язык? // Человек. — №1. 2016.
19. *Паронджанов В. Д.* Алгоритмизация медицины и реформа медицинского языка // Евразийский союз ученых (ЕСУ) # 11 (20), 2015 | МЕДИЦИНСКИЕ НАУКИ — С. 144–156.
20. Начальная неотложная акушерская помощь. Учебник. / Под ред. Р. Й. Надишаускене. — Литва: Центр исследования кризисов, Университет наук здоровья Литвы, 2012. — 204с.
21. Специализированная реанимация новорожденного. Учебник. / Под ред. Р.Й. Надишаускене. — Литва: Центр исследования кризисов, Университет наук здоровья Литвы, 2012. — 396 с.
22. Неотложная медицинская помощь. Учебник. / Под ред. Д. Вайткайтиса. — Литва: Центр исследования кризисов, Университет наук здоровья Литвы, 2012. — 265 с.
23. Травма. Учебник. / Под ред. Д. Вайткайтиса. — Литва: Центр исследования кризисов, Университет наук здоровья Литвы. — 2012. — 440 с.
24. *Vileikytė A., Nadišauskienė R.J. et al.* Algoritminės „Drakon“ kalbos pritaikymas medicinoje // Lietuvos akušerija ir ginekologija. — 2014 rugsėjis, tomas XVII, Nr. 3. 192–196.
25. *Воронцова З. А., Шишкина В. В.* Алгоритмы в гистологии (эргономическое учебно-методическое пособие). — Международный журнал прикладных и фундаментальных исследований. – 2012. – № 9. – С. 43-44.