АРХИТЕКТУРА И МАТЕМАТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ
(Вычислительные системы)
1984 год Выпуск 104

УДК 519.685

ТЕХНОЛОГИЧЕСКИЕ АСПЕКТЫ СОВРЕМЕННЫХ ТЕНДЕНЦИЙ В ПРОГРАММИРОВАНИИ; ИНЖЕНЕРНЫЕ КОНЦЕПЦИИ

Д.И.Свириденко, Н.Т.Тукубаев

§ I. Введение

Программирование за 40-летнюю историю своего развития превратилось в один из самых социально сложных и значимых видов человеческой деятельности. Среди множества взглядов на природу этой деятельности, нашедших свое отражение в научной литературе, наиболее заметны два: во-первых, программирование - это ремесло, вид прикладного искусства, а программы являются художественными изделиями, поделками, и, во-вторых, программирование - это инженерная деятельность, а программы являются промышленными изделиями. Кроме них, в последнее время все большую популярность приобретает взгляд на программирование как на прикладную математическую деятельность, где программы рассматриваются как определенным образом формализованные знания. Фактически, эти три точки зрения отражают историю развития программирования.

На заре программирования (40-е и 50-е годы) программы создавались преимущественно для вычислительных задач и к программированию относились как к искусству. Превосходный анализ данного периода и соответствующей точки зрения на программирование мы можем найти у Ч.Уэзерелла [I]: "... Программирование - это ремесло, и каждый программист должен достичь нужного профессионального уровня. Надо сказать, что программированием, как правило, занимаются кустарно, в небольших организациях, где имеются лишь примитивные инструменты, многое делается вручную, необходимые сведения, в лучшем случае, черпаются у более опытных мастеров, а бывает, что получить их и вообще неоткуда. ... Классическое обучение

ремеслу состояло в том, что ученик в течение многих лет выполнял простейшие вспомогательные операции, перенимал основные приемы у более опытных работников..." Методы ремесленнического программирования - это традиции, передаваемые от мастера ученику, представляющие собой эмпирически найденные правила "хорошего" программирования. Главной заслугой данной точки зрения является то,что она указывает на необходимость серьезного отношения к обучению программированию, предъявляя серьезные эстетические требования к программам, что, естественно, предполагает высокую квалификацию программистов. Ремесленнический подход привлекателен своей неформальностью, ориентацией на творчество. Для этого подхода характерно незначительное внимание к вопросам организации процесса разработки программ. Носители этой точки зрения не желают (не умеют или считают ненужным) явно формулировать и отторгать от себя технологическое знание и умение, работая по принципу "смотри и делай как я (если сможешь, конечно)". Они считают бесполезным определение и изучение процесса программирования как самостоятельного понятия ("кто не умеет делать сам, тот учит этому другого"). Рассматриваемый период вообще характерен нестрогостью используемых понятий и большой эмоциональностью изложения результатов. Нужно сказать,что появление и распространение в последнее время персональных ЭВМ существенно укрепило позиции ремесленнической точки зрения (правда, уже на базе современной системы понятий).

Начиная с 60-х годов,отмечается экспоненциальный рост затрат на программное обеспечение. Эти годы характеризуются существенным усложнением программистской деятельности, что обусловлено:

- растущей сложностью и масштабностью задач, их комплексным характером;
- возросшими требованиями к качеству программного обеспечения и желанием рассматривать программу как товар;
- развитием аппаратных возможностей и усложнением архитектуры вычислительных систем;
- резким увеличением числа и круга пользователей программ.

Эти обстоятельства заставили программистов искать другие способы производства программ. Один из них был подсказан историей развития техники - индустриализацией производства. Возникла потребность перейти от индивидуального творчества к организованной коллективной программистской деятельности. Конец 60-х - начало 70-х годов характеризуются осознанием возможности применения ин-

женерных методов в программировании. Появилось понятие технологии программирования, толкуемое как совокупность инженерных знаний, призванных обеспечить упорядоченную и целенаправленную деятельность по созданию программ.

Одно из ключевых положений инженерной точки зрения - это возведение программы в статус промышленного изделия. Смена парадигмы повлекла за собой появление новых проблем, в частности, проблемы обеспечения высокого потребительского и технологического качества программ. Из объекта творчества отдельных личностей программа превратилась в праксеологический объект - результат планомерной, организованной программистской деятельности, объект эксплуатации, модификации и сопровождения. "Программные изделия - это не просто программы для ЭВМ,а продукт тщательного планирования и целенаправленной разработки, сопровождаемый четкой документацией, прошедший все необходимые испытания, описанный в соответствующих научно-технических публикациях, обеспеченный обученным персоналом, и размноженный в требуемом количестве экземпляров, обслуживаемый и контролируемый поставщиком по заранее продуманному плану" [2]. Таким образом, праксеологический статус программы в инженерном подходе проявляется в строгой регламентации деятельности. Заслуга инженерной точки зрения в том, что она заставила обратить внимание на сам процесс разработки, постулируя его первичность по отношению к желаемым потребительским характеристикам программы в виде принципа их обеспечения по построению [3,7]. В последнее время предприняты усилия по уточнению показателей качества программ , ведется активный поиск методов измерений соответствующих характеристик [4,5].

В настоящее время инженерный подход позволяет зачастую успешно справляться с решением проблем обеспечения потребительских свойств программ,планирования сроков и оценки трудозатрат,организации и управления производством. Все это делает инженерный подход наиболее влиятельной тенденцией в современной практике программирования. Но тем не менее индустриализация программирования пока не дала тех результатов, которые от нее ожидали в главном - резком повышении производительности труда программистов. И если переход от ремесленного производства программ к индустриальному связывали с первым кризисом в программировании, то сейчас стали говорить о втором кризисе.

Так или иначе вновь остро встал вопрос о поиске новых точек зрения на программистскую деятельность и ее результат - программу. Одной из таких точек зрения, могущих помочь программированию преодолеть возникшие препятствия, является точка зрения, основную идею которой можно сформулировать так [6,7]: программирование - одна из самых трудных отраслей прикладной математики. В соответствии с этим тезисом, если говорить об о с н о в а н и я х программирования (как научной дисциплины), то поскольку программа - это конструктивный объект (алгоритмической природы), они (основания) по необходимости должны быть л о г и к о - м а т е м а т и ч е с к и м и [8,9]. О правомерности данного вывода говорят такие тенденции в программировании, как функциональное, трансформационное, индуктивное и дедуктивное программирования, абстрактные типы данных, потоковые схемы и т.д.

Статья посвящена анализу технологических аспектов некоторых концепций современного програмирования и представляет собой аналитический обзор концепций, примыкающих к инженерному подходу.

§ 2. Предмет технологии программирования

В технических науках под т е х н о л о г и е й чаще всего понимают определенную последовательность действий (операций, работ, мероприятий), направленную на достижение определенной цели. Другой смысл этого термина - определенный раздел инженерных знаний. Системы с фиксированной технологией (понимаемой в первом смысле) называются в литературе т е х н о л о г и ч е с к и м и. Такие системы способны решать задачи по уже известным схемам. Другой класс, так называемых о р г а н и з а ц и о н н ы х с и с т е м, ориентирован на задачи, схемы решения которых неизвестны [10]. В реальной программистской коллективной деятельности присутствуют как технологические, так и организационные аспекты. Разумное сочетание этих аспектов - одна из главных проблем, стоящих перед каждым руководителем проекта. Научное решение этой проблемы - задача программирования, а конкретно - ее специального раздела т е х н о л о г и и п р о г р а м м и р о в а н и я. Заметим, что, как и выше, данный термин также употребляется в двух смыслах:

- как раздел программистских знаний, посвященный вопросам применения методологических и теоретических выводов и положений

на практике (технология программирования в широком смысле слова)[*];

- как некая схема организации деятельности, способ производства программ (технология программирования в узком смысле слова).

Помимо этого, употребляются термины: "технология проекта" и "технологические линии" [2,3,II,I2]. Первый понимается как реализация некоторой технологии программирования применительно к созданию конкретной программной системы. Второй означает конкретизацию технологии программирования в операционной обстановке конкретного коллектива. Таким образом, имеется несколько уровней конкретизации технологических программистских знаний.

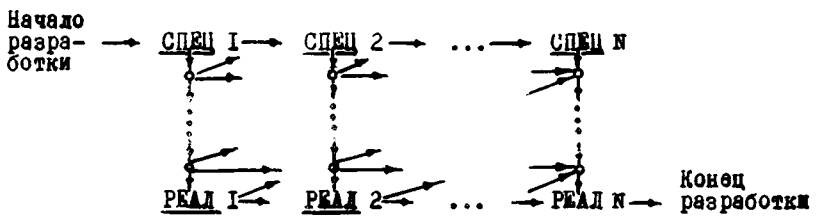
Конструирование программы традиционно начинается с уточнения исходных требований. Совокупность этих требований составляет спецификацию программы. Основа спецификации - описание того, что должна "уметь" делать программа. Обычно это описание ее функций. Существенным элементом является также описание того, какой хотел бы видеть программу ее пользователь (надежной, точной, информативной и т.п. [4,5]). Спецификация программы обычно пишется в содержательных терминах, которые зачастую оказываются неточными и двусмысленными. Выход ищут либо в стандартизации документов, которые нужно заполнять на этом этапе (технические задания, протоколы согласований и т.п.), либо в применении полуформальных методов (метод HIPO-диаграмм, метод формализованных технических заданий) [I4]. В последнее время стали говорить о формальных языках спецификаций.

Пусть у нас фиксирован некоторый (возможно, и неформальный) язык спецификаций. Тем самым определено множество возможных спецификаций программ - S-пространство. Программирование - это перевод спецификаций в программный код, который пишется в терминах некоторого языка программирования. Этот язык определяет свое пространство реализаций - R-пространство, отражающее специфику архитектуры некоторой вычислительной системы (возможно, и абстрактной). Каждое из этих пространств (S и R) имеет свои наборы понятий. Эти пространства совершенно различной природы и базируются на различных понятиях. Пространство S - это пространство задач, в нем главенствует человеческий фактор, тогда как набор понятий и аппарат пространства R отражают машинный фактор в програм-

[*] О структуре программистских знаний смотри [8].

мировании. Традиционно программирование, идя от конкретного, занималось, главным образом, разработкой, исследованием и развитием аппарата пространства реализации: любые новые идеи, методы, концепции преподносились через языки программирования. Формулирование же независимого понятийного аппарата S-пространств практически не уделялось никакого внимания: S-пространство рассматривалось как абстракция R-пространства. Разработка формальных языков спецификаций началась всего несколько лет назад.

Диаграмму состояний разработки можно весьма условно изобразить в следующем виде:



Здесь СПЕЦ i $\in$ S, РЕАЛ i $\in$ R (i$\in$ { 0,1,...,N}).

Конкретному процессу разработки соответствует некоторый путь в этой диаграмме. Стрелки по вертикали означают конкретизацию описанной программы, по горизонтали - преобразование описаний в рамках понятий одного уровня, по диагонали - изменение описаний в рамках понятий более высокого уровня. Идеальной является ситуация, когда все уточнения и модификации осуществляются на самом верхнем уровне, т.е. на уровне спецификаций.

В терминах S, R-пространств можно выявить и охарактеризовать различные типы технологий программирования. Если, например, пространства S и R фиксированы, то соответствующая технология программирования (УТП) может рассматриваться как частичное отображение "технология программирования": S(УТП) $\rightarrow$ R(УТП), где S(УТП) и R(УТП) есть S и R-пространства соответственно. Такие технологии программирования, как правило, являются узкоспециализированными и будут называться (S, R)-т е х н о л о г и я м и. Примерами (S, R)-технологий являются технологии, базирующиеся на пакетах прикладных программ.

В рассматриваемой технологии программирования может быть фиксировано только R-пространство ((-, R)-т е х н о л о г и я. Примерами (-, R)-технологий могут служить языки программирования вме-

сте с некоторой фиксированной дисциплиной программирования (таковыми, кстати, является большинство существующих технологий,см.§ 3).

В других случаях в технологии программирования может быть фиксированным только S-пространство ((S ,-)-т е х н о л о г и я). В этом случае технология выступает как конструктивный метод решения некоторого круга задач (метод решения задач линейного программирования, метод решения задач распознавания образов и т.п.). Решения задач здесь обычно фиксируются в форме весьма абстрактных алгоритмических схем, безотносительно к тому, к а к и на ч э м эти схемы будут реализованы.

Наконец, могут существовать технологии, для которых не фиксированы ни S, ни R-пространство, но указано,как можно было бы для некоторой проблемной области построить одно из них или оба и как осуществить отображение из S-пространства в R-пространство ((-,-)-технология). (-,-)-технология выступает скорее как некоторая к о н ц е п ц и я (подход, дисциплина) программирования. Примерами таких технологий являются структурное, модульное, функциональное, трансформационное, логическое программирования и др. (см. § 4). Целью развития (-,-)-технологии является создание, расширение и модификация семейства узкоспециализированных технологий, покрывающих как можно более широкий спектр областей применения. При создании такого семейства нужно помнить, что, несмотря на различие создаваемых программ (операционных систем , трансляторов,стандартных процедур , ППП, СУБД и т.п.), процессы их создания имеют много общего. Эта общность заключается в том, что на всех этапах конструирования используются одни и те же принципы и механизмы мышления. Поэтому развитие концептуальной технологии программирования должно происходить не только в сторону специализации, но и в сторону обобщения.

Что же должно составлять основу технологии программирования, что способно сделать ее концептуально единой и непротиворечивой? Чтобы ответить на этот вопрос, рассмотрим проблему понимаемости программ.

Программа - это знак,· лингвистическая конструкция, но знак, несущий некоторый смысл. До определенного момента развития программирования существующие синтаксические свойства программ и их неформальное "смысловое" толкование вполне удовлетворяли нужды практики. Однако неточности и противоречия, которые сопровождали процессы конструирования и использования программ заставили

программистов обратить самое серьезное внимание на семантическую сторону программирования. Оказалось, что и само развитие синтаксиса языков программирования существенно зависит от успехов в семантических исследованиях. И это не удивительно, поскольку при написании программы синтаксический уровень ее восприятия подчинен семантическому, а потому столь важно уметь правильно превращать семантические аспекты в синтаксические. В идентификации, изучении и использовании механизмов подобного перехода заключается основная задача технологии программирования.

Обычное представление о смысле знаков связано с понятием д е н о т а т а, т.е. того объекта(ов), который данный знак обозначает. Таким образом, чтобы понимать программу, нужно знать ее денотат. Программы, являясь синтаксически правильными, могут иметь много денотатов, или вообще не иметь их. Поэтому предпочтительнее не столько знание денотата программы, сколько знание п р о ц е д у р ы, с помощью которой можно построить (найти, выделить) этот денотат среди объектов того же типа. При таком интенсиональном подходе семантика программ - это прежде всего с о о т в е т с т в и е, сопоставляющее каждой программе ее денотат вместе с конструктивными указаниями, задающими это соответствие.

Рассмотрим процесс создания программы как процесс понимания того, ч т о и к а к конструировать. На уровне "что" осмысливаются спецификации программы в контексте тех ограничений, которые будут сопровождать процесс конструирования. Уровень "как" - это уровень аргументации того, что поставленная задача при заданных ограничениях разрешима, и построение на основе этой аргументации самой программы.

Осмысливание спецификаций - это прежде всего уяснение тех ситуаций, в которых будет использоваться программа. Оно позволяет образовать некоторую теорию, а чаще всего просто выдвинуть гипотезу относительно смысла будущей программы. Понимание смысла программы проявляется в дальнейших действиях по ее конструированию и закрепляется в "материале" и структуре этой программы. Таким образом, т е х н о л о г и ч е с к а я с е м а н т и к а*) отражает только те ситуации использования программы, которые принимались во внимание при ее создании.

---

*) Возможна и другая, п р а г м а т и ч е с к а я с е м а н т и к а программы, связанная с ее использованием, а не с конструированием.

Объяснение правил использования программы составляет содержание пользовательской документации и представляет собой не описание программы, а описание нормативов поведения, связанного с этим использованием. Множественность семантик порождается множественностью ситуаций использования. Поэтому невозможно дать конструктивное определение семантики программы, не зная той совокупности ситуаций, в которых она будет использована. Так, например, если программа транслируется в другую систему программирования, то необходимо задать ее операционную семантику (в терминах некоторой вычислительной системы). Если же программа нас интересует как функция, перерабатывающая входные данные в выходные, то мы нуждаемся в функциональной семантике, которую можно задать многими способами: выбор способа зависит от выбора языка, в котором представимо понятие функции (язык теории множеств, язык теории категорий, язык λ-исчисления и т.д.). Наконец, если данная программа используется с целью познать, научиться реализации некоторых типичных ситуаций или адаптировать программу к новым условиям, то необходима реализационная семантика. Возможны и другие семантики. При этом можно придерживаться либо содержательного ее определения, либо формального – все зависит от целей, которые преследуются.

Выдвигается следующий тезис: **концептуальную основу любой технологии составляют используемые в ней семантические представления программ.**

Из него непосредственно следует, что эффективность технологии программирования во многом определяется полнотой, адекватностью и простотой выражения семантических свойств програмы языковыми средствами, используемыми в ней.

Другой вывод из вышеприведенных рассуждений формулируется следующим образом: не существует универсальных технологий программирования, поскольку принятые в ней семантические представления могут оказаться совершенно неадекватными в некоторых случаях. Более того, не существует единого универсального семантического представления программ. Программа должна мыслиться (и существует) как некоторая системе таких представлений, каждое из которых может (и должно) иметь свой синтаксис.

По этой причине ведущийся сейчас спор о стилях программирования (см., например, [15]) нам представляется следствием методологических заблуждений.

§ 3. Основы инженерного технологического знания

Данный параграф содержит краткий аналитический обзор основных разделов традиционного технологического знания о программах и процессах их создания, целью которого является выявление основных тенденций развития этой области. Вся совокупность технологических знаний может быть условно разделена на три взаимообусловливающих части [8], относящихся соответственно к тому:

- что конструируется (архитектура программных изделий);
- как конструируется (технологический процесс, организация и управление разработкой);
- чем конструируется (инструментальные средства).

I. Основные понятия и принципы инженерного подхода. Нетривиальную программу нельзя написать за один прием. Поэтому программирование предпочтительнее рассматривать как сложный процесс последовательных преобразований, совершенствующих представление программы от некоторой исходной спецификации до кода инструментального языка. Как было уже сказано, технология программирования определяется выбором семантических представлений, которые и играют роль промежуточных форм. Особый интерес вызывает получение первоначального, базового представления.

К основным чертам инженерного подхода относятся:

а) концепция программного изделия, его отторженность от разработчика и улучшение потребительских качеств (товарность);

б) уменьшение роли творческого фактора в программировании, в смысле ориентации на коллективы рядовых специалистов;

в) коллективный характер труда и разделение труда;

г) систематизация деятельности программистов и тщательная регламентация процесса разработки программного изделия;

д) комплексная автоматизация технологического процесса;

е) стандартизация архитектуры программного изделия;

ж) обеспечение потребительских качеств по построению;

з) широкое использование ЭВМ в технологическом процессе.

Технология изготовления промышленных изделий существенно зависит от характера производства. Каждая программа уникальна, но

вместе с тем выработались определенные типы (классы) программных изделий (ОС, трансляторы, пакеты программ), которые с технологической точки зрения имеют много общего. Поэтому наряду с универсальными средствами, характерными для единичного производства, в программировании появились специализированные технологические комплексы, ориентированные на определенный тип программного изделия. Однако программирование является единичным производством в том смысле, что здесь разработка даже N-го изделия одного типа не является простым воспроизведением.

Наряду с очевидной (проблемной) общностью производства программных и материальных изделий имеются и отличия. В материальном производстве конструкторская и технологическая стадии четко разделяются формальным представлением конструкции изделия в виде чертежа, который содержит только "что" без "как". В программировании аналогичную роль играет спецификация. Однако в рамках инженерного подхода не были (да и не могли быть) найдены формальные средства спецификации программных изделий, которые не затрагивали бы одновременно вопросов "как" это сделать. Поэтому зачастую применяются неформальные описания на естественном языке. По мере усложнения программных изделий такие спецификации объемом в сотни страниц начинают страдать неточностью, противоречивостью и, будучи по сложности сравнимы с самой программой, сводят на нет пользу от спецификаций.

В современном программировании в качестве специализированной спецификации программных изделий выступает протопрограмма или программа на некотором достаточно высоком уровне абстракции. Архитектура такой протопрограммы представляется в виде некоторой абстрактной машины, которая является общим семантическим представлением программ в рассматриваемых технологиях программирования.

Абстрактная машина - это четверка (F, D, C, U), где F - функциональная структура (операционный блок машины), D - структура данных (память машины), C - логическая структура (блок управления машины), U - канал ее связи с внешней средой.

В зависимости от первичности соответствующей структурной компоненты машины, имеются три метода спецификации, которые дают известные три подхода к проектированию: от функций (функциональный подход); от данных (информационный подход); от управления.

Приводимые ниже сведения основаны на анализе пяти наиболее популярных зарубежных технологий программирования: технологии

структурного программирования корпорации IBM [I6], методологии SADT Д.Росса [I7], структурно-композиционного проектирования (Structured/Composite Design или S/CD) [3], методов Джексона [I8] и Варнье [I9], а также отечественной Р-технологии программирования [II-I3] и некоторых других более узких разработок.

2. Архитектура программного изделия. Архитектура программного изделия представляется через аппарат пространства реализации, его описание есть описание свойств и структуры того материала и тех конструкций, из которых делаются, в конечном счете, программы. Этот аппарат также состоит из указанной ранее четверки (F,D,C,U), т.е. имеется в виду аппарат процедур, аппарат данных, аппарат управления и аппарат общения с программным изделием соответственно. Основные тенденции развития аппарата пространства реализации заключаются в следующем:

а) стандартизация, начиная с малых конструкций языков программирования до крупных блоков [20] вплоть до типовых программных изделий, а также выявление типовых образцов программирования [2I];

б) разработка механизмов конструирования новых типов конструкций из имеющихся;

в) поддержка нетрадиционных конструкций;

г) совмещение человеческого и машинного факторов в одном представлении.

Развитие F-аппарата происходит в рамках методов модульного программирования. Первоначальный мотив модульности - декомпозиция сложности и выделение общих частей алгоритма - привел к появлению программной единицы, названной п р о ц е д у р о й. Другой мотив - независимость - к понятию отдельно транслируемой подпрограммы. Постепенно выявилось главное значение модуля как единицы программного проекта; были осознаны не только архитектурные аспекты модульности, но и организационно-технологические. С усилением мотивации модульности, постепенно усиливается аксиома модульности. Кроме того, выяснилась нетривиальность модуляризации программного изделия, заключающаяся в том, что различные способы модульной декомпозиции дают различный эффект и, в частности, было обнаружено влияние модульной структуры на характеристики качества программного изделия.

Одним из первых это осознал Д.Парнас [22], предложивший новый критерий - п р и н ц и п у п р я т ы в а н и я и н ф о р м а ц и и, когда каждый модуль проектируется таким образом, чтобы

проектные решения были скрыты и максимально локализованы. Впоследствии развитый многими авторами этот принцип получил название *принципа свертки* или *инкапсуляции* (см.[23]). Развитие модульного программирования с характерной для него иерархической структурой программного изделия потребовало точного понимания смысла иерархии. Установление того, что иерархия модулей порождается различными отношениями частичного порядка между ними, такими как "использует", "вызывает", "может быть доступно из", "является частью", "распределяет область памяти для", "дает работу" и т.п. и изучение их особенностей существенно обогатило понимание механизмов модульности.

В рамках метода композиционно-структурного проектирования (S/CD-технология [3]) Майерс и Константайн на основе введенных понятий "прочности" и "сцепления" модулей дали классификацию типов модульности и показали их влияние на свойства программ. Приведенная в [3] шкала прочности модулей состоит из семи видов. Самой высшей является функциональная прочность. В этой шкале критерию инкапсуляции соответствует так называемая информационная прочность. Сцепление модулей понимается как мера взаимозависимости модулей по данным, т.е. характеризует межмодульный интерфейс. Шкала видов сцепления содержит шесть наименований, из которых самым жестким сцеплением является прямая ссылка из одного модуля на содержимое другого, а самым слабым - сцепление по данным, когда связь осуществляется через передачу параметров, которые являются простыми типами данных.

Понятно, что наиболее технологичны высокопрочные модули со слабым сцеплением между ними. Дальнейшее развитие модульности приводит к понятию модуля как замкнутой программной единицы, в которую упакованы данные и/или функции, доступные вне модуля только по явному запросу на обслуживание. Такие модули, следуя Б.Лискову [24], стали называть *кластерами*.

Модульность в архитектуре программного изделия в своем развитии вышла за пределы программы и распространилась на программирование "в большом". В результате существенно выросла мощность строительных блоков, из которых собираются программные системы. Разработка адекватного Ф-аппарата для крупноблочного программирования, выразилась в концепции *пакетов программ*. Для прикладных задач, у которых проблемная специфика отражается, главным образом, в методах обработки данных, характерно относи-

тельное постоянство F-компоненты (модульного наполнения системы обработки данных). Чтобы воспользоваться этим обстоятельством для повышения эффективности процесса программирования задач, активно разрабатывается аппарат пакетов программ - от библиотек стандартных модулей до систем автоматического синтеза программ из готовых подпрограмм.

Чисто прагматически, пакет программ можно рассматривать как полуфабрикат, из которого можно без особых усилий получить любую рабочую программу из класса программ,определяемого фиксацией множества модулей. Это позволяет конструировать программное изделие на более высоком уровне и порождать системы с возможностями, намного превышающими возможности заново написанных программ при одинаковых трудозатратах.

Таким образом, в архитектуре изделия происходит укрупнение единиц модульности (процедура, подпрограмма, программа, пакет программ, прикладная программная система), повышающая степень автоматизации конструирования программного изделия. Оно отражает общую тенденцию интеграции современных систем обработки информации, которая является следствием комплексности решаемых с помощью ЭВМ практических задач.

Аппарат данных осуществляет процесс отображения обрабатываемой информации в структуры машинной памяти. Технологическое значение этого аппарата определяется возможностью отражения семантики в синтаксисе, которую представляет структурирование. Удачная структура данных существенно упрощает алгоритм, а в некоторых случаях полностью предопределяет его.

Имеется два уровня аппарата данных, соответствующих внутренней и внешней памяти машины. Структуры данных внутренней (оперативной) памяти и соответствующий аппарат формировался в рамках языков программирования высокого уровня и отражает технологический уровень разработки программ (или программирования "в малом"). На уровне программных систем рассматриваются структуры данных внешней памяти ЭВМ. Здесь развитие аппарата данных шло от простейших файловых систем до механизмов баз данных сложной структуры (СУБД).

Несмотря на то, что развитие структур данных на этих двух уровнях привело к выделению их в самостоятельные дисциплины, можно заметить концептуальное единство этих дисциплин,проявившееся в:

а) выделении стандартных структур данных, позволяющих адекватное и эффективное информационное моделирование реальных объектов и процессов;

б) обобщении вышеуказанного процесса в концепции типа данных, в которой тип, наряду с некоторым типовым проектным решением, дающим содержательное понимание, смысл объектов, моделируемых данными, определяет также структуру хранения и способы доступа к информации [23];

в) разработке механизмов конструирования новых, более сложных типов данных из имеющихся (определяемых типов и конструкторов типов);

г) тенденции к разделению логической спецификации данных от конкретной реализации, приведшей к концепции абстрактного типа данных.

Вывод, к которому приходит технология программирования в отношении данных заключается в том, что структуры данных должны конструироваться из определенного множества примитивных типов с помощью явно выраженных ограниченных средств конструирования. Понятие абстрактного типа данных и создаваемая математическая теория этого понятия, обеспечивающая формальный подход к спецификации программ, явилась закономерным этапом развития аппаратов данных и процедур. Эта концепция объединяет модульное программирование и программирование "от данных", поставив их на прочный методологический и теоретический фундамент, имеющий явную логико-математическую природу.

Аппарат управления определяет способы комбинирования действий для построения более сложных действий и программ. Возрастающая сложность действий, порождаемых в вычислительных системах, приводит к необходимости обуздания этой сложности путем структуризации порождающего механизма.

С другой стороны, программа определяет некоторый класс вычислений, а текст программы представляет собой общее статическое описание вычислений этого класса. Аппарат управления позволяет в "статистическом" тексте программы задать динамику ее выполнения.

Основой механизма управления являются так называемые у п р а в л я ю щ и е с т р у к т у р ы. Их можно классифицировать следующим образом [25]:

- безусловные управляющие структуры (не зависят от состояния данных вычислительного процесса): последовательность, безусловная передача управления, бесконечный цикл fork-join и т.д.;

- условные управляющие структуры (последовательность действий зависит от состояния данных): условные операторы, циклы, спусковые функции и т.д.;

– потоковые управляющие структуры, неявно задающие информа – цию об управлении в структурах данных (т.е. вычисления, управляемые структурами данных).

Известно, что для построения любой программы достаточно оператора перехода go to и механизма подпрограмм с возможностью либо их рекурсивного вызова, либо последовательной структуры (це – почки) и цикла. Первые языки программирования высокого уровня включали наряду с минимально необходимыми управляющими структурами некоторый дополнительный набор для удобства программирования. Теорема Бема-Якопини-Глушкова [44] выявила достаточность трех управляющих структур: цепочки, ветвления по условию и цикла.

Предположение Дейкстры о том, что понимаемость программы тесно связана с ее управляющей структурой, имело серьезные технологические следствия. Стало ясно, что для того чтобы интеллектуально контролировать вычисления, исходя из статического текста программы, следует довольствоваться наиболее систематизированными способами организации следования, гарантирующими соответствие продви – жения вычислений с продвижением по тексту. Для решения технологической проблемы убедительной демонстрации правильности программы главной характеристикой управляющих структур становится не "удобство", а наличие свойств, помогающих в решении этой задачи. Базовые структуры Бема-Якопини-Глушкова, помимо полноты, обладают свойствами, позволяющими, рассуждая "статически" о программе, доказывать правильность относящихся к ней утверждений. Хоар и Дейкстра [26] показали, что свойства управляющих структур, выводимые из интуитивного определения базовых структур, могут быть основаны на строгом определении и построили аксиоматику управляющих структур.

Все эти обстоятельства привели к концепции с т р у к т у р н о г о п р о г р а м м и р о в а н и я, лежащей в основе мно – гих современных технологий программирования.

Еще один важный метод организации управления выполнением программ дают потоковые управляющие структуры, которые определяют перенос управляющей информации в данные, т.е. перенос динамики выполнения в статику структур данных. Такие вычисления управляются структурами данных. Этот процесс лежит в основе методологии п р о г р а м м и р о в а н и я о т д а н н ы х и широко используется в технологии программирования (управление с помощью таблиц и диаграмм переходов) [12,13,18,19].

Дальнейшее развитие управляющих структур определяется концепцией абстрактных типов данных, которая оказалась весьма полезной и для аппарата управления. Абстракция управления привела наряду с рассмотренным выше набором универсальных полезных конструкций, к механизму конструирования управляющих структур. Тип управления может быть о п р е д е л е н подобно типам данных с использованием параметров ранее сконструированных типов и управляющих примитивов. Благодаря этому возможно конструирование структур, необходимых для адекватного представления алгоритма задачи, предваряя конкретное программирование абстрактной спецификацией задачи. Кроме того, абстрактные управляющие структуры способствуют согласованию структуры программы со структурой вычислительной системы, а для вычислительной системы с перестраиваемой динамической структурой типы управления специфицируют необходимую виртуальную конфигурацию [25,27].

Дальнейший рост мощности управляющих конструкций связывается с проблемно-ориентированными "схемами счета", которые практикуются в пакетах прикладных программ.

Широкий спектр средств общения с программными изделиями различных категорий пользователей составляет U-аппарат.

К этим средствам относятся механизмы вызова программных модулей и передачи параметров, служебные форматизированные управляющие карты, входные языки пакетов программ, языки заданий (или, иначе, к о м а н д) операционных и мониторных систем и т.п. Основная цель языков общения - это обеспечение такой связи между вычислительной системой и ее пользователями, при которой каждый пользователь может решить свою задачу на семантическом уровне, соответствующем этой задаче.

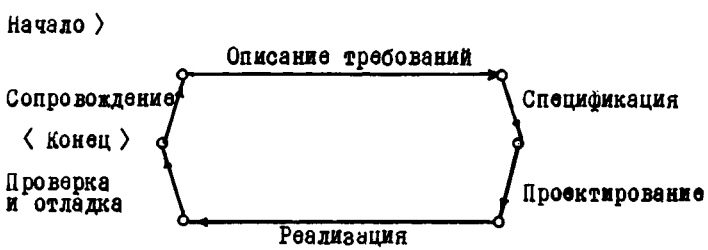
В настоящее время отмечается быстрый рост значения этих средств [34]. Среди языков общения, которые делятся на два класса: пакетных и интерактивных, наблюдается такое же разнообразие, как и среди языков программирования.

Разрабатываются как чисто декларативные языки общения (для систем генерации и автоматического синтеза), так и процедурные. При этом имеется тенденция увеличения мощности процедурных языков и приближения их возможностей к возможностям языков программирования высокого уровня. Надо отметить, что грани между языками общения и программирования постепенно стираются, критерии, ограничивающие сферы этих языков, практически отсутствуют. Языки об -

щения постепенно превращаются в средства сверхвысокого уровня для конструирования решений комплексных задач, поскольку они позволяют интегрировать целые пакеты программ.

Для интерактивных средств общения характерна тенденция к повышению "локального интеллекта" терминала [29,30], стремление к компактности, синтаксической гибкости, удобству языка. Общей тенденцией развития языков общения является также стремление к независимости языка от вычислительной системы. По-видимому, не за горами создание единого языка общения с ЭВМ, удовлетворяющего всех пользователей в любой операционной обстановке, построенного по многоуровневому принципу: непроцедурный уровень, процедурный прикладной уровень, системный уровень. Такой язык должен допускать встраивание входных языков пакетов программ, построенных по единому стандарту.

3. Процесс разработки программного изделия и средства представления. В инженерном подходе технологический процесс является предметом изучения и последующей его регламентации. Верхний уровень описания технологического процесса представляет собой, по существу, описание жизненного цикла программного изделия, типичный вид которого представлен ниже диаграммой



Дальнейшая детализация процесса выделяет внутри этих фаз отдельные технологические операции. Выделение технологических стадий управляется, на наш взгляд, двумя критериями:

1) наличием соответствующего устойчивого промежуточного представления, отражающего тот или иной семантический аспект создаваемого программного изделия;

2) минимизацией (фильтрацией) "шума трансляции". "Шум" - искажение смысла, потери информации при переходе от одного семантического представления к другому, является основным источником ошибок конструирования программного изделия [3].

Таким образом, технология регламентирует создание программного изделия в некоторой логической последовательности шагов, позволяющей знать текущее состояние работы и оценивать ее объем до момента завершения. Описание частично упорядоченного множества действий технологического процесса вместе с множеством форм представления результатов действий, стандартов и соглашений называется *технологической схемой*. Имеются попытки программирования деятельности в виде формализованного языка описания технологических схем (язык АКТ [31]), но целесообразность этого сомнительна.

Нас интересуют прежде всего конструктивные стороны технологического процесса. Поэтому основное внимание мы обратим на начальные фазы разработки.

Этапы описаний требований и спецификации проекта - одни из самых плохо изученных, наименее разработанных этапов. Описание требований - это представление задачи заказчиком, выданное на проблемно-ориентированном естественном языке. Полнота описаний, устранение двусмысленностей и неточностей связывается с формализацией. Пока имеются лишь несколько примеров формализованных языков, например, PSL [32], RSL[33], PRL[34]. Этап спецификации - это логический анализ решаемой задачи, конечной целью которой является абстрактное выражение исходных реальных требований в форме, понятной проектировщикам, спецификация того, *что* нужно сделать, чтобы удовлетворить требованиям, и служит начальным представлением проекта программного изделия. Выбор семантического представления в качестве спецификации определяется выбором метода проектирования: функциональным, от данных или от управления (встречаются и смешанные методы). Отметим, что проблема спецификации является главной в современных технологиях программирования.

В функциональном методе в качестве спецификации выступает *функциональная архитектура* программного изделия, представляющая собой иерархическое функциональное описание последнего. В качестве средств представления используются иерархические структурные схемы и различные неформальные графические языки диаграмм (HIPO [35], диаграммы Росса [17], Варнье [19] и др.), которые одинаково годятся как для описания функциональной архитектуры, так и для описания модульной архитектуры программ.

Для подхода "от данных" (Джексон, Варнье, Р-технология) характерен вывод управляющей структуры программы из структуры дан-

ных и даже отождествление этих структур. Поэтому в качестве спецификации выступают представления структур и потоков данных на формализованных, чаще всего графических языках. Здесь следует отметить язык Р-графов, который позволил ввести в программирование понятие чертежа в буквальном смысле. При подходе "от управления" проектирование начинается с описания логики процесса, описания поведения программы на некотором процедурном языке сверхвысокого уровня, языке блок-схем или псевдокоде [36,37].

Этап проектирования программного изделия осуществляет переход от спецификации задачи к описанию процесса обработки данных на концептуальном уровне, т.е. к описанию алгоритмического решения, которое в дальнейшем будет конкретизировано для некоторой вычислительной системы и закодировано в соответствующем языке программирования.

Детализация шагов проектирования определяется с т р а т е г и е й проектирования, которая задает направление этого процесса. Известны две основные стратегии: горизонтальное слоение и вертикальное слоение. Горизонтальное слоение имеет, в свою очередь, две разновидности: аналитический (нисходящий, сверху-вниз) метод и синтетический (восходящий, снизу-вверх), которые, в свою очередь, имеют модификации в виде стратегий "снаружи-вовнутрь" и "изнутри-наружу". Описание техники горизонтального слоения можно найти в работах [2,3,7,12,26,37,38]. Техника вертикального слоения исследована в [31]. На практике, как правило, применяются смешанные стратегии. Смешивание стратегий заключается, с одной стороны, в совмещении подходов на одной стадии, а с другой - в использовании различных стратегий на разных стадиях технологического процесса. Разумный набор стратегий составляет искусство проектирования.

Итак, типичный процесс проектирования функциональным методом выглядит следующим образом:

I) Иерархическое разбиение исходной задачи на подзадачи сверху-вниз (отметим здесь интересный метод STS-декомпозиции [3] на наиболее независимые подзадачи). Каждый шаг разбиения описывается на формализованном графическом языке набором диаграмм для функций и для данных. Каждая задача описывается как функция, либо как пара ⟨предикат,функция⟩, а входы и выходы функций специфицируются на соответствующем уровне абстракции.

2) Отображение функциональной архитектуры в модульную (системную) на основе формальных критериев модуляризации. Специфика - ция потока данных на языке описания межмодульного интерфейса.

3) Разработка модулей "снаружи-вовнутрь" (от внешней архитектуры к внутренней). Процедурное структурное описание высокого уровня в виде блок-схемы или псевдокода. Последовательное уточнение с переходом от псевдокода к коду.

Процесс разработки программного изделия "от данных" выглядит так:

1) полуформальное описание структур данных и характера обработки данных;

2) формальная спецификация логической структуры данных;

3) вывод структуры программы из структуры данных, путем, например, доопределения спецификации данных [12];

4) представление программы на псевдокоде;

5) систематическое уточнение псевдокода сверху-вниз до кода программы на базовом языке.

4. Организация коллективной разработки программного изделия и управление проектированием. Инженерный подход принес в проектирование принцип организации разработки программного изделия в форме п р о е к т о в [2]. Организационные мероприятия и затраты на управление являются естественной платой за коллективный метод работы, распараллеливание разработки и разделение труда. Управление процессом проектирования и изготовления изделия предполагает регламентацию взаимодействия разработчиков, контроль за выполнением требований технологического процесса, планирование и учет деятельности разработчиков, анализ хода разработки и эффективное влияние на него, маневрирование людскими и материальными ресурсами с целью ускорения сдачи изделия в эксплуатацию.

Основной проблемой организации проекта является проблема "к о м м у н и к а ц и о н н о г о ш у м а", который означает искажение или потерю информации при передаче ее от одного человека (группы) другому человеку (группе) посредством некоторого неформального семантического представления. Некоторые специалисты считают, что организационная структура коллектива программистов является главным, определяющим фактором обеспечения таких атрибутов качества, как правильность, надежность [3,39]. В качестве критерия выбора правильной организационной структуры предлагается "закон" Конвея [40], который мы сформулируем так: гомоморфность

структур разрабатываемой и разрабатывающей системы уменьшает коммуникационный шум.

Таким образом, управление производством программного изделия становится поиском эффективных организационных структур и методов эффективной фильтрации шума коммуникаций в течение всего цикла его разработки. Вывод: для любой технологии программирования необходимо найти организационную структуру коллективной работы, согласованную с данной технологией. К проблемам управления разработкой программного изделия следует отнести тот факт, что процесс управления происходит в условиях гораздо большей неопределенности, чем в материальном производстве.

Структура технологической цепочки - конвейера с пооперационной специализацией приводит к высокому уровню шумов, поскольку при этом шумы трансляции и коммуникации суммируются. Внедрение бригадных методов с неоднородным составом бригад и функциональной специализацией [42] , широко практикуемых за рубежом, затруднено из-за неодинаковой престижности ролей в бригаде, иерархии, "душащей всякую инициативу", и расчета на недюжинные способности главного программиста. По-видимому, более практична однородная структура коллектива разработчиков, предусматриваемая, например, Р-технологией. В ней, в соответствии с "законом" Конвея, в качестве объектов разделения труда программистов и групп рассматриваются элементы и составные части архитектуры программного изделия.

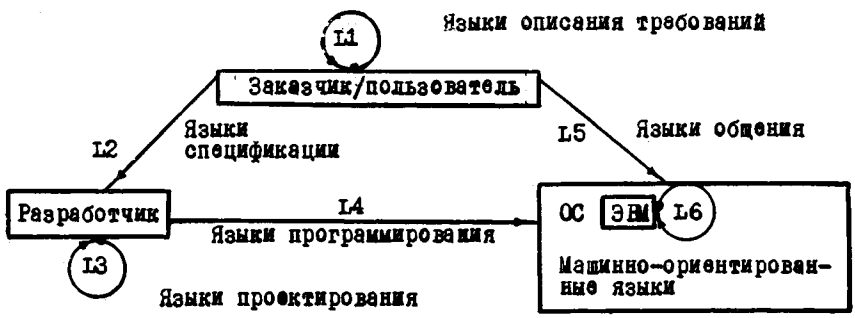
Создание программного изделия протекает в рамках тщательно разработанного плана, который можно рассматривать как отображение технологической схемы на трудовые ресурсы и время. Чаще всего план представляется в виде сетевого графика. Некоторые технологии программирования предусматривают автоматическое формирование сетевого графика работ по исходной информации, содержащейся в интегрированной базе данных проекта.

Вопросы организации и управления на различных стадиях технологического процесса конструирования изделия весьма подробно освещены в книге [2]. В заключение мы лишь подчеркнем важную тенденцию в этой области: организацию машинного контроля и диспетчеризации работ при управлении проектом. Перекладывание организационной рутины на плечи ЭВМ становится возможным благодаря ведению всей базы данных проекта на магнитных носителях и оперативному доступу разработчиков к этой информации. Общедоступность базы данных проекта позволяет организовать эффективный контроль за выпол-

нением работ, облегчает коммуникации между исполнителями и уменьшает шумы.

5. Инструментальные средства. Индивидуальные "инструментальные сундучки", содержимое которых характеризовало квалификацию программиста, индустриальный подход заменил универсальными стандартными комплексами, которые характеризуют уже уровень технологии программирования. Инструментальное обеспечение представляет собой языковые и программные средства машинной поддержки и автоматизации технологического процесса создания программного изделия.

Языковые средства, во-первых, являются средствами коммуникации между различными субъектами рассматриваемой деятельности, а во-вторых - средствами фиксации семантических представлений программного изделия, используемых в данной технологии. Приведенная диаграмма коммуникаций дает спектр языков технологии программирования, состоящей из групп (L1-L6) языков, поскольку каждая комму-



никация осуществляется обычно через несколько промежуточных семантических представлений. Из анализа этой диаграммы можно сделать следующий вывод (который приводится и в [13]): языки программирования, которым до сих пор уделяется основное внимание в технологии программирования, решают лишь небольшую (примерно шестую) часть проблематики конструирования программного изделия.

Можно отметить два подхода к лингвистическому обеспечению технологии программирования: а) выделение отдельных специальных языков; б) построение языков "широкого спектра" [43], охватывающих, например, языки L2-L4.

Общей тенденцией развития языковых средств технологии программирования является повышение их эффективности (в смысле адек-

ватности, естественности и формальности описания соответствующих семантических представлений) и технологичности (в смысле более конструктивной поддержки архитектуры, технологических принципов, стремления к "встроенности" технологии в инструмент).

К программным средствам относятся различные программные системы, способные выполнять или облегчать отдельные технологические операции создания программного изделия. Инструментальные системы можно классифицировать следующим образом:

а) системы общего назначения;

б) системы поддержки отдельных технологических этапов;

в) системы поддержки некоторой технологии, ориентированной на обширный класс программных изделий;

г) системы автоматизации разработки определенных типов программных изделий.

В класс "а" можно отнести системы, формирующие некоторую операционную обстановку (обслуживающие компоненты операционных и мониторных систем, системы типа PRIMUS), компиляторы, макрогенераторы, редакторы, генераторы документов, СУБД, а также интегрированные системы разработки программ. Центральным элементом последних является база данных проекта, содержащая все данные, относящиеся к разработке: спецификации, исходные, объектные, загрузочные коды программ, тесты, документация, служебная информация и т.п. Интегрированные системы обеспечивают средства хранения и манипулирования всей этой информацией.

К системам класса "б" относятся системы анализа требований заказчика (на полноту, непротиворечивость), системы статического анализа программ, системы верификации, генераторы тестов, отладочные стенды, автоматизированные системы нормоконтроля, системы программирования и т.д.

Системы класса "в" называются технологическими комплексами. Они осуществляют интеграцию технологического процесса и инструментальных средств в единую логически взаимоувязанную систему машинно-ассистируемой разработки программного изделия (пример - РТК [II]). Технологические комплексы можно компоновать из средств классов "а" и "б".

Наиболее эффективны системы автоматизации конструирования программных изделий (класс "г"), которые являются конкретизациями технологических комплексов для конкретного типа изделия (системы построения трансляторов, системы конструирования пакетов программ

и т.п.). В состав таких комплексов обычно входят метасистемы, которые представляют собой общую, инвариантную часть серии программных изделий одного типа.

Литература

1. УОЗЕРБЕЛЛ Ч. Этюды для программистов.-М.:Мир,1982.-287 с.

2. ГАНТЕР Р. Методы управления проектированием программного обеспечения.-М.:Мир,1981. - 392 с.

3. МАЙЕРС Г. Надежность программного обеспечения. - М.: Мир, 1980. - 360 с.

4. ЛИПАЕВ В.В. Качество программного обеспечения. - М.: Финансы и статистика,1983. - 263 с.

5. БОЭМ Б. и др. Характеристики качества программного обеспечения. - М.:Мир, 1981. - 208 с.

6. ДЕЙКСТРА Э. Дисциплина программирования. - М.: Мир, 1978. - 275 с.

7. ТУРСКИЙ В. Методология программирования. - М.: Мир, 1981. - 264 с.

8. НЕПЕЙВОДА Н.Н., СВИРИДЕНКО Д.И. Программирование с логической точки зрения. - Новосибирск,1981. - 96 с. (Препринты/ Институт математики СО АН СССР: Т-1,Т-2).

9. СВИРИДЕНКО Д.И. О природе программирования. - В кн.: Математическое обеспечение вычислительных систем на микро-ЭВМ (Вычислительные системы,вып.96).Новосибирск,1983,с.51-74.

10. ПОСПЕЛОВ Г.С., ИРИКОВ В.А. Программно-целевое планирование и управление.- М.: Советское радио, 1976. - 240 с.

11. ВЕЛЬБИЦКИЙ И.В. Технологические линии производства программ. - Тез.докл. I Всесоюз.конф. "Технология программирования" , Киев, 1979, с.31-35.

12. ВЕЛЬБИЦКИЙ И.В., ХОДАКОВСКИЙ В.Н., ШОЛМОВ Л.И. Технологический комплекс производства программ на машинах ЕС ЭВМ и БЭСМ-6. - М.: Статистика, 1980. - 263 с.

13. ВЕЛЬБИЦКИЙ И.В. Безбумажная технология программирования в диалоговой среде. - УСиМ,1982,№ 6,с.29-37.

14. ГЛУШКОВ В.М., КАПИТОНОВА Ю.В., ЛЕТИЧЕВСКИЙ А.А. О применении метода формализованных технических заданий к проектированию программ обработки структур данных. - Программирование,1978, № 6, с.31-43.

15. BACKUS J. Can programming Be liberated from the Von Neumann Style? A functional style and its algebra of programs.-CACM, 1978,v.21,N 8,p.613-641.

16. STRUCTURED PROGRAMMING SERIES. Program design study.Vol.8. IBM Corp.,1974.- 210 p.

17. ROSS D., SCHOMAN K. Structured Analysis for Requirements Definition.- IEEE Trans.on Software Eng.,1977,v.3, N 1,p.6-15.

I8. JACKSON M. Principles of Program Design. - N.Y.: Academic Press. 1975.

I9. WARNIER J., FLANAGAN B. Entraiment de la construction des programmes d'informatique.V.1,2. Editions d'organizatins, Paris,1972,

20. WINOGRAD T. Reyond programming languages.- CACM,1979,v. 22,N 7,p.391-401.

2I. FLOYD R.W. The paradigms of programming. - CACM, 1979, v.22, N 8,p.455-460.

22. PARNAS D.L. On the criteria to be used in Decomposing System into Modules.- CACM,1972,v.15,N 12,p.1053-1058.

23. АГАФОНОВ В.Н. Типы и абстракция данных в языках программирования (обзор). - В кн.: Данные в языках программирования, М., I982, с.265-327.

24. Abstraction Mechanisms in CLU/Liskov B., Snyder A. a.o. - CACM,1977,v.20,N 8,p.564-576.

25. КОТОВ В.Е. Параллельное программирование с типами управления. - Кибернетика, I979,№ 3.

26. ДАЛ У., ДЕЙКСТРА Э., ХОАР К. Структурное программирование. - М.: Мир,I975. - 247 с.

27. LISKOV B., ZILLES S. Programming with Abstract Control Types.- SIGPLAN Notices,1974,v.9,N 5.

28. Command Language directions /Ed. by D.Beech. - IFIP,North-Holland,1979.

29. MILLER L.A., THOMAS J.C. Behavioral issues in the use of interactive systems. - Lest.Notes in Comp.Sci.,1976,p.193-216.

30. GILOI W.K. Interactive systems patterns and prospects. - Lect.Notes in Comp.Sci.,1976,v.49,p.144-161.

3I. ФУКСМАН А.Л. Технологические аспекты создания программных систем.-М.: Статистика,I979. - I84 с.

32. TEICHROEW D., HERSHEY E. PSL/PSA: A computer aided Technique for structured documentation and analysis of information processing systems. - IEEE Trans.on Software Eng.,1977,v.SE-3,N 1, p.41-48.

33. BILL T., BIXLER D., DYER M. An extendable approach to computer - aided software requirements engineering. - IEEE Trans. on Software Eng.,1977,v.SE-3,N 1,p.49-60.

34. DAVIS A.M. Formal techniques and automatic processing to ensure correctness in requirements specifications. - Proc.IEEE Conf. on Specification of Reliable Software, 1979, p.15-35.

35. JONES M.N. HIPO for developing specifications. - Datamation,1976,v.22,N 3,p.112-125.

36. CAINE S., GORDON E. PDL: A Tool for Software Design,Tutorial on Software design.,oct.12,1976. San-Franc., Calif.

37. ДЗЕРЖИНСКИЙ Ф.Я. Язык для проектирования структурированных программ. - В кн.: Алгоритмы и организация решения экономических задач, 1980, вып.14, с.83-95.

38. ХЬЮЗ Дж., МИЧТОМ Дж. Структурный подход к программированию. - М.: Мир, 1980. - 278 с.

39. WALKER M.G. A theory for software reliability.- Datamation,1978,v.24,N 9,p.211-214.

40. CONWAY M. How do committees invent?.- Datamation, Apr., 1968,v.14,N 4,p.28-31.

41. МИЛЛС Х. Программирование больших систем по принципу сверху-вниз. - В кн.: Средства отладки больших систем. М., 1977, с.41-56.

42. BAKER F.T. Chief Programmer Team Management of Production Programming.-IBM Systems Journal,1972,v.11,N 1,p.56-73.

43. BAUER F.L. TOWARDS a wide Spectrum Language to Support Program Specification and Program Development.- Lect.Notes in Comp. Sci.,1979,v.69,p.543-552.

44. BOHM C., JACOPINI G. Flow diagrams, turing machines and languages with only two formation rules.-CACM,1966,v.9,p.366-371.

Поступила в ред.-изд. отдел
27 февраля 1984 года