Б. Боэм, Дж. Браун, Х. Каспар, М. Липов,
Г. Мак-Леод, М. Мерит

ХАРАКТЕРИСТИКИ КАЧЕСТВА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Издательство «МИР»


ИЗДАТЕЛЬСТВО
МИР

TRW Series
on Software Technology
Volume 1

CHARACTERISTICS OF SOFTWARE QUALITY

Barry W. Boehm
John R. Brown
Hans Kaspar
Myron Lipow
Gordon J. MacLeod
and
Michael J. Merritt

TRW Systems and Energy, Inc.
Redondo Beach, California

North-Holland Publishing Company — Amsterdam
New York Oxford 1978

Б. Боэм, Дж. Браун, Х. Каспар, М. Липов,
Г. Мак-Леод, М. Мерит

ХАРАКТЕРИСТИКИ КАЧЕСТВА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Перевод с английского
канд. техн. наук Е. К. Масловского

Издательство «МИР»
Москва 1981

ББК 32.973
Б 72
УДК 681.3

Боэм Б., Браун Дж., Каспар Х. и др.
Б 72 Характеристики качества программного обеспечения/Пер. с англ. Е. К. Масловского.— М.: Мир, 1981 — 208 с., ил.

В книге рассматриваются проблемы и методы оценки качества программного обеспечения (ПО) сложных систем на различных этапах его разработки. Предлагается система показателей, позволяющая выявлять дефекты ПО на ранних стадиях проектирования систем. Излагаются вопросы автоматизации процедур оценки качества программных средств, даются практические рекомендации, касающиеся методики анализа свойств ПО.

Для системных и прикладных программистов, разработчиков АСУ.

$\text{X}\,\frac{30502\text{-}150}{041\,(01)\text{-}81}$ 150-81, ч. 1 1502000000 ББК 32.973

Редакция литературы по новой технике

От переводчика

В настоящее время создание высокопроизводительного программного обеспечения для цифровых вычислительных машин и автоматизированных систем управления является одной из наиболее важных задач развития науки и производства. Трудности реализации крупных программных комплексов связаны прежде всего с необходимостью координации усилий большого числа разработчиков в соответствии с целевым назначением программных средств и требованиями заказчика системы, в рамках которой эти средства должны использоваться. От того, насколько удачно построено программное обеспечение системы, зависит в итоге ее жизнеспособность. Однако в связи с тем, что лишь очень немногие существенные свойства программных средств могут быть непосредственно оценены количественными показателями, достаточно серьезных работ, посвященных проблеме качества больших программных комплексов, в печати до сих пор не появлялось.

Поэтому внимание специалистов, несомненно, должна привлечь подготовленная американской фирмой TRW серия книг по технологии программного обеспечения. Книга Боэма и др., русский перевод которой предлагается советскому читателю, первая в этой серии. Основная мысль, проводимая авторами книги, состоит в том, что значительной части трудностей, возникающих при внедрении проектов, можно избежать, если с самого начала создавать систему программного обеспечения в соответствии с определенной методологией. Эта методология должна учитывать непрерывность процесса внедрения, статичность целей разработчика, динамичность требований заказчика, необходимость обеспечения высокой надежности функционирования, удобства эксплуатации и гибкости системы при изменении нормати-

ных данных, информационных массивов и программ. Последнее условие особенно важно, поскольку, согласно сведениям, приводимым авторами книги, затраты на доработку уже действующих прикладных программ и устранение ошибок, выявляемых в процессе их использования, достигают иногда 75% общей суммы эксплуатационных расходов.

В книге изложены результаты исследований, выполненных с целью определения совокупности количественных характеристик программного обеспечения, учет которых способствует предотвращению ошибок или их выявлению на возможно более ранних этапах создания систем. Предлагаются способы оценки таких свойств программных средств, как завершенность, информативность, полнота описания, приспособленность к взаимодействию с человеком, блочность построения, понятность, соответствие целям и др.

В основу описываемой в книге методологии оценки качества программного обеспечения положен перечень требований, которым, по мнению авторов, должны удовлетворять хорошие программы. Отмечая, что введение дополнительных систематизированных процедур оценки качества программных средств увеличивает продолжительность проектирования, частоту общения разработчика с заказчиком, необходимое машинное время, объем работ, связанных с документированием, планированием и организацией разработок, авторы вместе с тем показывают практическую полезность такого подхода. Положительный эффект выражается в уменьшении числа тестовых прогонов программ, сокращении количества ошибок, обнаруживаемых с большим опозданием на стадии системных испытаний, снижении объема доработок программ и документации, повышении общей функциональной и эксплуатационной надежности системы.

Главное достоинство книги — ее конкретная направленность, за которой стоит большой практический опыт авторов. Приведенные в ней результаты и статистические данные пригодны для непосредственного использования разработчиками программных средств самого различного назначения.

В связи с тем что книга представляет собой публикацию материалов научно-технического отчета фирмы, переводчику пришлось устранить погрешности стиля и существенно изменить последовательность изложения. Внесенные изменения позволили сделать книгу более лаконичной и придать ей характер методического пособия по оценке качества программного обеспечения.

Эту книгу с большой пользой для себя прочтут системные и прикладные программисты, а также все те, кто связан с использованием ЭВМ в автоматизированных системах обработки данных и управления.

Е. Масловский

Предисловие

Предположим, что вы получили разработанное кем-то программное обеспечение системы в срок, не превысив при этом предусмотренных затрат. Допустим даже, что оно точно и эффективно выполняет все функции, перечисленные в документации. Значит ли это, что вы будете удовлетворены полученными средствами программного обеспечения? По целому ряду причин ответ на этот вопрос может быть отрицательным. В чем же здесь дело? А в том, что вы столкнетесь с определенными трудностями при эксплуатации, среди которых чаще других могут встретиться следующие.

1) Программное обеспечение может оказаться трудным для освоения и модификации, что вызовет дополнительные затраты на его текущее обслуживание, а эти издержки не так уж малы. Например, в недавней статье Элшофа [1] сообщается, что 75% усилий фирмы General Motors, связанных с программным обеспечением, тратится на его поддержание в работоспособном состоянии, а эта фирма — типичный представитель современного крупного промышленного производства.

2. Программное обеспечение может оказаться трудным для правильного использования и легким — для неправильного. Так, в одном из последних отчетов Главного бюджетно-контрольного управления США отмечалось, что издержки государственного бюджета на решение проблем автоматизированной обработки данных превысили 10 млн. долл.; большая часть этих затрат была связана с защитными мерами против неправильного использования программных средств [2].

3. Может оказаться, что программное обеспечение в значительной мере привязано к конкретной ЭВМ или плохо стыкуется с ранее разработанными программами. Уже сейчас эта проблема создает большие затруднения,

а по мере неизбежного увеличения числа типов вычислительных машин положение дел будет только ухудшаться.

Существует, однако, целый ряд типичных ситуаций, в которых можно эффективно воздействовать на качество программного обеспечения, и поэтому очень важно знать его основные качественные характеристики. Назовем для примера лишь несколько этапов, на которых формируются эти характеристики.

1. *Подготовка технических требований к качеству программного обеспечения.* Указание требуемых функций и параметров функционирования (скорости и точности вычислений) особого труда не представляет. Не менее важно оговорить требования, предъявляемые к удобству эксплуатации и понятности программного обеспечения, но эти свойства гораздо труднее поддаются количественному описанию, позволяющему контролировать их наличие.

2. *Проверка соответствия программных средств техническим требованиям.* Этот этап весьма важен, если требования сформулированы правильно и такая проверка имеет смысл. Она может быть четко выполнена посредством привлечения большого числа квалифицированных специалистов, однако связана с большими денежными затратами и ухудшением психологического климата в коллективе.

3. *Выработка проектных решений, обеспечивающих подходящий компромисс между затратами на разработку и эксплуатацию программного обеспечения.* Это особенно важно, так как в условиях жестких финансовых и временны́х ограничений процесса проектирования вопросы удобства эксплуатации, мобильности и пригодности для использования часто рассматриваются весьма поверхностно.

4. *Выбор пакета прикладных программ.* На этом этапе во многих случаях необходима оценка степени приспособляемости каждого конкретного пакета к изменяющимся требованиям действующей вычислительной системы и ее техническим средствам.

Первостепенный результат, которого удалось бы добиться в случае расширения наших возможностей

управления качеством программного обеспечения,— это повышение экономичности текущего обслуживания программных средств. Однако очень немногие качественные показатели последних могут быть выражены непосредственно в терминах затрат, таких, как эксплуатационная надежность, низкий уровень которой приводит к высокой стоимости обслуживания на протяжении всего срока использования программного обеспечения из-за частых корректировок и трудности приспособления к новым требованиям пользователей. Имея в виду эту простую истину, приходится только удивляться, что до сих пор в области оценки качества программного обеспечения не проводилось серьезной конструктивной работы.

Еще одна трудность состоит в том, что существующие показатели качества программных средств, как правило, неадекватно отражают те или иные их свойства, определяемые желанием, потребностями и предпочтениями будущего пользователя. В связи с большим разнообразием пользователей невозможно предложить какую-то единую универсальную меру качества программного обеспечения. Поэтому лучшее, на что может рассчитывать пользователь,— это эффективная методика оценки качественных показателей на основе хорошо продуманных детальных вопросников и ранжирования соответствующих характеристик по их важности. Однако, поскольку метрику программного обеспечения нельзя считать всеобъемлющей, общий результат такой оценки должен рассматриваться скорее как информация к размышлению, чем как окончательные выводы или предписания. Следовательно, наиболее рациональный способ действий по оценке качества программного обеспечения на сегодня состоит в том, чтобы разработать некоторую систему индикаторов его дефектов и использовать эту систему для определения направлений дальнейшего усовершенствования программных средств, планирования их испытаний, решения вопросов о целесообразности приобретения и организации эксплуатации.

Для создания методики оценки качества программного обеспечения необходимо найти ответы на три ключевых вопроса:

— Можно ли дать определения характеристикам качества программного обеспечения, которые были бы измеримы и обеспечивали бы достаточно независимую оценку качества?

— Насколько точно можно измерить качество программного обеспечения в целом или его отдельные характеристики?

— Как использовать информацию о характеристиках качества программного обеспечения для усовершенствования процесса его длительной эксплуатации в течение предполагаемого срока службы?

Эти вопросы рассматриваются в гл. 1, которая, кроме того, содержит изложение целей исследований в соответствии с принятым в данной книге подходом к решению проблемы оценки качества программного обеспечения и полученных результатов. В гл. 2 проводится анализ процесса разработки программного обеспечения. В гл. 3 даются определения различных качественных характеристик программного обеспечения и исследуются их взаимосвязи. Четвертая глава посвящена отбору подходящих характеристик для формирования метрики программного обеспечения, которая позволяла бы на практике определять, в какой мере та или иная система программного обеспечения обладает заданным набором свойств, характеризующих ее качество. В гл. 5 формулируются и анализируются основополагающие практические принципы использования метрики программного обеспечения на различных этапах разработки с целью обеспечения его высокого качества. Приложение А содержит подробное описание возможных характеристик качества программного обеспечения и примеры их конкретного применения для оценки качества программ, написанных на Фортране. Приложение Б представляет собой аннотированную библиографию работ, посвященных проблеме оценки качества программного обеспечения.

Мы надеемся, что предлагаемая вниманию читателей книга создаст хорошую основу для отчетливого рассмотрения различных аспектов качества программных средств. И хотя эта основа, конечно, не совершенна, все же ее разработка доведена до такой стадии, когда она

уже может способствовать экономической оценке программного обеспечения с точки зрения затрат и эффективности, а также послужить отправным пунктом для будущих исследований в рассматриваемой области.

ГЛАВА 1

Современное состояние проблемы оценки качества программного обеспечения

1.1. ПРЕДШЕСТВУЮЩИЕ ИССЛЕДОВАНИЯ

Вероятно, первая организованная попытка разработки методов оценки качества программного обеспечения была предпринята Руби и Гартвиком [3]. Метод, предложенный в этой работе, состоял в том, чтобы определить набор «свойств» программы и их «измерители». Желаемым свойствам давалось словесное описание, а в качестве измерителей выступали математические выражения, в которых аргументами были параметры, прямо или косвенно отражавшие конкретные свойства машинной программы. В частности, назывались такие свойства, как «A_1 — правильность выполнения математических вычислений», «A_5 — понятность программы», «A_6 — простота внесения изменений», затем — анализировавшиеся более подробно для определения более конкретных характеристик этих свойств, которые можно было бы измерять и тем самым выявлять, в какой степени то или иное свойство присуще данной программе (по 100-балльной шкале). И хотя такая детализация была выполнена для каждой из предложенных авторами характеристик качества, лишь немногим из них удалось поставить в соответствие измеримые показатели; кроме того, не приводилось никаких примеров практического применения разработанного метода.

В более позднем исследовании Брауна и Липова [4] была сформулирована система мер качества программного обеспечения и показано ее использование в рамках управляемого эксперимента для оценки двух машинных программ (каждая объемом примерно в 400 операторов Фортрана), написанных независимо друг от друга на основе одной и той же спецификации требований. Однако это исследование ограничивалось небольшим на-

бором характеристик, главным образом теми, которые соответствовали свойствам A_5 и A_6, отмеченным выше. Основное внимание в данном исследовании уделялось анализу двух различных подходов к обеспечению высокого качества программы, которая в одном случае разрабатывалась «напористым» программистом, побуждавшимся к написанию максимально эффективного варианта, а в другом — осмотрительным программистом, поощрявшимся к обеспечению простоты программы. Основные результаты проведенного исследования таковы: в «эффективной» программе было обнаружено в 10 раз больше ошибок, чем в простой (за 1000 тестовых прогонов в том и в другом случае); значения показателей качества «простой» программы оказались значительно выше. Таким образом, можно сделать вывод, что рассмотренные характеристики служат хорошими индикаторами функциональной надежности, по крайней мере в рамках данного исследования.

Одновременно и другими авторами была признана необходимость установления показателей качества программного обеспечения и использования их для оценки конкретных свойств программ. Вулф [5] предложил для этой цели и описал семь важных и достаточно независимых характеристик: удобство эксплуатации или модифицируемость, жизнеспособность, четкость, эффективность, стоимость, мобильность и степень учета человеческих факторов. Абернати и др. [6] установили ряд характеристик операционных систем и проанализировали некоторые связывающие их компромиссы. Вайнберг [7] осуществил эксперименты, в которых нескольким группам программистов давалось одно и то же задание, но устанавливались различные критерии успеха (скорость разработки программы, количество операторов, объем используемой оперативной памяти, четкость выходных данных и ясность программы). Каждая характеристика достигала своего наивысшего значения тогда, когда она являлась критерием успеха группы, разрабатывавшей программу.

Вслед за этими авторами все большее количество специалистов начинает признавать важную роль высокого качества программных средств, предпринимая

попытки в явном виде влиять на качественные характеристики программного обеспечения путем создания «хороших практических пособий по программированию». Пожалуй, наилучшим образцом такой литературы является книга Кернигена и Плоужера, посвященная стилю программирования [8]. Следует отметить также серию научно-исследовательских отчетов корпорации CIRAD [9, 10] о работах в области эксплуатационной надежности программного обеспечения; в этих отчетах можно найти полезные основополагающие сведения по проблеме, а также ряд практических рекомендаций по методам программирования и метрике эксплуатационной надежности. Среди методов программирования рассматриваются, например, концептуальная группировка, нисходящее программирование, модульное построение; в качестве характеристик программного обеспечения, определяющих удобство его эксплуатации, называются осмысленность, единообразие, компактность, понятность, переносимость на другие машины, наличие комментариев, скобочных группировок и имен для ссылок. Прекрасный обзор различных современных подходов к обеспечению мобильности программного обеспечения, таких, как моделирование одной ЭВМ на другой, эмуляция, интерпретация, самозагрузка программ, использование языков высокого уровня, дан в работе Уоррена [11]; основное внимание автор уделяет рассмотрению вопросов машинной независимости лингвистических процессоров.

Недавно был предпринят ряд инициативных шагов в направлении признания важности учета соображений относительно надежности при проектировании систем программного обеспечения. К таким шагам относятся, например, четыре доклада, представленные на конференции по проблемам управления разработками программного обеспечения, которая проводилась Американской ассоциацией авиакосмической промышленности, Ассоциацией по вычислительной технике, Институтом инженеров по электротехнике и радиоэлектронике и Министерством обороны США. Последнее проявило наибольшую инициативу в исследованиях, связанных с «установлением требований к качеству программного

обеспечения и выработкой разумных компромиссов», результаты которых излагались в докладе Де-Роуза [12]; в докладе Коснакоффа [13] рассматривались семь характеристик, которыми должны обладать «правильные» технические требования к программному обеспечению; доклад Уитэкера [14] был посвящен установлению двенадцати четких целей, которые должны удовлетворяться новыми языками программирования, создаваемыми по планам Министерства обороны США, чтобы обеспечивалось высокое качество программ; наконец, в докладе Лайта [15] были приведены пять измеримых характеристик качества программного обеспечения, предназначенных для использования при проектировании в целях гарантирования высокого качества разрабатываемых программных средств.

В самое последнее время появилось и несколько солидных по объему книг, посвященных проблемам создания метрики программного обеспечения и анализу его качества. Такова, например, книга Майерса, касающаяся вопросов надежности программного обеспечения [16], в которой внимание сосредоточивается на методах конструирования высоконадежных программ и способах оценки надежности программного обеспечения по результатам наблюдения за процессами отладки программ и устранения ошибок. В главах, посвященных стилю и языкам программирования, рассматривается помимо надежности еще и ряд других показателей качества программного обеспечения. В книге Холстэда, касающейся основ теории программного обеспечения [17], подводится итог успешных исследований, направленных на установление и экспериментальную проверку общей меры сложности программного обеспечения как функции числа различных операторов и операндов и частоты их использования.

Следует особо отметить книгу Гилба, посвященную рассмотрению метрики программного обеспечения, [18], которая наиболее близка по духу к настоящей работе. Для нее характерен столь же широкий охват показателей качества программного обеспечения, и так же определенно подчеркивается важность их использования в практических целях на основе формализованных

методов оценки (по возможности, количественных). Однако настоящая книга отличается более широким диапазоном рассматриваемых вопросов: сохраняя преемственность в отношении работы Гилба, она содержит также целый ряд результатов исследований в иных областях проблемы оценки качества программного обеспечения.

1.2. ПОКАЗАТЕЛИ КАЧЕСТВА МАШИННОЙ ПРОГРАММЫ

Первоначальные цели исследований, результаты которых являются предметом дальнейшего рассмотрения, состояли в том, чтобы выявить целесообразную совокупность характеристик качества программного обеспечения и установить затем для каждой такой характеристики один или более измеримых показателей, которые, во-первых, позволяли бы судить, в какой степени та или иная программа обладает некоторым свойством, и, во-вторых, давали возможность выводить общую оценку качества программного обеспечения как функцию значений частных показателей.

Хотя программное обеспечение может включать в себя множество таких компонентов, как перечень выполняемых функций, планы тестовых проверок и инструкции по эксплуатации, ниже речь будет идти только о метрике, применимой к входным программам, написанным на языке Фортран.

В своих исследованиях мы опирались на несколько исходных положений. Первое из них состоит в том, что при разработке программного обеспечения нас гораздо больше интересует, *где* и *как* проявляются его дефекты, чем *частота* их появления. Поэтому наибольшую ценность представляют такие автоматические средства анализа качества программного обеспечения, которые регистрируют конкретные неполадки в программе, а не просто выдают на печать обезличенные цифры, как это делалось в прошлом диагностическими средствами трансляторов: сообщения типа «1,17% операторов имеют несогласованные скобки» могли только вызывать раздражение у автора программы.

Суть второго исходного положения заключается в том, что практически для любой простой формулы количественной зависимости можно подыскать опровергающие примеры, ставящие под сомнение приемлемость ее как критерия качества программного обеспечения. Вот несколько таких примеров:

1. Существует мера структурированности программы, вычисляемая как средняя длина программных модулей. Предположим, однако, что мы имеем программное обеспечение в составе n управляющих программ объемом по 100 операторов каждая и библиотеку из m вычислительных программ, содержащих всего по 5 операторов. Такая система программного обеспечения должна признаваться хорошо структурированной при любых разумных значениях m и n. Тем не менее, если $n=2$ и $m=98$, средняя длина модуля получается равной 6,9 оператора, а при $n=10$ и $m=10$ она составляет 52,2 оператора, что должно приводить к неправильным суждениям о степени структурированности.

2. Известна мера «завершенности» программы, определяемая долей операторов, при работе которых возможны особые ситуации (деление, извлечение квадратного корня, взятие логарифма и т. п.) и которые требуют за собой написания операторов проверки факта наличия таких особенностей для их компенсации. Однако часто подобные операции выполняются в контексте, исключающем возникновение особых условий; примером может служить операция вычисления гипотенузы прямоугольного треугольника:

```
Z=SQRT(X ** 2+Y ** 2).
```

3. Был предложен ряд показателей «информативности» программы, основанных на подсчете числа перфокарт с комментариями, средней длины комментариев и т. п. Однако весьма просто найти программы, содержащие небольшое количество коротких комментариев, но более легкие для понимания по сравнению с другими программами, изобилующими бедными по своему существу комментариями.

Третье положение, принятое авторами за основу,

состоит в том, что сама технология разработки программного обеспечения пока еще находится в состоянии быстрого эволюционного развития, что в некоторых случаях затрудняет использование каких-либо устойчивых показателей качества. Введение таких метрик означало бы принудительное воздействие на существующие методы работы и не могло бы дать желаемых результатов. Так, например, практика блокирования данных и автоматического распознавания их типов делает бессмысленным применение некоторых показателей надежности, предполагающих обнаружение неправильных выражений, нарушений допустимых пределов изменения значений параметров и т. п.

Наконец, четвертое исходное положение сводится к тому, что вычисление и понимание конкретного численного значения некоторого единственного обобщенного показателя качества программного обеспечения может оказаться связанным с такими трудностями, которые нельзя оправдать. Основную проблему здесь представляет тот факт, что многие частные характеристики качества программных средств противоречивы: увеличение эффективности нередко становится возможным лишь за счет ухудшения мобильности, точности, понятности и удобства эксплуатации; повышение точности часто отрицательно сказывается на мобильности вследствие зависимости обеих характеристик от длины машинного слова; достижение высокой степени осмысленности может вступать в конфликт с ограничениями на открытость. Когда же возникают подобные конфликтные ситуации, пользователи обычно затрудняются указать, какие же характеристики с их точки зрения являются более существенными.

С учетом высказанных выше четырех соображений мы поставили перед собой задачу предложить совокупность взаимосвязанных характеристик качества программного обеспечения и разработать систему показателей, позволяющих обнаруживать аномалии соответствующих характеристик. Наш план исследований включал следующие этапы:

1. Установление совокупности характеристик, важных с точки зрения обеспечения высокого качества

программного обеспечения, не перекрывающих друг друга и достаточно исчерпывающих.

2. Разработка возможной системы показателей для оценки степени, в которой конкретному программному обеспечению присуща определенная характеристика.

3. Исследование установленных характеристик и связанной с ними системы показателей в целях выяснения степени их корреляции с качеством программного обеспечения, важности применения или выгод использования, возможности представления в количественной форме и легкости автоматического получения.

4. Оценка каждого из выбранных показателей с позиций указанных выше критериев, а также с точки зрения его взаимодействия с другими показателями с целью выяснения перекрытий, зависимостей, недостатков и т. п.

5. Преобразование множества исходных характеристик, установленных на этапе 1, в совокупность более независимых и исчерпывающих характеристик качества программного обеспечения путем уточнения исходного множества на основе оценок, полученных на этапе 4.

6. Уточнение системы показателей качества и их перегруппировка в соответствие с пересмотренным набором характеристик.

На первом этапе мы ввели следующие 11 характеристик качества программного обеспечения: понятность, завершенность, осмысленность, мобильность, согласованность, удобство эксплуатации, оцениваемость, надежность, используемость, структурированность, эффективность. Определения всех этих характеристик приводятся в гл. 3.

На втором этапе мы определили измеримые свойства Фортран-программы (т. е. ее метрику), которые могли бы служить полезными индикаторами ее понятности, удобства эксплуатации и т. д. Проделав это, мы обнаружили, что всякая мера понятности программы оказывалась одновременно и мерой удобства ее эксплуатации, поскольку для поддержания программы в работоспособном состоянии необходимо, чтобы она была понятна обслуживающему персоналу. В то же время некоторые из показателей удобства эксплуатации не имели ника-

кого отношения к понятности программы. Например, такие средства обеспечения оцениваемости, как промежуточная печать и эхо-контроль вводимых данных, важны для проведения тестирования в процессе эксплуатации программы, но никак не связаны с ее понятностью.

Таким путем мы обнаружили древовидную структуру характеристик программного обеспечения, фрагмент которой показан на рис. 1.1. На этом рисунке стрелки

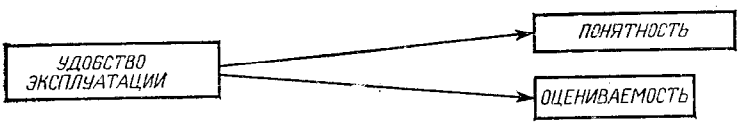


Рис. 1.1. Пример взаимосвязи характеристик программного обеспечения.

указывают логическое отношение следования: если программа удобна в эксплуатации, то она обязательно понятна и оцениваема. Иначе говоря, высокая степень удобства эксплуатации (эксплуатационной надежности) требует высокой степени понятности и оцениваемости программы.

Далее мы стали обнаруживать, что существует еще один уровень более простых показателей, который, если рассматривать рис. 1.1, должен находиться справа от характеристик понятности и оцениваемости. Например, если программа понятна, она обязательно оказывается структурированной, согласованной и осмысленной (три характеристики, входившие в исходное множество), а, кроме того, открытой и информативной (две дополнительные характеристики, не перекрывающиеся с первыми тремя). Расширив таким образом множество исходных характеристик, мы поняли, что вся их совокупность может быть отображена в виде дерева, в котором более элементарные характеристики являются необходимым условием существования более обобщенных.

В окончательном виде дерево характеристик качества программного обеспечения показано на рис. 1.2.

Верхние уровни этой структуры показывают сферы приложения оценок качества программных средств. Как правило, если кто-то приобретает пакет программ, его главным образом волнуют три вопроса:

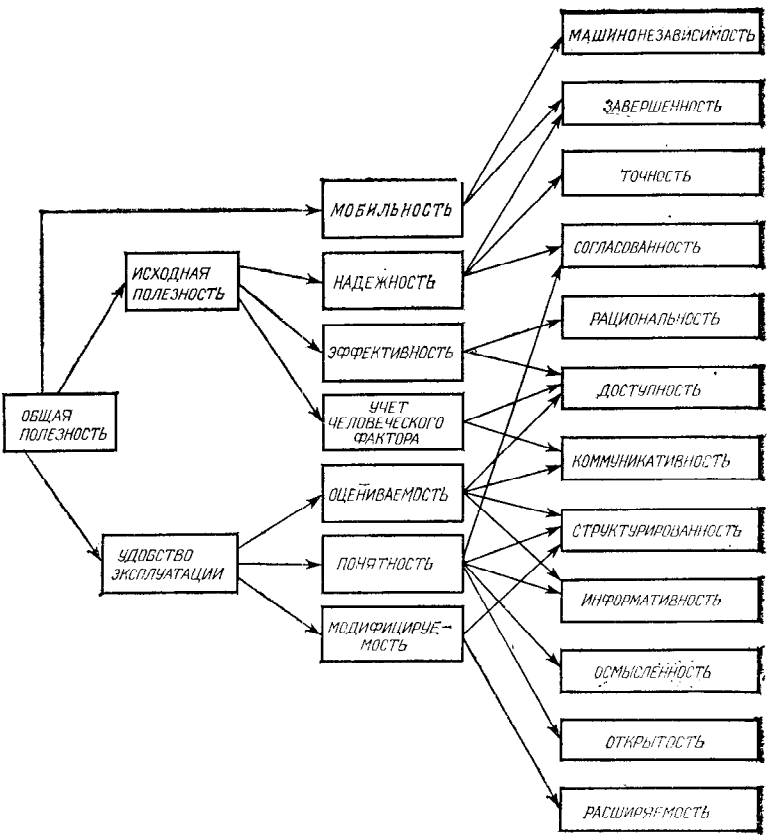


Рис. 1.2. Дерево характеристик качества программного обеспечения.

— Насколько хорошо (просто, надежно, эффективно) можно использовать пакет в его исходном виде?

— Насколько легок пакет в эксплуатации (для понимания, модифицирования и повторных испытаний)?

— Можно ли будет продолжать использовать пакет при изменении условий его применения?

Так, например, исходная полезность, удобство эксплуатации и мобильность являются необходимыми (но не достаточными [1])) элементами оценки общей полезности. Свойство исходной полезности требует, чтобы программа была надежной, в достаточной мере эффективной и разработана с учетом человеко-машинного общения, однако оно совсем не требует, чтобы пользователь испытывал программу, понимал ее внутренние механизмы работы, мог ее модифицировать или пытаться применять программу не в сфере ее основного назначения. Для обеспечения удобства эксплуатации программы необходимо, чтобы пользователь был в состоянии понять ее, модифицировать и испытывать, опираясь на возможности, предоставляемые с учетом человеческого фактора, но совсем не требуется наличия высокой надежности, эффективности или мобильности (за исключением тех случаев, когда вычислительная система пользователя работает в условиях постоянных эволюционных изменений).

Характеристики, расположенные на нижнем уровне дерева, представляют собой множество элементарных свойств, которые имеют фундаментальные отличия друг от друга и группируются в наборы соответственно условиям, необходимым для существования конкретных характеристик промежуточного уровня. Например, программа, не способная самостоятельно инициировать обращение к нужной ей области памяти, не может быть признана завершенной, а следовательно, и мобильной, если даже она будет обладать свойством машинонезависимости; программа, работающая с форматами типа 12А6, не является машинонезависимой, а значит, и мобильной, даже если она обладает свойством завершенности; а программа, которая не зависит от конкретной машины, но не обладает свойствами точности, со-

[1]) Существуют также сферы приложений, в которых необходимы и другие характеристики: например, в тех случаях, когда требуется обеспечение условий безопасности для ЭВМ. Более подробное обсуждение таких секретных систем можно найти в работе [20].

гласованности, рациональности, доступности, коммуникативности, информативности, структурированности, осмысленности, открытости и расширяемости, все же будет удовлетворять критериям мобильности.

Выделение множества элементарных характеристик, приведенных на рис. 1.2, создает гораздо лучшую основу для определения количественных показателей, которые могли бы применяться для оценки степени, в которой программное обеспечение обладает как указанными элементарными свойствами, так и характеристиками более высоких уровней. И те и другие должны участвовать в формировании оценки общей полезности системы программного обеспечения (оценки высшего уровня). Выявление и анализ соответствующих показателей проводится в следующем разделе.

1.3. ИЗМЕРЕНИЕ КАЧЕСТВА ПРОГРАММЫ

Термин «показатель» определяется нами как мера степени, в которой некоторый продукт (в данном случае программа) обладает той или иной характеристикой и проявляет ее (в данном случае качество) в процессе эксплуатации. Выше уже отмечалось, что процессы разработки и уточнения показателей и характеристик неразрывно связаны и носят итеративный характер. Нами были сформулированы и проанализированы многие показатели, применимые для оценки Фортран-программ, что в результате нескольких итераций привело к построению приведенной выше иерархической совокупности характеристик качества программного обеспечения. На основе этой структуры была выполнена оценка полезности каждого из ее элементов по следующим критериям.

1. *Наличие корреляции с качеством программного обеспечения.* Проверка по этому критерию предполагала получение ответа на вопрос, действительно ли показатель, который по предположению может служить мерой «элементарной» характеристики, имеет связь с качеством программного обеспечения? При этом под высокой положительной корреляцией мы (грубо) понимали ситуацию, когда большинство машинных программ с

высокими оценками по данному показателю обладали одновременно и соответствующей «элементарной» характеристикой. Разумеется, можно было бы использовать и более строгое в статистическом смысле определение корреляции, однако рассматриваемый этап формирования оценок носил весьма субъективный характер, при котором вряд ли было бы оправдано более точное вычисление коэффициентов корреляции, требующее значительного времени на сбор большого объема данных и вынесение суждений по множеству самых различных программ. С учетом этих соображений для каждого показателя была установлена следующая шкала оценок [1]):

В — очень высокая положительная корреляция с качеством; почти все программы с высоким значением данного показателя должны обладать связанными с ним характеристиками *(всегда)*.

ПВ — высокая положительная корреляция с качеством; значительная часть (75—90%) всех программ с высоким значением данного показателя должна обладать соответствующей характеристикой *(почти всегда)*.

О — программы с высоким значением данного показателя *обычно* (в 50—75% случаев) будут обладать соответствующей характеристикой качества.

И — некоторые из программ с высоким значением данного показателя будут обладать связанными с ним характеристиками *(иногда)*.

2. *Потенциальная выгода применения показателя.* Одни показатели дают очень важную для понимания свойств программного обеспечения информацию и помогают принимать решения как его разработчику, так и будущему пользователю; другие же, хотя и дают интересную информацию с точки зрения выявления возможных впоследствии затруднений, не приводят к

---

[1]) Здесь и далее английские сокращенные наименования оценок и показателей заменены русскими.— Прим. перев.

большим потерям при невысоких значениях даже в случае сильной корреляции этих показателей с соответствующими характеристиками качества программного обеспечения. Ясно, что суждение о потенциальной выгоде показателя зависит от целей использования программных средств, поэтому для ее оценки была установлена следующая шкала:

5 — крайне важно, чтобы данный показатель имел высокое значение, иначе возможны серьезные затруднения;
4 — важно, чтобы данный показатель имел высокое значение;
3 — хорошо бы иметь высокое значение данного показателя;
2 — в некоторой степени полезно иметь высокое значение данного показателя;
1 — выигрыш от высокого значения данного показателя крайне незначителен; при низких значениях ощутимых потерь нет.

3. *Возможность количественного представления показателя и удобство его автоматической оценки.* Возможно достижение некоторым показателем наивысших значений как по корреляции с качеством программного обеспечения, так и в отношении полезности применения, однако при этом определение численной его величины может потребовать больших затрат времени и оказаться дорогим в осуществлении. Если оценка показателя требует прочтения программы экспертом и вынесения последним экспертного суждения, то численное значение показателя будет обычно нести в себе гораздо меньше информации, чем получит специалист-эксперт в процессе выработки своей оценки. Кроме того, показатели, требующие экспертизы, крайне дороги для использования на практике. Следовательно, для больших программ желательно иметь алгоритм автоматической оценки, который анализировал бы программу и формировал соответствующее значение показателя (а по возможности, и выдавал бы на печать перечень тех мест программы, в которых соответствующая характеристика качества отсутствует). Часто оказывается более доступным промежуточный вариант, при котором используется авто-

матический компаратор, осуществляющий сравнение свойств реальной программы с перечнем желаемых характеристик, составленным пользователем. Для оценки целесообразного способа квантификации показателя, т. е. способа, обеспечивающего наиболее экономичное его определение, была установлена следующая шкала:

АЛ — требуется *автоматический алгоритм;*

КО — возможно использование *компаратора* при наличии заданного перечня требуемых характеристик (одно такое средство под названием «Программный ревизор» описывается в гл. 4);

НС — требуется *неквалифицированный специалист* для прочтения программы или принимающее лицо без большого опыта;

КС — требуется прочтение программы *квалифицированным специалистом;*

ЭК — требуется просмотр программы *экспертом;*

ПР — требуется *прогон* программы на ЭВМ.

Автоматизация таких процессов формирования оценок, как подсчет средней длины модуля или контроль наличия определенных информативных сведений, может быть выполнена довольно легко и просто. В то же время автоматизация таких оценочных операций, как глобальное сканирование программы для выявления повторяющихся выражений и гарантирования неизменности их компонентов между повторными обращениями к ним, хотя и возможна, но затруднительна. Есть и другие процедуры оценивания, автоматизировать которые практически невозможно, например суждения о содержательности и информативности выдаваемых программой сообщений, равно как и просто об их наличии или отсутствии. Таким образом, ряд автоматических средств может оказаться полезным для оценки качества программного обеспечения, но лишь частично, поскольку во многом остальном требуется прочтение программ человеком. Для ранжирования показателей по степени простоты и полноты их автоматической оценки была предложена следующая шкала:

Л — *легко* разработать автоматический алгоритм или компаратор;

Д — *довольно трудно* создать автоматический алгоритм или компаратор;

Т — *трудно* создать автоматический алгоритм или компаратор;

П — алгоритм или компаратор обеспечивает *полную* оценку показателя;

Ч — алгоритм или компаратор обеспечивает *частичную* оценку показателя;

Н — алгоритм или компаратор дают *неокончательную* оценку.

Оценка показателей по предложенным критериям. Три критерия, описанные выше, были применены для оценки первичного набора показателей, выбранных в целях анализа характеристик качества программного обеспечения. В табл. 1.1 приведен пример оценки некоторых таких показателей, включающий лишь незначительную их часть. Всего в ходе исследований был проанализирован 151 показатель. Табл. 1.1 содержит данные по пяти элементарным характеристикам и одному-двум показателям, связанным с ними. Ниже дается пояснение конкретных значений показателей, указанных в таблице.

Данные по показателю МН-1 говорят о том, что была обнаружена высокая степень его корреляции с характеристикой машинонезависимости (оценка «В») и установлена крайняя его важность для суждения об этой характеристике (оценка «5»). Кроме того, выяснилось, что наиболее экономичный способ определения степени, в которой программное обеспечение обладает данной характеристикой, состоит в совместном использовании автоматического алгоритма, прогона программы и чтения ее квалифицированным специалистом (оценка «АЛ+ +ПР+КС»). Автоматический алгоритм мог бы контролировать форматы операторов с целью выявления машинной зависимости программы, т. е. обнаружения, например, таких форматов, как 12А6 или F15.11 (означающих точность, превышающую точность вычислений большинства машин), и соответствующих им констант избыточной точности (π=3,14159265359). Такие проверки могут легко автоматизироваться (оценка «Л»), однако обеспечивают получение лишь частичной оценки характеристики машинонезависимости (оценка «Ч»).

Таблица 1.1

Анализ показателей качества программного обеспечения

Элементарные характеристики и показатели	Сущность показателей	Корреляция с качеством	Потенциальная выгодность	Возможность количественного выражения	Возможность автоматической оценки	Полнота автоматической оценки
Машинонезависимость						
МН-1	Обеспечена ли независимость точности вычислений и схем хранения данных от длины машинного слова?	В	5	АЛ+ПР+КС	Л	Ч
МН-4	Снабжены ли метками и комментариями машинозависимые операторы (т. е. те вычисления, которые зависят от аппаратных возможностей ЭВМ по адресации полуслов, байтов, определенных комбинаций двоичных разрядов или по использованию расширенных средств входного языка программирования)?	В	5	АЛ	Д	Ч
Завершенность						
ЗВ-1	Содержит ли программа средства, обеспечивающие очистку оперативной памяти перед началом ее использования?	В	5	АЛ	Л	Ч
ЗВ-2	Содержит ли программа средства начальной настройки устройств ввода-вывода данных?	В	5	КО	Л	Ч

Продолжение

Элементарные характеристики и показатели	Сущность показателей	Корреляция с качеством	Потенциальная выгодность	Возможность количественного выражения	Возможность автоматической оценки	Полнота автоматической оценки
Точность ТЧ-1	Соответствуют ли численные методы, используемые в программе, требованиям решаемой прикладной задачи?	В	5	КС		
Согласованность СГ-1	Все ли спецификации совокупностей общих переменных (т. е. тех, которые появляются в двух и более подпрограммах) тождественны (например, снабжены меткой COMMON)?	ПВ	4	АЛ	Л	П
СГ-2	Всюду ли тип переменной (действительная, целочисленная и т. д.) указан одинаково?	В	5	АЛ	Л	Ч
Расширяемость РС-1	Зарезервирована ли какая-то часть оперативной памяти для обеспечения в будущем расширенных функциональных возможностей?	В	5	АЛ	Л	Ч

Оценка эффективности метрики на основе данных об ошибках в конкретном проекте. Наилучшей основой для анализа показателей и оценок, представленных в табл. 1.1, явилась обширная база данных, связанных с ошибками в программном обеспечении одного из проектов фирмы TRW, выполненного для ВВС США. Ошибки были расклассифицированы по типам; кроме того фиксировались способы их обнаружения и устранения. Фрагмент этой базы данных, охватывающей 224 типа программных ошибок 13 основных категорий, показан в табл. 1.2, где приведены сведения о первых 12 типах ошибок. В целом база данных содержала следующие 13 категорий ошибок в программном обеспечении: 1) ошибки подготовки и ввода данных с перфокарт; 2) ошибки обработки данных при вводе перфолент; 3) ошибки при работе с дисками; 4) ошибки вывода данных; 5) неправильная выдача сообщений об ошибках; 6) ошибки сопряжения программ; 7) ошибки обращения к внешним устройствам; 8) ошибки обращения к базам данных; 9) ошибки взаимодействия с пользователем; 10) ошибки вычислительного характера; 11) ошибки модифицирования и индексации; 12) ошибки итеративных процедур; 13) ошибки поразрядной обработки двоичных слов.

На первом этапе исследований был проведен анализ ошибок каждого типа с целью определения тех стадий разработки программного обеспечения, на которых они обычно возникают (но не всегда), и тех стадий, на которых они обычно (но не всегда) исправляются. При этом рассматривались восемь стадий создания программного обеспечения: определение технических требований к программному обеспечению; разработка алгоритмов; кодирование и отладка программ; разработка контрольных примеров и тестов; контрольная проверка; приемочные испытания; стыковка с другими программами; сдача пользователю.

В табл. 1.2 буква «В» против соответствующего типа ошибок указывает стадию, на которой ошибки данного типа обычно возникают, а буква «О» — стадию, на которой они, как правило, обнаруживаются и исправляются. Для оценки применимости и полезности предложенной нами метрики исходя из возможностей выяв-

Таблица 1.2

Оценка возможностей метрики на основе выявления ошибок различных типов (данные по первым 12 типам ошибок в программах)

Стадии создания программного обеспечения / Типы ошибок	Определение требований	Разработка алгоритмов	Кодирование и отладка	Разработка контрольных примеров	Контрольная проверка	Приемочные испытания	Стыковка с другими программами	Сдача пользователю
Ошибки подготовки и ввода данных с перфокарт								
1. Программа получает параметр в формате, не предусмотренном спецификацией требований		В СГ-13				О		
2. Программа не получает или не воспринимает требуемый параметр		В СГ-13					О	
3. Программа получает параметры в порядке, не предусмотренном спецификацией требований		В СГ-13					О	
4. Программа не способна работать с данными во всем диапазоне их изменения		В СГ-13				О		
5. Программа рассчитана на работу с элементами параметра, не предусмотренными спецификацией требований		В СГ-13				О		
6. Номинальное значение параметра или значение, присваиваемое по умолчанию, отличается от предусмотренного спецификацией требований		В					О	
7. Программа воспринимает данные как правильные за пределами допустимого диапазона изменения		В						О

Продолжение

Типы ошибок \ Стадии создания программного обеспечения	Определение требований	Разработка алгоритмов	Кодирование и отладка	Разработка контрольных примеров	Контрольная проверка	Приемочные испытания	Стыковка с другими программами	Сдача пользователю
8. Программа не воспринимает данные, находящиеся в границах допустимых значений		В						О
9. Программа вызывает переполнение таблиц в оперативной памяти, занося в них данные, лежащие в допустимых пределах		В		ЗВ-9				О
10. Программа занимает отведенную ей область внешней памяти данными, лежащими в допустимых границах		В		ЗВ-9				О
11. Программа выполняет первый тест успешно, но при последующих тестах не работает		В				О		
12. Программа предусматривает местоположение параметра, не предусмотренное спецификацией требований		В СГ-13				О		

Условные обозначения, принятые в таблице:

В — стадия *возникновения* ошибки,

О — стадия *обнаружения* ошибки,

СГ-*N* — использование на данной стадии *N*-го контрольного показателя *согласованности* позволяло бы обычно обнаруживать ошибку.

ЗВ-*N* — использование на данной стадии *N*-го контрольного показателя *завершенности* позволяло бы обычно обнаруживать ошибку.

ления и корректировки ошибок по каждому типу ошибок был проведен дополнительный этап анализа: определение стадии, на которой ошибки данного типа могли бы наверняка обнаруживаться и устраняться, если бы использовалась предлагаемая метрика. Поясним это на примере. Из первой строки таблицы видно, что ошибки, характеризующиеся тем, что «программа получает параметр в формате, не предусмотренном спецификацией требований», возникают на стадии разработки алгоритма (по крайней мере так было в изученных проектах), а исправляются на стадии приемочных испытаний. Однако если бы применялся показатель СГ-13 (расширение показателя информативности ИН-1 на вводный комментарий) [1]), то его вычисление давало бы возможность выявлять и устранять указанную ошибку, как правило, еще на стадии разработки алгоритма.

Таким образом, одним из результатов проведенного анализа ошибок было расширение сформированной системы показателей качества программного обеспечения для их эффективного использования в целях обнаружения и исправления ошибок на возможно более ранних этапах. Для этого, например, требуется средство автоматического определения показателя СГ-13, которое могло бы сканировать начальные блоки программных модулей. Такое средство должно было бы проверять соответствие содержащейся в них информации той, которая необходима для характеристики требуемых входов и выходов, в том числе:

— сведениям о формате и типе данных,
— объему вводимых данных,
— последовательности ввода данных,
— описанию элементов данных,
— приемлемым диапазонам изменения данных,
— выделенным областям памяти,
— источникам данных (техническое устройство, последовательный файл или запись),
— методам доступа (только для чтения, ограниченный доступ).

---

[1]) В дальнейших главах показатель СГ-13 именуется как ИН-1$^+$.— *Прим. перев.*

Пометки СГ-13 в таблице означают, что если бы кодированию модуля предшествовало указанное выше описание его входов и выходов, то представлялось бы возможным с помощью автоматического компаратора выявить соответствующие ошибки еще до начала программирования.

Из табл. 1.2 следует, что из 12 перечисленных в ней типов ошибок обработки данных при вводе перфокарт компаратор мог бы выявить шесть. Всего благодаря введению показателя согласованности СГ-13 в рассмотренном проекте могло быть обнаружено 18 типов ошибок из 224. Следующая по эффективности дополнительная возможность выявления ошибок могла бы появиться в случае реализации проверки соответствия фактической программы тому описанию модуля, которое предлагалось выше при рассмотрении показателя СГ-13. Например, для каждой операции выдачи данных можно было бы проверять, появляется ли переменная слева от знака равенства, и выполнять контроль вычисляемых элементов. Благодаря этому могло бы выявляться еще 10 типов ошибок, однако уже только на этапе сканирования готовой программы.

Из табл. 1.3 видно, что в общей сложности показатели завершенности и согласованности оказались самыми эффективными средствами выявления программных ошибок: в целом с помощью метрики завершенности можно

Таблица 1.3

Исправляющая способность показателей качества программного обеспечения

Показатели — элементарные характеристики	Количество исправленных типов ошибок	Суммарный сдвиг фаз обнаружения ошибок
Завершенность	37	62
Согласованность	34	89
Коммуникативность	17	35
Структурированность	2	2
Информативность	1	4
Осмысленность	1	4
Точность	1	2
Доступность	1	2

было бы «выловить» 37 типов ошибок из 224; суммарный накопленный сдвиг фаз в сторону ускорения обнаружения этих ошибок составил 62, или в среднем 1,7 фазы на каждый тип ошибок. В отношении метрики согласованности эти цифры равны соответственно 89 и 2,5.

Главное, о чем говорит табл. 1.3, это то, что использование автоматических или полуавтоматических средств контроля согласованности и завершенности на начальных этапах разработки программы ведет к значительному повышению эффективности обнаружения и устранения ошибок в программном обеспечении. Это, конечно, очень важный результат, однако он не должен вызывать удивления, поскольку показатели завершенности и согласованности относятся к двум элементарным характеристикам качества, непосредственно связанным с надежностью программного обеспечения.

Еще одним полезным результатом исследования было сопоставление корректирующих возможностей каждого показателя с оценками потенциальной выгоды его использования, определенными в табл. 1.1. К нашему удовлетворению, практически все показатели с высокой обнаруживающей и корректирующей способностью достигали максимальной величины оценки выгодности их применения, равной 5, и ни один из показателей, существенных с точки зрения возможностей устранения ошибок, не получил оценки ниже 3.

Разумеется, проведенное исследование базы данных, связанных с ошибками программирования обеспечило лишь частичную оценку характеристик качества программного обеспечения. Однако в связи с тем, что тестирование всегда занимает довольно большую часть ресурсов, необходимых для его разработки, полученные в результате анализа оценки оказались чрезвычайно полезными для принятия решения относительно того, какие показатели следует преимущественно разрабатывать и использовать. В дальнейшем нами были созданы и довольно успешно применялись средства контроля некоторых из рассмотренных показателей; в частности, показатель СГ-13 послужил основой недавно разработанного способа контроля согласованности проектных решений, описанного в работе [21].

1.4. ИСПОЛЬЗОВАНИЕ ХАРАКТЕРИСТИК КАЧЕСТВА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ ВНЕСЕНИЯ УСОВЕРШЕНСТВОВАНИЙ НА ОТДЕЛЬНЫХ ФАЗАХ ЕГО ЖИЗНЕННОГО ЦИКЛА

Цикл жизни программного обеспечения начинается с выработки требований ко всей создаваемой системе и ее программным средствам, затем следуют этапы технического проектирования, рабочего проектирования, кодирования и отладки и, наконец, завершающая фаза — ввод в действие и текущее обслуживание для поддержания программных средств в работоспособном состоянии. Повышение жизнеспособности программного обеспечения можно осуществлять следующими четырьмя способами, основывающимися на использовании обсуждавшихся выше характеристик его качества:

— установлением в явном виде целевых параметров качества и их относительных приоритетов;

— применением контрольных таблиц и вопросников по анализу качества;

— введением специальных мер, гарантирующих высокое качество программ;

— использованием специальных средств и методов повышения качества.

Рассмотрим каждый из этих способов более детально. **Целевые параметры качества и их относительные приоритеты.** Эксперименты, описанные в работах [4, 7], показали, что уровень качества, закладываемый исполнителем в программу, сильно коррелирован с теми параметрами качества и приоритетами, которые указываются в задании. Поэтому, если пользователь предпочитает эффективности программы ее мобильность и удобство эксплуатации, важно довести это до сведения разработчика, причем, сделать это в такой форме, которая позволяла бы в дальнейшем определить, до какой степени желаемые свойства реализованы в готовой системе программного обеспечения.

По всей видимости, на сегодня наиболее эффективным путем достижения указанной цели является нормиро-

вание качества программного обеспечения [1]). Обычно оно применяется лишь для определения и обеспечения рациональности использования технических средств системы и ее вычислительных ресурсов в условиях специфики пользовательских задач, однако аналогичным образом нормирование может служить и для приемочных испытаний или выбора программных пакетов по критериям, включающим иные характеристики качества. Так, например, в целях обеспечения удобства эксплуатации возможно создание контрольного примера, достаточно широко отображающего возможные модификации программных средств в реальных условиях пользователя, и проверка на нем квалифицированности и эффективности действий конкретного обслуживающего персонала.

При использовании нормирования для организации приемочных испытаний для всех участников разработки программного обеспечения на всех этапах разработки устанавливается четкое множество целей. Сейчас это делается преимущественно для гарантирования характеристик надежности и коэффициента готовности технических средств и программного обеспечения (особенно успешно, например, в лабораториях фирмы Bell при создании электронных систем коммутации), но может столь же эффективно использоваться и для других критериев качества.

Нормирование качества программного обеспечения может применяться также для оценки альтернативных возможностей приобретения программных средств. При этом объем усилий, затрачиваемых на контроль качества, должен быть пропорционален ожидаемой интенсивности использования приобретаемых средств, а не их стоимости. Мы, например, неоднократно несли потери в размере более 10 000 долл. на дополнительные затраты по разработке и эксплуатации вследствие несовершенного контроля качества приобретавшихся про-

[1]) Другой важный подход состоит в анализе и разработке специальных функциональных требований, в результате выполнения которых программному обеспечению придаются необходимые свойства. Прекрасным примером такого подхода является исследование Фишера, рассмотренное в приложении Б.

граммных пакетов стоимостью всего в 1—2 тыс. долл.

Контрольные таблицы и вопросники по анализу качества. Рассматривавшаяся выше метрика качества программного обеспечения, детально обсуждаемая в последующих главах книги, может служить основой для разработки совокупности вопросников, которые пригодны для применения при оценках, обзоре, контроле и других независимых конструктивных действиях, сопутствующих созданию систем программного обеспечения. Но и эти средства использовались пока лишь преимущественно для обеспечения показателей надежности, однако возможно их эффективное применение и для других крите-

Таблица 1.4

Фрагмент вопросника для оценки информативности программного обеспечения

А. Содержит ли каждая программа начальный блок комментариев, который включает имя программы, дату ввода в действие, требования к точности данных, назначение, ограничения и условия использования, хронологию внесения изменений, описание входов и выходов, метода решения задачи, принятых допущений, процедур выявления ошибок во всех предвидимых ситуациях неправильных входов?

Б. Достаточно ли полно описаны логические блоки и связанные с ними альтернативные ветви алгоритма?

В. Адекватно ли описаны функции модулей, их входы и выходы с точки зрения возможностей тестирования?

Г. Имеются ли сведения, обеспечивающие возможность работы с определенными конкретными значениями входов для выполнения специализированных проверок программы?

Д. Существует ли информация о процедурах, учитывающих влияние изменений в одном блоке на все остальные?

Е. Имеется ли информация, обеспечивающая распознавание программы, в которую должны вноситься изменения?

Ж. В тех случаях, когда модули зависимы, отображается ли этот факт в специальном комментарии, программной документации или внутренней структуре программы?

З. Несут ли имена переменных информацию об их физических свойствах или функциональном назначении?

И. Достаточно ли четко описываются специальные функции программы и целевое назначение каждой из них (например, в комментариях)?

К. Достаточен ли объем информации, для того чтобы понять связь конкретных имен переменных с соответствующими физическими свойствами или объектами?

риев качества. Для примера в табл. 1.4 приведен фрагмент вопросника, касающегося свойства информативности программного обеспечения, имеющего важное значение для оценки долговременных затрат на программное обеспечение, освоение, контроль и эксплуатацию. Говорят, что программные средства обладают свойством информативности, если их документация содержит достаточно сведений, для того чтобы пользователь мог устанавливать или изменять свои цели, принятые допущения, систему ограничений, входы, выходы, компоненты программного обеспечения и принимать решения по его доработке.

Специальные меры, гарантирующие высокое качество. Как с точки зрения достижения необходимых качественных показателей, так и с точки зрения экономичности использования программного обеспечения становится все более важным введение при разработке проектов специальных действий, направленных на гарантирование характеристик качества. Руководитель соответствующих работ должен, как правило, подчиняться руководителю всего проекта и удерживаться в стороне от выпуска какой-либо программной продукции, с тем чтобы всегда иметь объективный взгляд на проект. Конкретные задачи, включаемые в сферу его деятельности по обеспечению качества, индивидуальны для каждого проекта, причем зависят от масштабов и цели последнего. Такой подход продемонстрировал высокую эффективность в отношении учета желаний заказчика программных средств и особенностей конкретного применения. Функции обеспечения качества обычно включают следующие виды деятельности:

— Планирование, заключающееся в подготовке плана работ, в котором находят отражение требования к качеству, распределяются задачи между исполнителями, составляются графики и разделяется ответственность.

— Разработку стратегии, методов и процедур, состоящую в создании нормативных инструкций, охватывающих все этапы проектирования, в том числе, установление требований к системе, программирование, тестирование в соответствии с нуждами конкретного проекта. Ключевым моментом этой деятельности яв-

ляется возможно более раннее внимание к вопросам качества.

— Развитие средств, предназначенных для достижения высокого качества программного обеспечения, которые предполагают адаптацию имеющихся и разработку новых ручных и автоматических процедур, позволяющих контролировать его соответствие установленным требованиям, касающимся эффективности функционирования и стандартов качества.

— Проведение ревизий, состоящих в анализе процедур проектирования и нормативной документации с точки зрения их соответствия установленным планам стандартизации процесса разработки программного обеспечения, в слежении за выполнением этих планов и наличием корректирующей документации.

— Наблюдение за испытаниями программных средств, предполагающее составление отчетов об испытаниях с анализом проблем, причин ошибок и обоснованием корректирующих действий.

— Сохранение рабочей документации, касающейся отчетов о проблемах проектирования и программирования системы, контрольных примеров, протоколов испытаний, тестовых данных, обзоров качества и прочих аспектов.

— Физический контроль запоминающих устройств, включающий проверку дисков, магнитных лент, перфокарт и иных носителей, предназначенных для хранения программ, при каждом случае передачи их на использование или сохранение, с тем чтобы гарантировать отсутствие разрушений или изменений в результате воздействия внешней среды или последствий неверного использования.

Большинство из перечисленных функций относилось пока лишь к обеспечению надежности программных средств, их соответствию стандартам и техническим условиям. Однако указанная деятельность может также охватывать и задачи обеспечения других характеристик качества, например удобства эксплуатации и мобильности.

Специальные средства и методы повышения качества. Практически любое программное средство или метод

программирования — будь то генераторы перекрестных ссылок, программы для вычерчивания блок-схем, процедуры управления выбором конфигурации системы или программные мониторы — в той или иной степени способствует улучшению качества программного обеспечения, по крайней мере в аспекте хотя бы одной его характеристики. Целесообразно здесь обратить внимание на некоторые из таких средств, значение которых особенно велико на этапах выработки системных требований и программирования.

На этапах выработки системных требований и проектирования системы приобрести некоторую уверенность в достижении требуемых характеристик качества можно посредством использования руководящих материалов и детальных вопросников. При этом, конечно, цель состоит не в том, чтобы определить численную оценку степени, в которой достигнута та или иная характеристика качества, а в установлении конкретных задач, которые требуется решить для ее обеспечения. В настоящее время накоплен большой арсенал средств, использующих машинный анализ программных спецификаций для автоматического определения их согласованности, завершенности и т. п. в соответствии с выдвинутыми требованиями. К таким средствам относятся, в частности, системы ISDOS [22] и SREP [23, 24], позволяющие значительно надежнее гарантировать требуемые характеристики качества программного обеспечения по сравнению с ручными методами и создающие благоприятные условия для сокращения расходов по тестированию и эксплуатации, связанных с ошибками в программах. Этого удается добиться потому, что большинство ошибок возникает вследствие неверного понимания требований и несовершенства процесса проектирования [21], и исправление их на ранних этапах создания программного обеспечения обходится гораздо дешевле, чем на более поздних.

Одним из самых существенных источников ошибок и трудностей является нечеткость требований к программному обеспечению, указываемых в соответствующих спецификациях. Между тем различные группы специалистов — проектировщики, испытатели, инструкторы,

пользователи — должны разбираться в этих спецификациях и работать с ними независимо друг от друга. Поэтому, если толкование выдвигаемых требований будет у них различным, неизбежно возникнут многочисленные трудности разработки и эксплуатации.

Лучший способ избежать указанных трудностей — это проводить анализ требований с точки зрения возможности контроля их выполнения. Для примера ниже приводятся фрагменты двух спецификаций:

Спецификация, непригодная для проверки свойств системы	Спецификация, удобная для проверки свойств системы
Точность должна быть достаточной для планирования выполнения задания	Ошибка в определении координат местоположения объекта должна быть не более 50' по горизонтали и не более 20' по вертикали
Система должна реагировать на текущие запросы о положении объекта в реальном масштабе времени	Система должна иметь реакцию на запросы типа А — не более 2 с, на запросы типа Б — не более 10 с, на запросы типа В — не более 2 мин
Моделирование поведения объекта должно заканчиваться по истечении времени соответствующей смены	Моделирование поведения объекта должно заканчиваться после истечения 8 ч модельного времени

Очевидно, что правая спецификация не только более пригодна для проверки выполнения поставленных требований, но, кроме того, более конкретна и создает лучшую основу для проектирования, оценки затрат, документирования, эксплуатации и текущего обслуживания системы.

Для анализа требований к качеству, устанавливаемых на этапе выработки требований к системе, существует один полезный метод, основанный на составлении матрицы «требования — свойства» [25]. В этой матрице в столбцах располагаются отдельные функциональные требования, а в строках — основные желаемые характеристики качества (или свойства) программного обеспечения. Возможно и обратное расположение строк и столбцов. Элементы матрицы представляют собой до-

Требования / Свойства	Моделирование поведения объекта должно заканчиваться по истечении времени соответствующей смены	...
Оцениваемость	Моделирование поведения объекта должно заканчиваться после истечения 8 ч модельного времени	...
Модифицируемость	Пользователь должен иметь возможность задавать цикл моделирования как входной параметр, которому по умолчанию присваивается значение, равное 8	...
Завершенность	Необходимо предусмотреть альтернативный критерий окончания моделирования на случай, когда невозможно уложиться в заданное время моделирования	...
.	.	.
.	.	.
.	.	.

Рис. 1.3. Фрагмент матрицы «Требования — свойства».

полнительные функциональные требования, возникающие в ходе детального анализа аспектов качества, связанных с обеспечением каждого необходимого свойства. Для примера на рис. 1.3 показан фрагмент подобной матрицы, относящийся к третьему пункту рассмотренной выше спецификации; ясно, что полученные в результате анализа уточненные и детализированные требования приведут к повышению качества программного обеспечения. Разумеется, повышения качества невозможно добиться без дополнительных усилий. Однако, если разрабатываемая программа характеризуется частым использованием и высокой интенсивностью обслуживания, эти усилия окупятся за счет сокращения эксплуатационных расходов в течение всего цикла жизни системы.

Возможно использование средств повышения качества и на стадии программирования. Для примера

рассмотрим отображенное на рис. 1.2 свойство структурированности программы, являющееся необходимой элементарной характеристикой для обеспечения ее оцениваемости, понятности и модифицируемости. Последние в свою очередь представляют собой условия, требуемые для обеспечения удобства эксплуатации. Были разработаны допустимые синтаксические конструкции Фортрана, охватывающие основные управляющие операторы SEQUENSE, IF-THEN-ELSE, CASE, DO-WHILE и DO-UNTIL [26] и предназначенные для использования в качестве стандартов при разработке крупных систем программного обеспечения, функционирующих в режиме реального времени. Для таких систем создана специальная программа под названием STRUCT, которая производит сканирование входных программ и обычно используется для определения того, в какой мере та или иная программа надлежащим образом структурирована с помощью стандартных конструкций. В случае обнаружения нежелательных отклонений от стандартов вызвавшая их конкретная программа выявляется и подвергается диагностированию.

Строгая дисциплина, требующаяся для применения указанных средств при разработке каждого проекта, первоначально вызывала определенное сопротивление и недовольство со стороны программистов. Руководители программных разработок приходили в смятение от того, что программы, созданные до введения стандарта, требовалось переделывать почти заново. Это приводило к напряженному бюджету, дополнительным затратам труда и трудностям соблюдения установленных графиков выполнения работ. Однако впоследствии как сами программисты, так и их непосредственные руководители стали отмечать в подавляющем большинстве случаев ожидаемое снижение эксплуатационных расходов, высказывая одновременно положительное мнение по поводу улучшения качества программ в отношении согласованности и понятности. Кроме того, выражалось мнение, что, если бы стандарт был введен с самого начала разработки, многих трудностей удалось бы избежать.

Последующий анализ ошибок в процессе тестирования программного обеспечения системы, о которой идет

речь в [26], показал, что частота их возникновения крайне мала; это достижение частично приписывалось применению стандарта структуризации программ, хотя в основном оно связывалось с введением требования детального анализа всех ветвей программы до начала ее испытаний в целом.

Стандарт структуризации — это всего только один из более чем 30 стандартов программирования, разработанных для рассматриваемого проекта и помогавших автоматически выявлять нежелательные отклонения с помощью специального средства анализа Фортран-программ под названием ПРОГРАММНЫЙ РЕВИЗОР. В результате его применения обнаруживались имевшиеся нарушения стандарта и указывалось в диагностической распечатке местоположение программы, не соответствующей стандарту. С помощью ПРОГРАММНОГО РЕВИЗОРА можно выявлять, в какой мере программы обладают элементарными характеристиками, отображенными на рис. 1.2. Например, существует возможность частично измерять информативность программного обеспечения путем установления факта наличия или отсутствия перфокарт с данными стандартного начального блока, несущего необходимое описание, а также карт, комментирующих передачи управления. Частично может быть измерена и согласованность посредством выявления разнородных выражений и несоответствий стандартам сопряжения модулей.

Не вызывает сомнения тот факт, что в будущем кое-что из сказанного будет более эффективно реализовано посредством применения стандартных возможностей специального языка структурного программирования, типизации данных и т. п. Тем не менее всегда будет сохраняться потребность в автоматическом просмотре машинных программ для гарантирования их соответствия частным требованиям (т. е. поименованным соглашениям) и для контроля за показателями, сигнализирующими о возможных потерях качества программного обеспечения.

Из табл. 1.1 со всей определенностью видно, что существует целый ряд аспектов оценки качества программ (и проектов), которые требуют привлечения ква-

лифицированных специалистов. В работе [27] с этой точки зрения анализируется опыт крупных разработок программного обеспечения, причем основное внимание сосредоточивается на сравнении достоинств разнообразных методов проверки и измерения качественных характеристик людьми и автоматическими средствами. Среди этих методов следует особо отметить метод, основанный на применении определенных принципов проектирования и инспектировании программ [28], а также родственный ему метод конструктивного анализа. В нашей работе [29], посвященной исследованию ошибок в программном обеспечении одного крупного проекта, обнаруженных на этапе его утверждения, мы определили, что 59% этих ошибок могло бы быть заранее исключено путем надлежащего использования имеющихся методов контроля проектов. Результаты, полученные Фаганом [28], свидетельствуют о том, что дополнительные усилия по контролю проектных решений позволяют добиться значительного сокращения количества ошибок в период эксплуатации системы: данные анализа проекта, в котором такой контроль был организован, довольно убедительно говорят о сокращении числа ошибок на 23%.

Вклад нашего собственного исследования в решение проблем оценки качества программного обеспечения состоит в том, что мы предприняли первую попытку создания вполне определенной основы для конструктивного рассмотрения порою неявных аспектов качества, представив совокупность согласованных и в общем полезных определений характеристик программных средств, проанализировав их различия, выработав практические рекомендации и обобщив накопленный опыт оценки качества программного обеспечения.

1.5. ПЕРСПЕКТИВНЫЕ НАПРАВЛЕНИЯ ДАЛЬНЕЙШИХ ИССЛЕДОВАНИЙ

На основе материалов, излагаемых в данной книге, возможна организация целого ряда исследовательских проектов, позволяющих надеяться на получение полез-

ных и важных результатов. Такими направлениями исследований могли бы быть следующие:

1. Разработка алгоритмов и машинных программ для автоматического определения оценок аномальных отклонений качества программного обеспечения, обнаруживаемых с помощью введенной метрики.

2. Применение машинных программ автоматического определения оценок в практике создания систем программного обеспечения и приобретения готовых программных пакетов с целью установления и совершенствования показателей их качества.

3. Преобразование системы показателей качества программного обеспечения в контрольные вопросники, позволяющие руководителям разработки программных средств и экспертам оценивать степень учета в проектных решениях различных требований к программному обеспечению.

4. Расширение предложенной метрики программного обеспечения с целью охвата помимо исходных программ еще и других программных средств.

5. Организация постоянно действующей службы сбора и распространения информации о системе показателей качества программного обеспечения и об опыте ее использования.

но наблюдающейся тенденцией модификации программных средств.) Для суждения по существу о качестве того или иного изделия — будь то технические средства или программное обеспечение — простой проверки свойств конечного продукта недостаточно. Невозможно организовать в условиях обычных ограничений на время и ресурсы испытания программных средств во всех режимах их функционирования, в любых окружающих условиях и при всех мыслимых ситуациях, в которых они должны сохранять свою работоспособность. Поэтому наиболее рациональным способом использования имеющихся ресурсов представляется такой, при котором на каждом этапе формализованного процесса разработки программного обеспечения производится тщательный анализ и проверка получаемых результатов работы.

2.1. СУЩНОСТЬ ПРОЦЕССА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Весь процесс создания программных средств может быть разделен на ряд более или менее независимых этапов, или фаз. Конкретное число таких этапов и конкретное их содержание определяется в какой-то мере целями и масштабами проекта. Этапы разработки программных средств, характерные для крупных систем программного обеспечения, поставляемых внешним заказчикам, отображены на рис. 2.1. На каждом из отмеченных этапов либо появляются готовые программные средства, либо вносится определенный вклад в их создание. На рис. 2.2 показаны основные виды «программного продукта», который может представлять собой либо документацию, либо машинную программу. Группирование этих видов программного продукта определенным образом позволяет создать основу для его формализованного анализа. После утверждения в соответствующих инстанциях некоторые из программных средств становятся «базовыми» и образуют фундамент для последующего целенаправленного усовершенствования и модифицирования программного обеспечения. Взаимосвязи между программными продуктами, анализом

ГЛАВА 2

Программное обеспечение и процесс его разработки

Под программным обеспечением мы понимаем совокупность программных средств, связанных с ними данных и необходимой документации. Разработка программного обеспечения представляет собой формализованный процесс, посредством которого цели создания программ трансформируются в требования к ним; требования в свою очередь воплощаются в проектные решения, реализуются в виде машинных программ, которые отлаживаются, тестируются и в итоге вводятся в эксплуатацию. Степень формализованности процесса разработки программного обеспечения зависит от целей его создания, конечного назначения и численности группы разработчиков. В одном крайнем случае эта группа может состоять из одного-единственного человека, осуществляющего поэтапную разработку программы со своего терминала в режиме разделения времени в непринужденной обстановке. В другом — предметом разработки может быть крайне сложная программа, предназначенная для функционирования в реальном масштабе времени и требующая трудозатрат объемом в тысячи человекочасов. Очевидно, что эти две крайние ситуации характеризуются в корне различной степенью формализации процесса разработки программных средств.

Связь качества программного обеспечения с процессом его разработки аналогична зависимости между качеством технических средств и процессом их создания. И точно так же, как обеспечение высокого качества аппаратуры требует формализации процессов ее проектирования, практика создания программных средств диктует необходимость правильной организации процесса их разработки. (Правда, это затрудняется постоян-

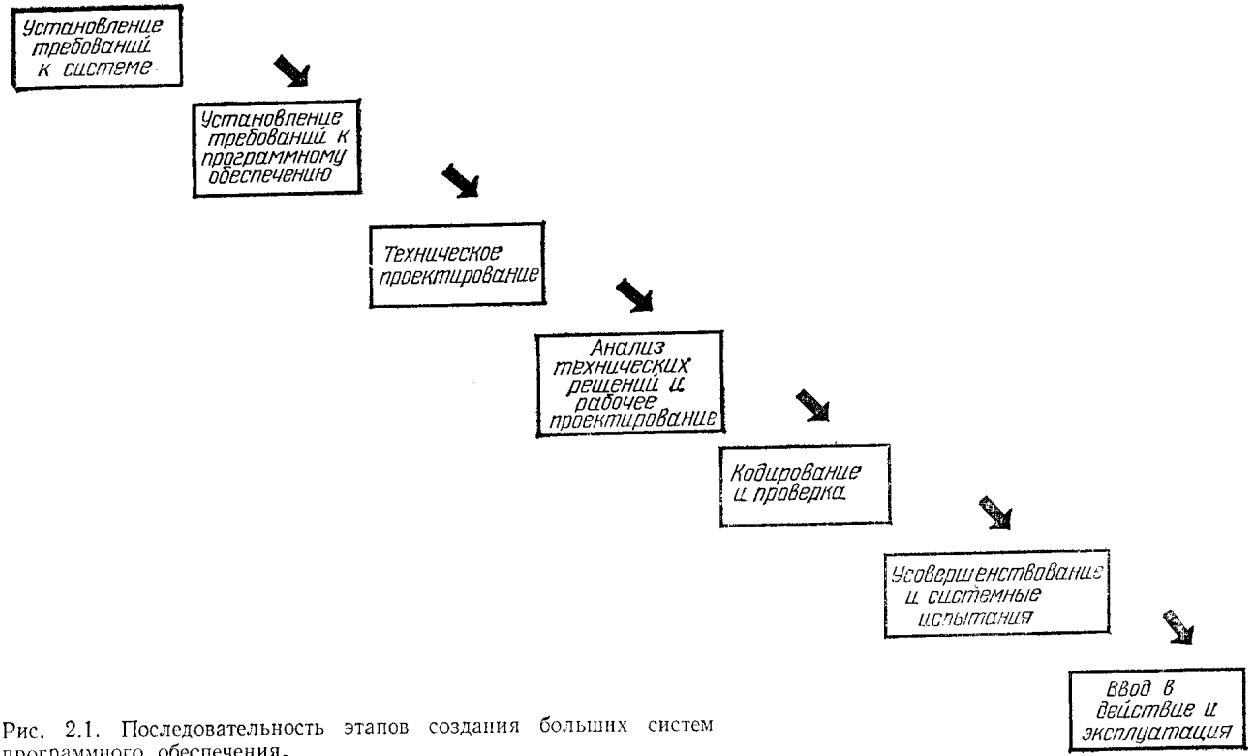


Рис. 2.1. Последовательность этапов создания больших систем программного обеспечения.

Рис. 2.2. Основные результаты, получаемые на отдельных этапах создания программного обеспечения.

Таблица 2.1

Взаимосвязи между программными продуктами, анализом технических решений и базовой документацией программного обеспечения

Вид анализа	Рассматриваемый программный продукт	Цели анализа	Формируемая базовая документация
Анализ системных требований	Спецификация требований к системе	Понять общие требования к создаваемой системе и ее интерфейсам в самом начале проектирования с точки зрения выработки условий контракта на разработку программного обеспечения [1]	Функциональная структура, определенная на предшествующих этапах создания системы [2]
Анализ требований к программному обеспечению	Спецификация требований к программному обеспечению (технические условия)	Утвердить требования к программному обеспечению и начать его техническое проектирование	Требования к организации проектирования (распределение требований по различным уровням разработки) [2]
Анализ требований к техническому проекту	Проектное задание, план контрольных проверок	Проверка учета требований в техническом проекте, в том числе к распределению функций, интерфейсам, структуре базы данных и проведению контрольных проверок	Базовый вариант технического проекта
Критический анализ требований к рабочему проектированию	Проектная спецификация, описание планируемых процедур контроля проектных решений	Утвердить рабочий проект, включая решения, относящиеся к базе данных, процедуры контроля проектных решений, и получить разрешение на начало работ по программированию	Базовый вариант рабочего проекта

Продолжение

Вид анализа	Рассматриваемый программный продукт	Цели анализа	Формируемая базовая документация
Анализ контрольных тестов	Описание планов и процедур контрольного тестирования	Утвердить формальные процедуры тестовых проверок и критерии приемлемости результатов, утвердить принятый набор модулей и начать официальное тестирование программ	Базовый план проведения испытаний
Анализ результатов приемочных испытаний	Проектная спецификация (скорректированная), листинги, инструкция пользователю, инструкция по эксплуатации, эталонная перфолента	Принять пакет программ, начать работы по обеспечению эксплуатационной готовности	Базовый вариант технического описания [2]
Проверка работоспособности программного обеспечения путем ее демонстрации в действии	Отчет об испытаниях	Подтвердить эксплуатационную готовность программных средств, начать их практическое использование	Базовый вариант эксплуатационной документации

[1] Предполагается, что отправным пунктом процесса разработки программного обеспечения служит подготовленная ранее системная спецификация той системы, по отношению к которой оно является подсистемой.
[2] Документация предусматривается государственным стандартом США MIL-STD-1521.

технических решений и базовой документацией описаны в табл. 2.1.

Основная задача целенаправленного модифицирования программного обеспечения, или управления его конфигурацией, состоит в контроле за внесением изменений в базовые средства и охватывает не только средства, одобренные заказчиком, такие, как проект организации работ и принципы приемки готовой продукции, но и внутренние средства проектирования, такие, как рецензирование проектов и организация тестирования программ. Функция управления конфигурацией программного обеспечения предусматривает также обеспечение возможностей ее идентификации, ведение записей текущего состояния программных средств, их текущий анализ и проверку.

Рассмотрим теперь более подробно основные виды программного продукта и факторы, определяющие качество программного обеспечения.

2.2. ВИДЫ ПРОГРАММНОГО ПРОДУКТА

В дальнейшем обсуждении предполагается, что программное обеспечение создается как подсистема более крупной системы, для которой предварительно сформулированы технические требования и установлены базовые принципы. Кроме того, предполагается, что эти технические требования включают в себя наиболее общие требования к подсистеме программного обеспечения в отношении необходимых функций, эксплуатационных качеств и организации проектирования, а также в отношении интерфейсов, которые должны использоваться в этой подсистеме. Требования, касающиеся внешней информации, как по входам, так и по выходам, считаются частью требований к интерфейсам. Документация, создаваемая на различных этапах разработки программного обеспечения, показана на рис. 2.2, где она помечена стрелками, ответвляющимися от соответствующих прямоугольников. Поясним коротко суть каждого из документов.

Технические условия на программное обеспечение отображают непрерывный процесс распределения функ-

циональных требований к эффективности и структуре системы по элементам нижних уровней — в данном случае по отдельным компонентам программного обеспечения. При этом требования, касающиеся сопряжения компонентов и необходимых данных, тоже рассматриваются как составная часть технических условий на программное обеспечение.

Технический проект включает предварительное описание проектных решений по четырем разделам будущего рабочего проекта, которые касаются функциональной блок-схемы программы, определения функций управления вычислительным процессом, распределения памяти и машинного времени и структуры базы данных. Здесь же детализируются требования к внутренним интерфейсам. Сюда же относится и план контрольного тестирования, который уточняется и корректируется в ходе анализа решений, предусматриваемых техническим проектом.

План контрольного тестирования содержит описание процедур проверки и приемки машинных программ, включая условия их проверки, порядок демонстрации в действии и оценки качества функционирования. Сюда же входит должным образом продуманный график осуществления работ и выпуска документации, необходимой для комплектной поставки пользователю пакета программ и для обеспечения его ориентации на специфические условия использования у заказчика.

Проектная спецификация определяет функции программного обеспечения, подлежащие реализации в рабочем проекте и выработанные в результате критического анализа технического проекта. Она служит основой для организации рабочего проектирования и обновляется после завершения проверки программ, с тем чтобы отразить реальную структуру программного обеспечения. Обновленная проектная спецификация совместно с перечнем принятых в программах символических имен по окончании анализа результатов приемочных испытаний образует базис технического описания системы программного обеспечения.

Описание планируемых процедур контроля проектных решений содержит сведения о неформализуемых опера-

циях проверки и ожидаемых результатах, а также о порядке отчетности на этом этапе. Все действия по контролю проектных решений должны осуществляться на основании этого документа, играющего организующую роль и предназначенного для самих разработчиков программного обеспечения. Он не включает формализованных тестовых процедур, выполняемых специальной независимой группой гарантийных испытаний.

Описание процедур контрольного тестирования включает в себя формальные контрольные примеры и критерии анализа результатов тестовых проверок. Здесь излагаются цели тестирования, распределение обязанностей и ответственности между штатным персоналом, порядок проведения тестирования, детальное описание самих тестов, включая исходные данные, выходы, отображаемые события и ожидаемые результаты, а также требования к процедурам предварительной обработки и анализа данных.

Отчет об испытаниях содержит анализ хода формального тестирования и его результаты и охватывает как контрольные, так и приемочные испытания. Этот отчет предоставляет рядовому и руководящему персоналу техническую информацию, касающуюся обнаруженных дефектов или неполного достижения поставленных целей и текущего состояния разработки.

Инструкция пользователю является справочным пособием для лиц, применяющих данные программные средства в своей работе, и потому содержит всю информацию, необходимую для эффективного их использования. Отдельные разделы этой инструкции охватывают вопросы работы за пультом ЭВМ, ввода исходных данных, реакции на запросы машиной требуемой информации, использования результатов. Специально должно быть выделено описание всех выходных форматов, сообщений об ошибках и процедур восстановления работоспособности системы.

Инструкция по эксплуатации содержит информацию, требующуюся программистам, ответственным за нормальное функционирование программных средств. Здесь необходимы весьма подробные сведения о самих программах, связанных с ними данных и условиях экс-

плуатации. Эта инструкция должна также содержать описание процедур сборки, загрузки и текущего обслуживания программ, равно как и процедур контроля их целостности.

2.3. КАК И ДЛЯ ЧЕГО НЕОБХОДИМО ИЗМЕРЯТЬ КАЧЕСТВО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ?

Если вы получили от поставщика пакет программ, то как определить, хороши или плохи приобретенные вами программные средства? При этом ваши интересы в разных случаях могут быть различными. Например, вас может интересовать не очень серьезная проблема выбора наилучшей из трех имеющихся программ для решения одной и той же задачи; однако она становится гораздо более серьезной, если такая программа является всего лишь частью — большой или малой — крупного проекта, который разрабатывается под вашим руководством, или системы программного обеспечения, приобретаемой на стороне. В обоих отмеченных случаях цели одинаковы — оценить качество программного продукта,— но ваши интересы и критерии оценки совершенно различны. Если вы пытаетесь выбрать лучшую программу из нескольких, центром вашего внимания может стать свойство мобильности, позволяющее определить, возможно ли использование этой программы в данном конкретном применении, либо свойства модифицируемости, понятности или точности. Если же рассматриваемый программный продукт является частью более крупного проекта, ваш интерес может быть сосредоточен вокруг проблемы постепенной управляемой эволюции программных средств. В частности, вас могут интересовать вопросы, касающиеся чувствительности программ к изменению требований, полноты и соответствия установленным нормативам и стандартам.

При разработке методологии оценки качества программного обеспечения это многообразие интересов пользователя должно стать определяющим фактором. Именно по этой причине невозможно предложить какую-либо единственную универсальную меру качества программных средств; здесь необходимо множество ха-

рактеристик, охватывающих целый спектр желаемых свойств. Каждая такая характеристика должна в свою очередь подразделяться на отдельные показатели, оценивающие ту или иную ее сторону. Выходом такой метрики может быть число (процент завершенности, процент ошибок, выявленных при тестировании и т. п.) либо качественное суждение, вынесенное человеком в одной из двух форм: в виде численной оценки или в виде решения типа «да-нет».

Процесс оценивания качества программного обеспечения начинается с выбора конкретных желаемых характеристик и установления соответствующей системы показателей, ряд из которых может потребовать сбора дополнительной информации о стандартах, приоритетах и т. п. После этого производится определение значений конкретных показателей иногда автоматически с помощью специальной программы или технического устройства, а иногда — человеком. Найденные значения показателей регистрируются и могут использоваться для вывода некоторой интегральной оценки достоинств программного обеспечения. Однако в связи с тем, что система выбранных показателей пока не охватывает всех аспектов той или иной характеристики, эта интегральная оценка скорее заставит задуматься, чем позволит дать окончательные выводы и рекомендации. Вероятнее всего, значения отдельных показателей должны анализироваться для обнаружения отклонений от номинальных или ожидаемых значений. При этом должны обнаруживаться всевозможные аномалии характеристик качества, которые следует классифицировать. Серьезные недостатки должны анализироваться и исправляться либо игнорироваться, если выяснится, что данный критерий неприменим к оцениваемым программным средствам, что бывает довольно часто. Например, низкий уровень понятности может быть допустим в случае высокоэффективной программы, работающей в реальном масштабе времени. В указанном смысле характеристики и их показатели являются детекторами аномалий качества программного обеспечения, и если обнаруживается значительное несоответствие принятым стандартам или обычной практике, то требуется дальнейшее

изучение природы обнаруженных отклонений и их влияния как на программные средства, так и на систему в целом.

В следующей главе приводятся и обсуждаются 11 возможных характеристик качества программного обеспечения.

ГЛАВА 3

Свойства качественного программного обеспечения

Если мы имеем какие-то программные средства, предназначенные для функционирования в определенных условиях, нас прежде всего волнуют три проблемы:

1. Возможность их использования в исходном виде.
2. Удобство эксплуатации.
3. Возможность применения в иных условиях.

Для решения первой проблемы необходимо:

а) понять, что способна делать программа,
б) получить достоверные результаты, выявить и устранить источники недостоверности и ненадежности (что перекликается со второй проблемой),
в) обеспечить рациональное и эффективное использование людских и машинных ресурсов.

Для решения второй проблемы необходимо:

а) понять структуру программы,
б) разработать и реализовать требуемые модификации,
в) провести тестирование в целях гарантирования надежной работы,
г) обеспечить рациональное и эффективное использование людских и машинных ресурсов.

Для решения третьей проблемы необходимо:

а) оценить исходную полезность программы (это приводит к первой проблеме),
б) оценить степень удобства эксплуатации (это приводит ко второй проблеме),
в) оценить мобильность,
г) приспособить программу к работе в новых условиях.

Идеальным представляется такой случай, когда от требуемых свойств, перечисленных в пунктах а), б), в) и г), можно было бы перейти к множеству характеристик имеющихся программных средств и с помощью этих

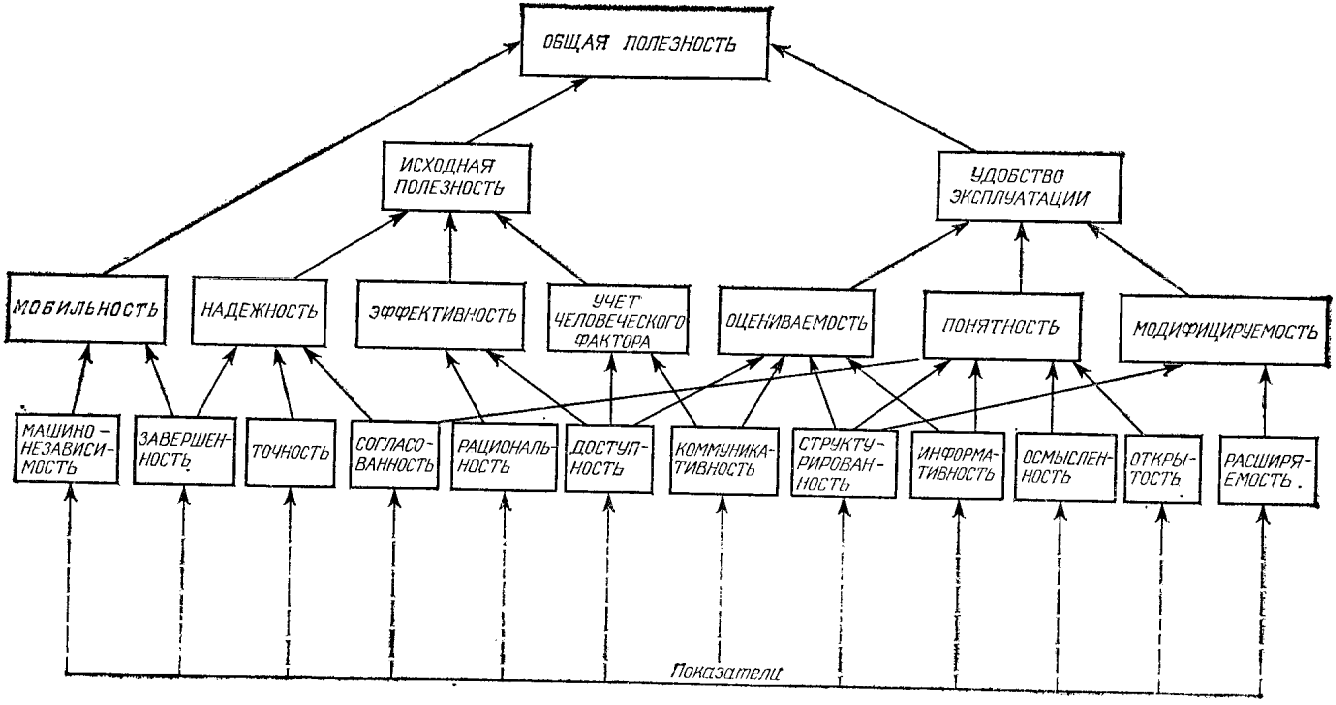


Рис. 3.1. Иерархическое дерево свойств программного продукта.

характеристик установить, в какой мере программное обеспечение обладает конкретными свойствами. Однако при реализации такого подхода возникает необходимость построения иерархической системы все более детализированных характеристик, в которой каждый вышестоящий уровень приближается к реальным нуждам пользователей, а каждый нижестоящий уровень постепенно ведет к получению численных оценок соответствующих свойств. Такое дерево характеристик программного обеспечения приведено на рис. 3.1.

Следует отметить, что множество характеристик, отображенное на рис. 3.1, не содержит каких-либо существенных изменений по сравнению с исходным набором, сформулированным перед началом детальных исследований. Единственное, что нам пришлось сделать в результате анализа, это разделить свойство полезности на две составляющие: исходную полезность, которая характеризует только пригодность программы для работы в исходном ее виде и на требуемой ЭВМ, и общую полезность, показывающую, в какой мере данная программа позволяет вносить изменения и использовать ее в условиях, отличных от исходных.

Ниже даются определения всех промежуточных и элементарных характеристик и приводятся примеры, иллюстрирующие те или иные свойства программ.

3.1. ПОНЯТНОСТЬ

Программное обеспечение обладает свойством ПОНЯТНОСТИ в той степени, в которой оно позволяет оценивающему лицу понять назначение программных средств.

Из этого определения следует, что человек, проводящий оценивание, должен иметь возможность проникнуть в смысл документации и принципов функционирования программного обеспечения, равно как и понять его взаимосвязи с другими программными средствами и подсистемами. Под этим определением подразумевается, что всякий программный продукт необходимо создавать с учетом нужд конечного пользователя, условий, оговоренных конкретным контрактом и т. п. Система

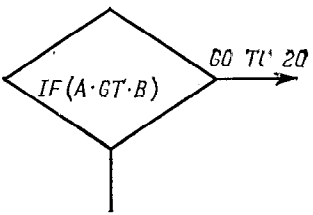
программного обеспечения понятна лишь в том случае, если она описана ясным и простым языком, свободным от жаргона и неадекватно определенных терминов или символов, и содержит необходимые ссылки на легкодоступные документы, позволяя читателю разобраться в сложных или новых элементах. В применении к блок-схемам алгоритмов и машинным программам свойство ПОНЯТНОСТИ означает четкость и аккуратность рисунков, расшифровку соответствующей символики, согласованное использование символов, адекватные комментарии или описания элементов диалога, одинаковое всюду в программах написание одних и тех же символических имен переменных и применение легко различимых имен (например, пару имен ABCD и ABCE следует признать неудачной, в то время как имена XVELOC и YVELOC уже более приемлемы). Приводимая ниже программа демонстрирует положительный пример ПОНЯТНОСТИ и обладает характеристиками информативности и открытости.

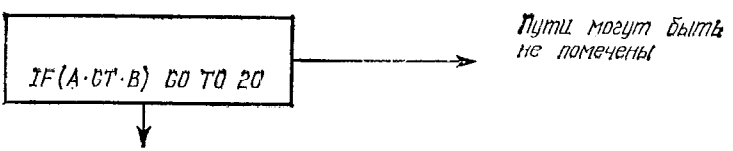
```
      SUBROUTINE ROOTS (A, B, C, POSROOT, NEG-
     ROOT, IFLAG)
C    ПРОГРАММА ВЫЧИСЛЯЕТ КОРНИ ПОЛИНОМА
     ВТОРОГО ПОРЯДКА
C    A*X*X+B*X+C=0. ВЫЧИСЛЕНИЕ КОРНЕЙ
     ПРОИЗВОДИТСЯ В БЛОКАХ
C    POSROOT И NEGROOT
C    ЕСЛИ КОРНИ КОМПЛЕКСНЫЕ, ТО ВЕЩЕСТ-
     ВЕННАЯ ЧАСТЬ ОПРЕДЕЛЯЕТСЯ
C    В БЛОКЕ POSROOT
C    А МНИМАЯ ЧАСТЬ В БЛОКЕ NEGROOT С УС-
     ТАНОВКОЙ ИНДИКАТОРА
C    IFLAG В СОСТОЯНИЕ 1. ЕСЛИ КОРНИ ДЕЙ-
     СТВИТЕЛЬНЫЕ,
C    ИНДИКАТОР IFLAG УСТАНАВЛИВАЕТСЯ В
     СОСТОЯНИЕ 0 DISCR=B*B—4.0*A*C
C    ПРОВЕРКА НА НАЛИЧИЕ КОМПЛЕКСНЫХ
     КОРНЕЙ
```

```
      IF (DISCR · LT · 0 · 0) GO TO 1
C     КОРНИ ДЕЙСТВИТЕЛЬНЫЕ
      IFLAG=0
      POSROOT= (—B+SQRT (DISCR))/ (2.*A)
      NEGROOT= (—B—SQRT (DISCR))/ (2.*A)
C     ВЫЧИСЛЕНИЕ ДЕЙСТВИТЕЛЬНЫХ КОРНЕЙ
      ЗАКОНЧЕНО
      RETURN
C     КОРНИ КОМПЛЕКСНО-СОПРЯЖЕННЫЕ
1     CONTINUE
      IFLAG=1
      POSROOT=—B/(2.*A)
      NEGROOT=SQRT (—DISCR)/ (2.*A)
C     ВЫЧИСЛЕНИЕ КОМПЛЕКСНЫХ КОРНЕЙ ЗА-
      КОНЧЕНО
      RETURN
      END
```

Отрицательными примерами ПОНЯТНОСТИ могут служить следующие разнородные изображения накопителей на магнитных лентах в разных местах блок-схем

или разветвления





Оба этих примера свидетельствуют об отсутствии свойства согласованности.

В заключение приведем еще пример плохо написанной программы, не обладающей свойствами ПОНЯТНОСТИ и сопутствующими характеристиками информативности и открытости:

```
      SUBROUTINE FIXUP (X3, X4, NINE)
      COMMON C(1000)
C     УКАЗЫВАЕТСЯ НЕОБХОДИМАЯ ВЕТВЬ ПРО-
      ГРАММЫ DOODAD
      IF (C(17)·GT·ABS(C(10)+NINE)
     1·AND·X3·LT·—1.90274·OR·
     2X4·EQ·1.111111)X/3=X4
      RETURN
      END
```

3.2. ЗАВЕРШЕННОСТЬ

Программный продукт обладает свойством ЗАВЕРШЕННОСТИ, если в нем присутствуют все необходимые компоненты, каждый из которых разработан всесторонне.

О документе говорят, что он завершен, если в нем присутствуют все элементы содержания, перечисленные в оглавлении, и это содержание с достаточной полнотой отражает аспекты функционирования системы, соответствующие всем другим характеристикам. Примером незавершенной программы может служить такая, в которой обнаружены неразрешенные внешние ссылки. Если база данных содержит архивные данные, то в документацию должно быть включено описание всех хранимых сведений. Кроме того, сама база данных не может быть признана завершенной, если она не защищена должным образом от потери информации и не предусматривает наличия необходимого количества резервных копий. Следовательно, ЗАВЕРШЕННОСТЬ предполагает замкнутость описания и живучесть программы.

Положительными примерами могут служить машинная программа со всеми разрешенными внешними ссылками, обеспечиваемая необходимыми входными данными, а также база данных, описание которой содержит все необходимые указатели, перекрестные ссылки, все требуемые данные и которая должным образом продублирована на стационарных запоминающих устройствах.

В качестве отрицательного примера ЗАВЕРШЕННОСТИ можно назвать любой документ, в котором отсутствует один или более разделов либо отмечено, что отсутствующие разделы будут написаны в дальнейшем. Второй отрицательный пример — машинная программа, содержащая неразрешенные внешние ссылки, ошибочные значения параметров, неполные спецификации входных данных, неправильные или неполные спецификации форматов данных и т. д. Третий отрицательный пример — блок-схема, на которой не отражены некоторые блоки команд или требуемые функции.

3.3. ОСМЫСЛЕННОСТЬ

Программный продукт обладает свойством ОСМЫСЛЕННОСТИ, если его документация не содержит избыточной информации.

Лишние фразы и повторы затемняют основную мысль и не позволяют сосредоточить внимание на важных подробностях. Это относится как к документации, так и к самим программам. Иногда требования к осмысленности и понятности могут противоречить друг другу, особенно если читатели не являются специалистами. Это легко показать на примере сравнения языков Фортран и АПЛ в аспекте ОСМЫСЛЕННОСТИ и Фортрана с Коболом в аспекте ПОНЯТНОСТИ.

Свойство ОСМЫСЛЕННОСТИ лучше всего иллюстрируется отрицательными примерами, в качестве которых можно привести:

— программу, содержащую в различных местах одну и ту же последовательность команд; такую последовательность команд следует описать в виде некоторой функции, подпрограммы или макрокоманды, что существенно уменьшит объем программы; макрокоманды используются тогда, когда предпочтительнее внутренние передачи управления, дополнительные обращения к подпрограммам и функциям;

— блок-схему, произвольным образом подразделенную на страницы с небольшим количеством блоков безотносительно к структуре программы и принципу ее работы;

— документацию, содержащую излишние повторения одних и тех же сведений в последующих разделах;

— программу, включающую излишне большое количество модулей, перекрытий, функций и подпрограмм, или программный комплекс, состоящий из большого количества коротких подпрограмм, вызываемых лишь какой-либо одной программой и не характеризующихся общностью использования.

3.4. МОБИЛЬНОСТЬ

Программный продукт обладает свойством МОБИЛЬНОСТИ, если он может легко и эффективно использо-

ваться для работы на ЭВМ иного типа, чем та, для которой он предназначен.

Свойство МОБИЛЬНОСТИ равнозначно свойству автономности, определяющему способность программных средств работать «на самообслуживании», без привлечения дополнительных программных ресурсов. Потребность в обеспечении МОБИЛЬНОСТИ зависит от конкретного проекта, и во многих случаях здесь должны сравниваться затраты и получаемый эффект. Может быть вполне оправданным назначение некоторого произвольно выбранного уровня мобильности, которому должно удовлетворять программное обеспечение, с определенными затратами, необходимыми для перехода к новым условиям функционирования в будущем.

Положительными примерами для свойства МОБИЛЬНОСТИ могут служить:

— документация, которая написана так, что для полного представления проблемы требуется минимальное число обращений к другой документации;

— программы, в которых используются библиотеки стандартных функций и подпрограмм универсального применения, аналогичные списки аргументов, выходные данные, требуемая точность и т. п.

— программы, минимально использующие операционную систему или вообще не использующие последнюю; к сильной зависимости от операционной системы приводит использование процедур управления заданиями, оверлейных структур, описаний наборов данных и операций ввода-вывода; в большинстве случаев изготовители ЭВМ не заботятся о стандартизации в этой области даже внутри собственного семейства машин; в еще меньшей степени можно надеяться на стандартизацию функций операционных систем разнотипных машин.

Приведенные положительные примеры одновременно демонстрируют характеристику завершенности программных средств.

Отрицательные примеры для свойства МОБИЛЬНОСТИ:

— какой-либо документ, в котором содержится ссылка на другой документ, несущий важную информацию;

часто оказывается, что документ, на который дается ссылка, на самом деле не имеет отношения к существу дела; иногда упоминаются внутренние или иные документы, доступ к которым затруднен, или документы, не подлежащие включению в данный проект;

— программа, использующая для достижения желаемого результата какую-либо специфическую особенность операционной системы; например, программисту может быть известно, что операционная система KLUDGEVAC 990 в случае обращения к ней без указания конкретной библиотеки, из которой должна быть вызвана необходимая программа, осуществляет вначале поиск в USERLIB, а затем в SYSLIB для разрешения внешних ссылок; программист может, опираясь на указанную особенность, организовать условный вызов конкретной программы, присвоив двум разным программам одно и то же имя и расположив их в двух разных библиотеках;

— программа, в которой используются специфические функции конкретного алгоритмического языка, не реализуемые иными способами, но легко осуществимые в других языках; например, неразумно без надобности использовать рекурсивные свойства Алгола, если в дальнейшем программа подлежит переводу на языки Кобол или Фортран;

— нарушение существующих стандартов МОБИЛЬНОСТИ, например использование более 32 бит 60-битового слова в тех случаях, когда требуется обеспечить совместимость программы с машинами серии IBM 360 или IBM 370.

3.5. СОГЛАСОВАННОСТЬ

Программный продукт обладает свойством ВНУТРЕННЕЙ СОГЛАСОВАННОСТИ, если он всюду содержит единую нотацию, терминологию и символику. Программный продукт обладает свойством ВНЕШНЕЙ СОГЛАСОВАННОСТИ, если можно проследить его соответствие требованиям.

Внутренняя согласованность пересекается с ПОНЯТНОСТЬЮ: очевидно, что согласованное применение системы обозначений, символов, терминов, имен переменных и программ способствует увеличению ПОНЯТНОСТИ. Кроме того, свойство внутренней согласованности является особым аспектом качества программного обеспечения, так как рассматривает программный продукт в относительной связи с другими частями создаваемой системы. Наличие несогласованности служит для руководителей проекта признаком возможного коммуникационного разрыва между разработчиками. В этом смысле внешняя согласованность программного продукта — характеристика совершенно иная: она определяет степень, в которой программный продукт может быть проверен на соответствие выдвинутым требованиям, спецификациям, ограничениям проектного или иного характера и т. п.

Положительные примеры:

— проектная документация, в которой каждый раздел или параграф непосредственно связываются с соответствующим пунктом технических условий;

— машинная программа, в которой каждая подпрограмма и функция в точности соответствуют блок-схемам, легко доступным для контроля в ходе проектирования;

— план испытаний, в котором каждая проверка соответствует некоторым пунктам технических условий на программу и требований к ее функционированию;

— документ, блок-схема или программа, в которых все символы, обозначения, термины, имена переменных и программ и т. п. либо самоочевидны для данной отрасли деятельности, либо специально определяются в составе документации программного продукта; если при этом определения разбросаны по всему тексту, то составляется словарь или указатель терминов или каждый новый термин подчеркивается при первом его появлении либо выделяется каким-то иным способом.

Что касается отрицательных примеров, то ими могут быть противоположные ситуации, в которых рассмотренные выше объекты не обладают свойствами ВНУТРЕННЕЙ или ВНЕШНЕЙ СОГЛАСОВАННОСТИ.

3.6. УДОБСТВО ЭКСПЛУАТАЦИИ

Программный продукт УДОБЕН В ЭКСПЛУАТАЦИИ, если предусматривается возможность его обновления в соответствии с новыми требованиями.

УДОБСТВО ЭКСПЛУАТАЦИИ предполагает ПОНЯТНОСТЬ, ОЦЕНИВАЕМОСТЬ и простоту внесения изменений, которая необходима для устранения обнаруживаемых дефектов, для добавления новых возможностей или приспособления программы к работе на другой ЭВМ.

Положительные примеры:

— документация, в которой страницы, рисунки и таблицы пронумерованы в соответствии с номерами разделов, т. е. 1.1, 1.2, 1.3, . . ., 2.1, 2.2, . . . и т. д., с тем чтобы добавления или исключения материала не вызывали необходимости перенумерации по всему тексту;

— программа, для которой строго документированы последовательности обращения к подпрограммам, использование общего блока и оверлейная структура, а также дан словарь имен переменных и списки перекрестных ссылок;

— программа, при создании которой *сознательно* учитывалась необходимость экономии ресурсов (оперативной, дисковой, ленточной, внешней памятей и т. п.), с тем чтобы обеспечить возможность последующих изменений.

Отрицательные примеры:

— программа, написанная как единое целое и документированная так, что отсутствует возможность создания оверлейных структур в целях обеспечения условий, облегчающих внесение изменений;

— программа, в которой оверлейная структура, последовательности команд обращения к подпрограммам и логика управления таковы, что модифицирование программы чересчур сложно;

— программа, работающая в реальном масштабе времени и тратящая 99,8 мс на обработку данных о событиях в 100-миллисекундном интервале времени;

— документ, в котором система нумерации затрудняет дополнения и исключения, таковым, например,

является документ, содержащий более пяти уровней градации разделов: если в подразделе 7.1.2.1.5.3.6.2 присутствует фраза типа «... прежде чем объединять компоненты, разделим их на элементы», то вслед за ней неизбежно должен появиться подраздел 7.1.2.1.5.3.6.2—«а»; в подобной системе нумерации весьма велика вероятность ошибок, а в случаях, когда требуется внести изменения более чем в один раздел, необходимо строгое соблюдение последовательности внесения изменений.

Приведенные выше положительные примеры говорят соответственно о наличии характеристик открытости, информативности и расширяемости, а отрицательные — об отсутствии характеристик информативности, структурированности, расширяемости и снова структурированности соответственно.

3.7. ОЦЕНИВАЕМОСТЬ

Программный продукт обладает свойством ОЦЕНИВАЕМОСТИ, если это свойство позволяет установить критерий приемлемости программного продукта для конкретного применения и обеспечивает возможность оценки качества функционирования программных средств.

Проблема обеспечения свойства ОЦЕНИВАЕМОСТИ имеет две стороны: 1) установление критерия и 2) разработку способа проверки соответствия программного продукта установленному критерию.

Например, программа, разделенная по функциональному признаку на модули, обладает свойством ОЦЕНИВАЕМОСТИ в большей степени, чем программа, в которой модули выделены случайным образом, скажем на основе ограничений, накладываемых объемом памяти. Программа, имеющая модульную структуру, дает возможность лицу, производящему ее оценивание, сконструировать множество последовательных тестов для каждого модуля. Это позволяет заблаговременно продумать сложную систему процедур тестирования. Благодаря такому подходу специалист, который не в состоянии сразу охватить все аспекты функционирования той или иной программы, имеет возможность изучить

ее посредством ряда последовательных проверок небольшого объема.

Свойство ОЦЕНИВАЕМОСТИ может относиться не только к программам, но и к блок-схемам. При наличии у последних такого свойства лицо, производящее оценивание, может осуществлять проверку блок-схем за рабочим столом, прослеживая логические пути алгоритмов. Если при этом удается достаточно глубоко понять суть программного продукта, то открывается возможность выявления неправильных логических переходов, ошибочных арифметических выражений, незамеченных условий возникновения ошибок и неадекватных или неполных сообщений о них.

Вторая сторона свойства оцениваемости состоит в обеспечении легкой тестируемости исходных программ. Возможность получения численных результатов диагностики или промежуточных результатов контроля может достигаться несколькими путями: например, наличие в 1-й колонке перфокарты символа X может указывать на операторы, которые подлежат исполнению только при диагностических или тестовых прогонах программы; еще один способ состоит в формулировании процедур выдачи результатов диагностики как макрокоманд, тогда возможно подавление этих команд при рабочих прогонах посредством одной управляющей перфокарты; наименее эффективный метод заключается в том, чтобы встраивать диагностические процедуры и результаты в основную программу на входном языке, не исключая их из готового программного продукта, что может выглядеть так:

```
IF(IDIAG.EQ.1)PRINT...
```

Положительные примеры для свойства ОЦЕНИВАЕМОСТИ:

— программа, разделенная на модули по функциональному принципу, в которой диагностические процедуры могут включаться либо исключаться по усмотрению пользователя в процессе условного компилирования;

— программа, содержащая дополнительный блок контроля правильности вычислений; например, блок определения полного энергетического баланса в системе

с постоянной энергией или блок, содержащий тригонометрическое тождество

$$\sin^2 x + \cos^2 x = 1$$

и используемый для проверки независимо вычисляемых $\cos x$ и $\sin x$ или в случае, когда $\cos x$ вычисляется специально ради контроля значения $\sin x$, единственно требующегося в дальнейшем.

Отрицательные примеры:

— программа, в которой понятные математические выражения или система уравнений затруднены для понимания или вообще скрыты от глаз из-за сложного стиля программирования; скажем, часть программы, заключенная между строками с номерами 50 и 86, может предназначаться для решения системы трех линейных уравнений целиком внутри этого блока; однако распознавание указанной цели по тексту программы может представлять большую трудность, а комментирующие карты могут не содержать пояснений в явном виде;

— программа, в которой используются одновременно и общие, и локальные переменные, а также перемежаются друг с другом переменные различных функций: в этом случае в программе трудно найти место для выдачи диагностической информации.

Оба положительных примера демонстрируют характеристику доступности; отрицательные примеры свидетельствуют в первом случае об отсутствии свойства информативности, а во втором — свойства структурированности.

3.8. ПОЛЕЗНОСТЬ

Программный продукт обладает свойством ПОЛЕЗНОСТИ, если он удобен для практического применения. Это свойство имеет две стороны:

1) Программы должны быть написаны так, чтобы была возможность их полного или частичного использования (в случае необходимости) в иных условиях. При этом речь идет не о свойстве МОБИЛЬНОСТИ, а об ответе на вопрос: полезна ли функция, выполняемая

данным программным продуктом, для других проектов? Например, если рассматривается информационно-справочная система, важно выяснить, может ли она использоваться не только для тех целей, которые ставились первоначально при ее создании.

2) Необходима надлежащая проработка вопросов обеспечения взаимодействия человека с машиной: должны быть четко определены входные параметры, форматы ввода данных, которые следует делать либо свободными, либо обеспечивающими определенную гибкость, например возможность вводить нулевые значения переменных, повторяющиеся значения или осуществлять серию последовательных прогонов при незначительных изменениях исходных данных. Описание выходов программ и базы данных должно быть ясным и легко интерпретируемым, позволяющим гибко определять формат и содержание результирующей информации.

Вопросы человеко-машинных интерфейсов важны даже тогда, когда программный продукт не отличается широтой использования и не обладает общей полезностью. Дело в том, что исходные данные готовятся людьми, результаты тоже должны читаться и интерпретироваться людьми и люди могут привлекаться к операциям ведения базы данных. В области сопряжения функций и машины существует широкий набор методов и средств, иногда они встречаются в языках высокого уровня, и большая часть этой работы ложится на разработчиков программного обеспечения.

Особенно благоприятные возможности для решения вопросов человеко-машинных интерфейсов представляют программы, предназначенные для работы с интерактивными терминалами типа графических устройств, проекционных дисплеев, индивидуальных пультов проектировщиков и т. п. Всюду, где такие терминалы существуют, они должны использоваться не только по основному назначению, но и для того, чтобы способствовать процессам подготовки исходных данных и представления результатов.

Положительные примеры:

— программа, предусматривающая свободный формат ввода или ввод списка имен в сочетании с контролем

допустимых значений в целях выявления ошибок; может быть и более сложная процедура контроля, предполагающая перекрестное сравнение различных величин: например, если входными данными служат сведения о потреблении энергии отдельными компонентами системы, то сумма частных расходов не должна превышать общего объема потребляемой энергии;

— программа, осуществляющая запись в память и извлечение из памяти какой-либо информации и предусматривающая алфавитно-цифровые метки для указания категорий и видов хранимых данных: так, программа, первоначально предназначенная для запоминания траекторий полета ракет, может быть легко приспособлена для применения в медицине в случае необходимости хранения данных электрокардиограмм и слежения за их изменениями;

— программа, которая выдает пользователю сообщения об ошибках на понятном ему языке, без применения специальных терминов вычислительной техники;

— программы обработки графической информации, вводимой с помощью светового пера, графического планшета или какого-либо другого интерактивного средства, ориентированного на человека и позволяющего последнему общаться с машиной в привычной манере.

Отрицательные примеры:

— программы, в которых пользователю могут выдаваться сообщения на машинном языке, скажем в середине имитационного моделирования системы массового обслуживания печатается следующее сообщение:

ARITHMETIC ERROR
OLD PSW — FFE0439DC100096

Такое сообщение совершенно бессмысленно для пользователя, даже если его содержание объясняется в инструкции.

— программа, колода карт которой содержит вперемежку целые и действительные числа различной разрядности, нанесенные на перфокарты бессистемно, без какой-либо группировки по смыслу или функциональному назначению;

— программа, результаты которой содержат лишь самое минимальное количество текстовой информации, позволяющей понять их смысл, и выходная информация которой, касающаяся процедур диагностики и тестирования, перемежается случайным образом с требуемыми результатами для пользователя.

Все приведенные в этом разделе примеры касаются одновременно характеристики коммуникативности программного обеспечения, за исключением второго положительного примера, демонстрирующего структурированность.

3.9. НАДЕЖНОСТЬ

Программный продукт обладает свойством НАДЕЖНОСТИ, если можно ожидать, что он будет удовлетворительно выполнять необходимые функции.

Обеспечение НАДЕЖНОСТИ предполагает получение ответов на следующие две группы вопросов:

1) Способен ли программный продукт удовлетворить выдвинутым требованиям к нему? Если программный продукт — программа, то достигается ли необходимая точность процедур ее трансляции, загрузки и выполнения?

2) При функционировании в реальных условиях продолжает ли программа работать правильно в случае исходных данных, существенно отличающихся от тестовых? Как много будет выявлено скрытых ошибок после аттестации программы как работоспособной? Какова вероятность того, что результаты будут содержать необнаруженные ошибки?

Положительные примеры для свойства НАДЕЖНОСТИ:

— программа, в которой запоминаются сведения об обнаруживаемых и устраняемых ошибках; впоследствии эти данные могут использоваться для оценки надежности программы;

— программа вместе с листингом трансляции, планом распределения памяти в результате загрузки и результатами проверки на контрольном примере.

Отрицательные примеры:

— программа, не содержащая какой-либо подпрограммы, к которой предусмотрено обращение;

— «самообслуживающаяся» программа, которую не удается загрузить, если в составе программного обеспечения нет специальной подпрограммы загрузки;

— использование на этапах тестирования и стыковки программы неадекватных данных для оценки ее надежности;

— программа, для которой оказывается недостаточным имеющийся объем памяти;

— документация, которая в процессе использования служит источником недостоверных сведений, т. е. либо явно неточна, либо не согласуется с другими документами;

— программа, выдающая неточные результаты;

— программа, в которой в процессе эксплуатации продолжают обнаруживаться все новые ошибки или дефекты.

Первый положительный пример демонстрирует наличие свойства завершенности, второй — наличие дополнительно свойств оцениваемости и информативности.

Отрицательные примеры свидетельствуют в трех первых случаях об отсутствии свойства завершенности, в двух следующих — об отсутствии точности и в последнем — тоже об отсутствии завершенности.

3.10. СТРУКТУРИРОВАННОСТЬ

Программный продукт обладает свойством СТРУКТУРИРОВАННОСТИ, если его взаимосвязанные части организованы в единое целое определенным образом.

Структурированность программы может иметь в своей основе самые различные причины: например, она может быть разработана в соответствии со специальными стандартами, определенными руководящими принципами и требованиями к интерфейсам, или она может быть написана с использованием языка структурного программирования, или может отражать в своей структуре процесс постепенного эволюционного развития на основе целенаправленных и систематизированных изменений.

СТРУКТУРИРОВАННОСТЬ не является обязательным свойством программного продукта. Стремление к высокой надежности программных средств, коротким срокам их разработки и низким затратам нашло свое выражение в тенденции к созданию средств автоматизации разработки программного обеспечения. Использование этих средств и методов приводит к созданию высокоструктурированных программ. Но эта технология еще находится на начальном этапе своего развития.

Положительные примеры для свойства СТРУКТУРИРОВАННОСТИ:

— программы и блок-схемы, соответствующие определенным стандартам;

— использование специальных правил именования переменных, функций и подпрограмм: например, программы, работающие в реальном масштабе времени, могут снабжаться именами, оканчивающимися буквой Р, в то время как программы, не критичные ко времени, могут получать имена, оканчивающиеся, скажем, буквой К;

— программы, написанные с применением методов структурного программирования и специальных программных модулей.

Отрицательные примеры:

— программы, написанные с нарушением установленных норм;

— программы, разбиение которых на модули не имеет под собой функциональной основы.

3.11. ЭФФЕКТИВНОСТЬ

Программный продукт обладает свойством ЭФФЕКТИВНОСТИ, если он выполняет требуемые функции без излишних затрат ресурсов.

Термин «ресурсы» здесь понимается в широком смысле: это может быть оперативная память, общее количество выполняемых команд на одну итерацию решаемой задачи или на один прогон, внешняя память, пропускная способность канала и т. п. Часто ЭФФЕКТИВНОСТЬ приобретается ценой ухудшения других характеристик, рассмотренных выше, так как нередко

является машинозависимой характеристикой и определяется свойствами конкретного используемого языка программирования. Обычно ЭФФЕКТИВНОСТЬ является антиподом МОБИЛЬНОСТИ. Например, программа, в которой не используются 28 бит 60-битового слова, как это требуется для обеспечения мобильности, не может считаться эффективной. В связи с тем, что программирование почти всегда выполняется с использованием языков высокого уровня, существуют специальные методы написания программ, позволяющие создавать входные программы, оптимизирующие работу транслятора. Например, последовательность операторов

```
DO 1 I=1,100
  1 A(I)+C(I)
```

не является оптимальной с точки зрения ее выполнения на машине типа CDC 7000. Эту последовательность действий следует писать в виде:

```
DO 1 I=1,100,2
A(I)=B(I)+C(I)
1 A(I+1)=B(I+1)+C(I+1)
```

что создает для транслятора возможность более эффективного использования аппаратных средств указанной машины.

Необходимость обеспечения эффективности за счет ухудшения других характеристик программного обеспечения должна особо отмечаться в задании на проектирование.

Лучше всего свойство ЭФФЕКТИВНОСТИ демонстрируется с помощью отрицательных примеров, в качестве которых можно назвать:

— программу, в которой формат А2 используется в операциях чтения, хранения и записи больших массивов текстовой информации; больший эффект могло бы дать использование формата А4 для машин серии IBM или формата А10 для машин серии: CDC;

— Фортран-программу, в которой имеются ненужные операторы DO, излишние операторы внутри циклов и нерациональные арифметические выражения, содер-

жащие переменные разных типов, например запись

```
X=4*Y
```

свидетельствует о неэффективности, тогда как выражение

```
X=4.0*Y
```

приводит к более простым операциям.

3.12. МАШИНОНЕЗАВИСИМОСТЬ

Программный продукт обладает свойством МАШИНОНЕЗАВИСИМОСТИ, если входящие в него программы могут выполняться на вычислительной машине иной конфигурации, чем та, для которой они непосредственно предназначены.

В качестве отрицательного примера можно назвать программу, использующую ФОРМАТ (12А6) для ввода текстовой информации и хранения ее в форме 6-символьных слов; дело в том, что ЭВМ серий IBM 360 и IBM 370, а также целый ряд других машин способны запоминать лишь 4-символьные слова.

Положительным примером может служить описание формата, учитывающее отмеченный выше факт:

```
FORMAT (18A4)
```

3.13. ТОЧНОСТЬ

Программный продукт обладает свойством ТОЧНОСТИ, если выдаваемые им результаты имеют точность, достаточную с точки зрения основного их назначения.

Отрицательный пример:

```
NDOLRS=CENTS1/100
NDOLRS=NDOLRS+CENTS2/100
```

Не говоря уже о нерациональности приведенного фрагмента программы и неподходящей форме записи математических действий, в данной программе при некоторых значениях переменных происходит нежелательная потеря целого доллара (например, если CENTS1=20, а CENTS2=81, то в результате NDOLRS=0). Для обес-

печения точности это выражение следовало бы записать так:

```
NDOLRS= (CENTS1+CENTS/2)/100
```

3.14. ДОСТУПНОСТЬ

Программный продукт обладает свойством ДОСТУПНОСТИ, если допускает селективное использование отдельных его компонент.

Отрицательным примером может служить запись:

```
GRVITY=1.40764548E16/R**2
```

Иногда может возникнуть необходимость в использовании иного значения гравитационной постоянной, чем то, которое указано в этой программе и относится к земному тяготению; скажем, может возникнуть потребность в замене этой постоянной характеристикой лунного притяжения в целях исследования траекторий движения вокруг Луны.

Следовательно, для обеспечения широкой доступности программы целесообразна следующая форма выражения:

```
GRVITY=GCONST/R**2
```

где переменной GCONST по умолчанию присваивается значение $1.40764548 \cdot 10^{16}$, но пользователь может задавать и иные ее значения.

3.15. КОММУНИКАТИВНОСТЬ

Программный продукт обладает свойством КОММУНИКАТИВНОСТИ, если он дает возможность легко описывать входные данные и выдает информацию, форма и содержание которой просты для понимания и несут полезные сведения.

Отрицательным примером может служить следующее сообщение об ошибке:

```
ERROR
OLD  PSW — FFE1237DC100224
```

В таком виде сообщение фактически не несет в себе диагностической информации, хотя программа могла бы ее обеспечить. Здесь требуется обращение к руководству по эксплуатации, которого может и не оказаться под рукой в нужный момент.

Положительный пример коммуникативности представляет собой сообщение следующего вида:

```
OUT-OF-BOUNDS REFERENCE NEAR END OF
SUBROUTINE QUECON USING IQ=257
```

в котором указывается, что в конце использования подпрограммы QUECON встречается обращение, параметр которого IQ=257 и выходит за допустимые границы.

3.16. ОТКРЫТОСТЬ

Программный продукт обладает свойством ОТКРЫТОСТИ, если его функции и назначение соответствующих операторов легко понимаются в результате чтения текста программы.

Отрицательный пример:

```
A=B**C/D**E*F/G*H
```

Положительный пример:

```
ANUM=GRAV*HGT*(BASE**CALIBR)
ADENOM=FORCE*(DENSTY**EXPDEN)
A=ANUM/ADENOM
```

3.17. ИНФОРМАТИВНОСТЬ

Программный продукт обладает свойством ИНФОРМАТИВНОСТИ, если он содержит информацию, необходимую и достаточную для понимания читающим лицом назначения программных средств, принятых допущений, существующих ограничений, исходных данных, результатов, отдельных компонентов и текущего состояния программ при их функционировании.

Положительным примером может служить подпрограмма SUBROUTINE ROOTS, рассмотренная выше в качестве положительного примера понятности, а отри-

цательным — приведенная в том же разделе подпрограмма SUBROUTINE FIXUP, служившая отрицательным примером того же свойства.

3.18. РАСШИРЯЕМОСТЬ

Программный продукт обладает свойством РАСШИРЯЕМОСТИ, если он позволяет увеличивать при необходимости объем памяти для хранения данных или расширять вычислительные функции отдельных модулей.

В качестве отрицательного примера можно привести программу, в которой описание структур данных построено так, что не остается возможности введения новых входных параметров. Положительным примером является такое описание структур входных данных, которое предусматривает добавление входных параметров.

3.19. УЧЕТ ЧЕЛОВЕЧЕСКОГО ФАКТОРА

Программный продукт учитывает ЧЕЛОВЕЧЕСКИЙ ФАКТОР, если он способен выполнять свои функции, не требуя излишних затрат времени со стороны пользователя, неоправданных усилий пользователя по поддержанию процесса функционирования программ и без ущерба для морального состояния пользователя.

Отрицательным примером является программа, в которой не предусмотрена предварительная обработка вводимых данных, позволяющая автоматически осуществлять их преобразование к виду, требуемому вычислительными модулями, вследствие чего соответствующие операции преобразования выполняются пользователями вручную.

Положительным примером может служить программное обеспечение, в составе которого для отмеченных выше целей имеется препроцессор.

3.20. МОДИФИЦИРУЕМОСТЬ

Программный продукт обладает свойством МОДИФИЦИРУЕМОСТИ, если он имеет структуру, позволяющую легко вносить требуемые изменения.

Отрицательный пример представляет собой программа, написанная крайне плотно, с применением упаковки нескольких переменных в одно слово, если нет необходимости экономить машинное время и память. В качестве положительного примера можно привести программу, в которой ограничения на память и машинное время менее жестки, данные упаковываются свободнее, их структура имеет необходимые свободные поля для облегчения будущих изменений, а программа имеет явно выраженную модульную структуру.

3.21. СВОЙСТВА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ И ИХ ЭЛЕМЕНТАРНЫЕ ХАРАКТЕРИСТИКИ

Анализ показателей качества программного обеспечения, соответствующих приведенным в разд. 3.1—3.11 определениям свойств программного продукта, позволил обнаружить многократное повторение одних и тех же показателей как измерителей нескольких различных свойств. Скажем, показатель «Наличие имен переменных» присутствовал в оценках ПОНЯТНОСТИ, ОЦЕНИВАЕМОСТИ, УДОБСТВА ЭКСПЛУАТАЦИИ и ряда других свойств. Причина этого состоит в том, что при прямом измерении рассмотренных свойств программного обеспечения наблюдается существенное перекрытие одного другим. Разрешить эту проблему удалось путем введения ЭЛЕМЕНТАРНЫХ ХАРАКТЕРИСТИК, показанных на нижнем уровне рис. 3.1. Показатели, относящиеся к этим характеристикам, всегда связаны только с какой-либо одной из них. В результате этого, например, показатель «Наличие имен переменных» был естественным образом отнесен к элементарной характеристике ИНФОРМАТИВНОСТИ.

Введение совокупности элементарных характеристик не приводит к нарушению правильности данных определений свойств программного продукта, а просто создает иерархическую структуру. Так, ИНФОРМАТИВНОСТЬ становится одним из аспектов ОЦЕНИВАЕМОСТИ, ПОНЯТНОСТИ и МОДИФИЦИРУЕМОСТИ, которые в свою очередь характеризуют различные стороны УДОБСТВА ЭКСПЛУАТАЦИИ, и эта цепочка нахо-

дится в полном соответствии с нашей трактовкой целей присвоения имен переменным. Из рис. 3.1 следует, что к ЭЛЕМЕНТАРНЫМ ХАРАКТЕРИСТИКАМ относятся 12 свойств, а 7 свойств представляют собой композицию выделенных элементарных характеристик.

ГЛАВА 4

Система показателей качества программного обеспечения

4.1. ВВОДНЫЕ ЗАМЕЧАНИЯ

По определению термин «показатель» характеризует некоторую стандартную меру. В нашем случае это мера той степени, в которой программный продукт обладает той или иной характеристикой и проявляет ее наличие (в аспекте качества, свойств или признаков).

Несмотря на то что целый ряд показателей, о которых будет идти речь, может, как это отмечалось ранее, найти непосредственное применение для оценки самых различных компонентов программных продуктов, например технических требований и планов испытаний, наше внимание было сосредоточено на разработке метрики свойств исходных модулей программ, написанных на языке Фортран и допускающих широкое использование карт с комментариями.

Какую же систему показателей качества программного обеспечения можно считать хорошей? Частично этот вопрос уже обсуждался в гл. 1, и здесь мы лишь резюмируем необходимые требования к такой метрике. Эти требования состоят в следующем:

— Каждый показатель должен быть коррелирован с установленными свойствами программного продукта, особенно с одной из характеристик, описанных в гл. 3. Это требование не тривиально, так как, например, показатель «средняя длина модуля» не всегда имеет определенную корреляцию со свойством структурированности.

— Каждый показатель должен быть существенным для оценки качества программного обеспечения, т. е. должны быть ясны потенциальные выгоды его использования.

— Должна существовать возможность количественного выражения показателей (автоматически или с помощью человека).

— Все показатели должны быть удобны с точки зрения создания автоматических средств их оценки, по возможности простых. Следует отметить, что невозможно ожидать одинаково хорошей корреляции всех показателей с соответствующими свойствами программного продукта, поскольку некоторые характеристики противоречат друг другу. Так, например, рациональность часто вступает в конфликт с машинонезависимостью, точностью, доступностью, коммуникативностью, структурированностью и расширяемостью (а также с завершенностью, если последняя предусматривает самофиксацию важных параметров работы программы и разумную степень сложности алгоритма). Осмысленность нередко несовместима с открытостью и иногда — с расширяемостью, а машинонезависимость — с точностью исходя из требуемой длины машинного слова. Более подробное обсуждение подобных конфликтных взаимоотношений характеристик программного обеспечения можно найти в работе Абернети [6], посвященной рассмотрению целей разработки операционных систем.

В следующем разделе приводится возможная система показателей, позволяющих оценивать качество программных средств, носящая форму вопросников, а в разд. 4.3 описывается алгоритм определения такого показателя, как информативность, и излагается способ его применения.

4.2. ОЦЕНОЧНЫЕ ТАБЛИЦЫ ПОКАЗАТЕЛЕЙ КАЧЕСТВА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Приводимые ниже таблицы получены в результате детального анализа исходного «потенциального» множества характеристик и отбрасывания тех из них, которые не удовлетворяли сформулированному выше набору критериев. Таблицы расположены в соответствии с порядком характеристик, установленным на рис. 4.1, а внутри каждой таблицы порядок следования показателей соответствует убыванию степени их важности для

Таблица 4.1

Показатели, подлежащие оценке для получения сведений об элементарной характеристике МАШИНОНЕЗАВИСИМОСТЬ (МН)

Шифры показателей	Сущность показателей, характеризующих МАШИНОНЕЗАВИСИМОСТЬ	Корреляция с качеством	Потенциальная выгодность	Возможность количественного выражения	Возможность автоматической оценки	Полнота автоматической оценки
МН-1 *	Обеспечена ли независимость точности вычислений и схем хранения данных от длины машинного слова?	В	5	АЛ + ПР + + КС	Л	Ч
МН-2 *	В случае если программирование выполнено на машинном языке (например, по соображениям более высокой эффективности), существует ли и доступна ли для использования какая-либо современная версия языка высокого уровня?	В	4	АЛ	Л	П
МН-3 *	Выделены ли в самостоятельную часть блоки программного обеспечения, использующие характерные особенности конкретной машины?	В	5	КО	Л	П

Продолжение

Шифры показателей	Сущность показателей, характеризующих МАШИНОНЕЗАВИСИМОСТЬ	Корреляция с качеством	Потенциальная выгодность	Возможность количественного выражения	Возможность автоматической оценки	Полнота автоматической оценки
МН-4 (!)	Снабжены ли метками и комментариями машинозависимые операторы (т. е. те вычисления, которые зависят от аппаратных возможностей ЭВМ по адресации полуслов, байтов, определенных комбинаций двоичных разрядов или по использованию расширенных средств входного языка программирования)?	В	5	АЛ	Д	Ч
МН-5	Характерен ли использованный язык программирования для ЭВМ нескольких типов?	В	5	НС		
МН-6	Допускает ли структура программы возможность ее последовательного выполнения на менее мощной ЭВМ, чем та, для которой программа непосредственно предназначена?	В	4	КС		

Таблица 4.2

Показатели, подлежащие оценке для получения сведений об элементарной характеристике ЗАВЕРШЕННОСТЬ (ЗВ)

Шифры показателей	Сущность показателей, характеризующих ЗАВЕРШЕННОСТЬ	Корреляция с качеством	Потенциальная выгодность	Возможность количественного выражения	Возможность автоматической оценки	Полнота автоматической оценки
ЗВ-1 *	Содержит ли программа средства, обеспечивающие очистку оперативной памяти перед началом прогона?	В	5	АЛ	Л	Ч
ЗВ-2 *	Содержит ли программа средства начальной настройки устройств ввода-вывода данных?	В	5	КО	Л	Ч
ЗВ-3 *	Содержит ли программа блоки, предусматривающие возникновение неопределенностей (например, при делении на нуль или извлечении квадратного корня из отрицательного числа), либо адекватные комментарии, описывающие условия, при которых появляется неопределенность?	В	5	АЛ	Л	Ч
ЗВ-4 *	Содержит ли программа внутри себя все требуемые подпрограммы, отсутствующие в обычной системной библиотеке?	ПВ	4	КО	Л	П

Продолжение

Шифры показателей	Сущность показателей, характеризующих ЗАВЕРШЕННОСТЬ	Корреляция с качеством	Потенциальная выгодность	Возможность количественного выражения	Возможность автоматической оценки	Полнота автоматической оценки
ЗВ-5 (!)	Имеются ли в программе контрольные точки, позволяющие осуществлять повторный запуск определенных блоков?	О	5	КО + КС	Л	Ч
ЗВ-6 (!)	Имеется ли в программе возможность присвоения значений по умолчанию неописываемым параметрам?	В	5	АЛ + КС	О	Ч
ЗВ-7 (!)	Проверяются ли исходные данные с целью выявления возможных ошибок по допустимому диапазону значений?	ПВ	5	АЛ + КС	Л	Ч
ЗВ-8 (!)	Осуществляется ли предварительная проверка параметров по диапазону значений до использования их для организации простых или вложенных циклов?	О	4	АЛ	Л	П
ЗВ-9 (!)	Осуществляется ли проверка индексов переменных перед использованием?	О	4	АЛ	Л	П
ЗВ-10 (!)	Все ли входные данные программы используются в процессе ее выполнения или объясняются соответствующими комментариями?	О	3	АЛ	Л	П

Продолжение

Шифры показателей	Сущность показателей, характеризующих ЗАВЕРШЕННОСТЬ	Корреляция с качеством	Потенциальная выгодность	Возможность количественного выражения	Возможность автоматической оценки	Полнота автоматической оценки
ЗВ-11 (!)	Отсутствуют ли в программе «запрещенные» имена (например, SQRT, ATAN и т. п.)?	В	3	КО	Л	П
ЗВ-12	Отсутствуют ли в программе «фиктивные» обращения к подпрограммам?	И	2	АЛ	Л	П
ЗВ-13	Исключена ли зависимость программы от библиотеки программ, имеющейся в конкретной вычислительной системе?	В	3	НС		
ЗВ-14	Допускает ли программа повторный прогон в случае ошибок, не приводящих к полной остановке системы?	В	5	КС		
ЗВ-15	Свободна ли программа от явных ошибок?	В	5	ПР + КС		

Таблица 4.3

Показатели, подлежащие оценке для получения сведений об элементарной характеристике СОГЛАСОВАННОСТЬ (СГ)

Шифры показателей	Сущность показателей, характеризующих СОГЛАСОВАННОСТЬ	Корреляция с качеством	Потенциальная выгодность	Возможность количественного выражения	Возможность автоматической оценки	Полнота автоматической оценки
СГ-1 *	Все ли спецификации наборов общих переменных (т. е. тех, которые появляются в двух и более подпрограммах) тождественны (например, снабжены меткой COMMON)?	ПВ	4	АЛ	Л	П
СГ-2 *	Всюду ли тип переменной (действительная, целая и т. д.) указан одинаково?	В	5	АЛ	Л	Ч
СГ-3 *	Все ли индексы, характеризующие размерность массива, указываются при обращениях к нему?	ПВ	3	АЛ	Л	П
СГ-4 *	Все ли индексируемые параметры присутствуют в операторах спецификации, которые идентифицируют параметр как массив определенной размерности?	В	3	АЛ	Л	П
СГ-5 (!)	Всюду ли в программе физические или математические константы представляются единым образом (например, не должно быть в одном месте представление числа π как 3,14159, а в другом как 3,1416)?	В	5	АЛ + КС	Л	Ч

Продолжение

Шифры показателей	Сущность показателей, характеризующих СОГЛАСОВАННОСТЬ	Корреляция с качеством	Потенциальная выгодность	Возможность количественного выражения	Возможность автоматической оценки	Полнота автоматической оценки
СГ-6 (!)	Согласована ли требуемая точность интегрирования или число итераций с количеством значащих разрядов во входных и выходных данных?	В	4	АЛ + КС	Д	Ч
СГ-7	Правильно ли используются функции конкретного типа?	В	5	КС		
СГ-8	Всегда ли используется одна и та же метка (мнемонический дескриптор) для идентификации одного и того же результата работы программы?	В	5	КС		
СГ-9	Всюду ли одинакова конструкция арифметических выражений, имеющих сходное функциональное назначение (т. е. не встречается ли в одном месте выражение Y = SQRT (X), а в другом Z = Q **.5)?	В	1	КО	Л	Ч
СГ-10	Исключено ли в программе использование одинаковых имен для различных по своему смыслу физических переменных, применяются ли в этих случаях разные имена?	В	4	КС		
СГ-11	Всюду ли в программе одни и те же физические величины встречаются под одинаковыми наименованиями переменных?	О	3	КС		
СГ-12	Имеют ли все элементы одного массива общее функциональное значение?	В	4	КС		

Таблица 4.4

Показатели, подлежащие оценке для получения сведений об элементарной характеристике РАЦИОНАЛЬНОСТЬ (РЦ)

Шифры показателей	Сущность показателей, характеризующих РАЦИОНАЛЬНОСТЬ	Корреляция с качеством	Потенциальная выгодность	Возможность количественного выражения	Возможность автоматической оценки	Полнота автоматической оценки
РЦ-1	Оптимальным ли образом сконструированы подпрограммы, к которым происходят частые обращения, с точки зрения скорости их выполнения?	О	4	ПР + КС		
РЦ-2	Всюду ли для указания порядка числа используются целые числа?	В	2	АЛ	Л	П
РЦ-3	Повторяются ли общие блоки в промежуточных результатах работы программы?	И	2	АЛ	Л	Ч
РЦ-4	Вводятся ли данные блоками или индивидуально?	О	2	АЛ	Л	Ч

Таблица 4.5

Показатели, подлежащие оценке для получения сведений об элементарной характеристике ДОСТУПНОСТЬ (ДС)

Шифры показателей	Сущность показателей, характеризующих ДОСТУПНОСТЬ	Корреляция с качеством	Потенциальная выгодность	Возможность количественного выражения	Возможность автоматической оценки	Полнота автоматической оценки
ДС-1 (!)	Доступно ли для пользователя включение по запросу дополнительных возможностей программы в отношении вычислений или выдачи результатов?	В	5	АЛ+КС	Д	Ч
ДС-2 (!)	Исключено ли из программы использование арифметических выражений со встроенными в них литералами, которые подвержены изменениям (например, CIRCUM = = 3.14*DIAM)?	В	5	АЛ+КС	Л	Ч
ДС-3 (!)	Допускает ли программа изменение режима использования ресурсов, например, путем применения массивов переменной длины?	В	4	КО+КС	Л	Ч
ДС-4 (!)	Допускает ли программа переменную точность вычислений?	В	4	КО+КС	Д	Ч
ДС-5	Допускает ли программа отключение ненужных входных данных, вычислений и выходных данных при различных возможных режимах?	В	4	АЛ+КС	Д	Ч

Таблица 4.6

Показатели, подлежащие оценке для получения сведений об элементарной характеристике КОММУНИКАТИВНОСТЬ (КМ)

Шифры показателей	Сущность показателей, характеризующих КОММУНИКАТИВНОСТЬ	Корреляция с качеством	Потенциальная выбросность	Возможность вычисления	Возможность автоматической оценки	Полнота автоматической оценки
КМ-1 *	Предусматривает ли программа выдачу всех входных данных с указанием всех меток по запросу?	В	5	ЛА+КС	Л	Ч
КМ-2*	Способна ли программа распознавать конец входных данных без возложения на пользователя обязанностей подсчитывать и указывать их количества (т. е. может ли она различать форматированные поля, символы, значения переменных, признак конца файла и т. п.)?	В	5	КО	Л	Ч
КМ-3 *	Содержит ли программа средства идентификации тестов и/или описания всех результатов тестирования?	В	3	КО	Л	П
КМ-4 (!)	Предусмотрена ли в программе выдача четких и полезных сообщений об ошибках в случае их возникновения?	В	5	КО+НС	Д	Ч
КМ-5 (!)	Предоставляет ли программа средства, позволяющие прогонять повторяющиеся тесты без излишнего описания неизменяющих значений входных данных?	В	4	КО+КС	Л	Ч

Продолжение

Шифры показателей	Сущность показателей, характеризующих КОММУНИКАТИВНОСТЬ (КМ)	Корреляция с качеством	Потенциальная выгодность	Возможность количественного выражения	Возможность автоматической оценки	Полнота автоматической оценки
КМ-6 (!)	Предусматривается ли в программе выдача промежуточных результатов по запросу?	В	5	АЛ+КС	Л	Ч
КМ-7 (!)	Все ли выдаваемые результаты снабжены описанием «шапок»?	В	5	АЛ+КС	Л	Ч
КМ-8 (!)	Способна ли программа воспринимать неформатированные данные (например, без вычисления номера колонки перфокарты)?	О	4	АЛ	Л	П
КМ-9 (!)	Содержит ли программа средства трассировки и отображения логики передач управления?	В	5	КО+КС	Л	Ч
КМ-10	Спроектированы ли основные формы выдачи результатов так, что по ним можно судить в общем виде о результатах прогона тестов?	В	5	ПР+КС		
КМ-11	Обеспечивает ли программа нумерацию страниц выдаваемых документов для облегчения их визуального анализа?	В	3	ПР+КС		

Таблица 4.7

Показатели, подлежащие оценке для получения сведений об элементарной характеристике СТРУКТУРИРОВАННОСТЬ (СТ)

Шифры показателей	Сущность показателей, характеризующих СТРУКТУРИРОВАННОСТЬ	Корреляция с качеством	Потенциальная выгодность	Возможность количественного выражения	Возможность автоматической оценки	Полнота автоматической оценки
СТ-1 *	Установлены и соблюдаются ли в программе стандартные формы передач управления между модулями?	В	5	АЛ	Л	Ч
СТ-2 *	Существует ли ограничение на размер модуля?	ПВ	4	АЛ	Л	Ч
СТ-3 *	Соблюдается ли порядок расположения, при котором первым идет начальный блок комментариев, затем описывающие операторы, а потом программа, подлежащая выполнению?	В	3	КО	Л	П
СТ-4 *	Имеется ли во всех подпрограммах хотя бы одна точка выхода?	ПВ	4	АЛ	Л	П
СТ-5 *	Имеют ли все подпрограммы и функции единственную точку входа?	В	4	АЛ	Л	П

Продолжение

Шифры показателей	Сущность показателей, характеризующих СТРУКТУРИРОВАННОСТЬ	Корреляция с качеством	Потенциальная выгодность	Возможность количественного выражения	Возможность автоматической оценки	Полнота автоматической оценки
СТ-6 (!)	Развивается ли выполнение программы от начала к концу с комментированием всех случаев отклонений от правила?	ВП	4	АЛ	Д	Ч
СТ-7	Соответствует ли оверлейная структура последовательности выполнения подпрограмм?	В	5	ЭК		
СТ-8	Соответствует ли разделение программы на модули четко распознаваемому их функциональному назначению?	В	5	КС		
СТ-9*	Написана ли программа с использованием стандартных конструктивных элементов, предусматриваемых конкретным языком программирования?	ПВ	4	КО	Л	Ч

Таблица 4.8

Показатели, подлежащие оценке для получения сведений об элементарной характеристике ИНФОРМАТИВНОСТЬ (ИН)

Шифры показателей	Сущность показателей, характеризующих ИНФОРМАТИВНОСТЬ	Корреляция с качеством	Потенциальная выгодность	Возможность количественного выражения	Возможность автоматической оценки	Полнота автоматической оценки
ИН-1 *	Содержит ли каждый программный модуль (под которым понимается функция, подпрограмма, операторная функция или блок программы со входом от пар операторов ASSIGN GO TO) начальный блок комментариев, описывающий имя программы, дату ввода в действие, требуемую точность, назначение, ограничения и условия применения, хронологию модификаций, входные и выходные данные, метод решения задачи, принятые допущения, процедуры восстановления счета для всех предвидимых ошибок?	В	5	КО	Л	Ч
ИН-2 (!)	Адекватно ли описаны логические блоки алгоритма и соответствующие его альтернативные ветви?	В	5	КО+НС	Л	Ч
ИН-3 (!)	Достаточно ли адекватно описание входных и выходных данных, функционального назначения модулей, чтобы выполнять их тестирование?	В	5	КО+КС	Л	Ч
ИН-4	Имеются ли в программе комментарии, поясняющие выбор каких-либо конкретных значений входных переменных для выполнения специального тестирования программы?	В	4	КС		

Продолжение

Шифры показателей	Сущность показателей, характеризующих ИНФОРМАТИВНОСТЬ	Корреляция с качеством	Потенциальная выгодность	Возможность количественного выражения	Возможность автоматической оценки	Полнота автоматической оценки
ИН-5	Имеется ли информация, позволяющая оценить влияние на данный модуль изменений, вносимых в другие модули?	В	5	КС		
ИН-6	Имеется ли в программе информация, позволяющая определить, какой конкретно блок необходимо модифицировать для внесения требуемого изменения?	В	5	КС		
ИН-7	В тех случаях, когда модули зависимы, четко ли это отображено в комментарии, программной документации или структурной схеме программы?	В	5	КС		
ИН-8	Несут ли имена, присвоенные переменным, информацию о представляемых или физических свойствах или функциях?	В	3	КС		
ИН-9	Содержат ли «непонятные» функции достаточно информации для понимания их значения?	В	3	КС		
ИН-10	Достаточна ли информация для понимания связи имен переменных с отображаемыми ими свойствами или объектами?	В	3	НС		
ИН-11	Понятно ли значение «рудиментарных» блоков программы?	В	1	КС		

Таблица 4.9

Показатели, подлежащие оценке для получения сведений об элементарной характеристике ОСМЫСЛЕННОСТЬ (ОС)

Шифры показателей	Сущность показателей, характеризующих ОСМЫСЛЕННОСТЬ	Корреляция с качеством	Потенциальная выгодность	Возможность количественного выражения	Возможность автоматической оценки	Полнота автоматической оценки
ОС-1 *	Все ли части программы реализуемы при ее прогоне?	В	3	АЛ	Л	П
ОС-2 (!)	Выполняются ли вычисления, не относящиеся к циклу, вне его?	В	4	АЛ	Д	Ч
ОС-3 (!)	Исключено ли излишнее повторение составных арифметических выражений?	В	3	АЛ	Д	П
ОС-4 (!)	Имеет ли сопутствующий описательный комментарий низкий «индекс непонятности», обсуждаемый в работе [30]?	О	4	АЛ	Л	П
ОС-5	Всегда ли передача управления осуществляется операторам, снабженным метками и легко ли локализуется точка, в которую передается управление?	О	2	КО	Л	П

Продолжение

Шифры показателей	Сущность показателей, характеризующих ОСМЫСЛЕННОСТЬ	Корреляция с качеством	Потенциальная выгодность	Возможность количественного выражения	Возможность автоматической оценки	Полнота автоматической оценки
ОС-6	Отсутствуют ли случаи использования в программе многомерных массивов, когда можно ограничиться несколькими массивами меньшей размерности?	ПВ	3	КС		
ОС-7	Отсутствуют ли в программе специальные или ненужные форматы ввода-вывода? Если они все же есть, то могут ли вместо них подставляться другие существующие форматы без потери информации [например, использовать FORMAT (I10) и FORMAT (2I10)]?	О	3	АЛ+НС	Д	Ч
ОС-8	Всегда ли используются бинарные логические конструкции, если выражения перехода могут быть определены только как true или false?	ПВ	2	КО	Л	Ч

Таблица 4.10

Показатели, подлежащие оценке для получения сведений об элементарной характеристике ОТКРЫТОСТЬ (ОТ)

Шифры показателей	Сущность показателей, характеризующих ОТКРЫТОСТЬ	Корреляция с качеством	Потенциальная выгодность	Возможность количественного выражения	Возможность автоматической оценки	Полнота автоматической оценки
ОТ-1 *	Используются ли отступы, пустые строки, столбцы или цепочки крестиков и звездочек для выделения самостоятельных частей программы?	В	4	КО	Л	Ч
ОТ-2 *	Используются ли скобки или какие-то иные средства для устранения неопределенности в отношении порядка действий при вычислении значений арифметических выражений или характера этих действий (над действительными или целыми числами, с обычной или удвоенной точностью)?	В	3	АЛ	Л	Ч
ОТ	Располагаются ли метки операторов в алфавитном порядке?	В	4	АЛ	Л	ПП
[illegible] *	Сведены ли к минимуму, либо устранены совсем трудности поиска по листингу всех операторов ветвления, которые могут передавать управление оператору с данной меткой?	В	4	КО	Л	П

Продолжение

Шифры показателей	Сущность показателей, характеризующих ОТКРЫТОСТЬ	Корреляция с качеством	Потенциальная выгодность	Возможность количественного выражения	Возможность автоматической оценки	Полнота автоматической оценки
ОТ-5 *	Во всех ли модулях программы согласованы формат и порядок следования операторов спецификации?	В	4	КО	Л	Ч
ОТ-6 *	Во всех ли модулях программы согласованы форматы комментариев?	В	3	КО	Л	Ч
ОТ-7 *	Всегда ли осуществляется в одной строке программы только одно присваивание значения переменной?	О	3	АЛ	Л	П
ОТ-8	Содержится ли в каждой строке программы только один исполняемый оператор?	О	3	АЛ	Л	П
ОТ-9	Имеется ли какое-либо правило присвоения меток функционально различным блокам программы?	В	3	КС		
СТ-10	Выделены ли смежные общие блоки, снабженные метками, путем соответствующего пространственного расположения комментариев?	В	1	АЛ	Л	П

получения информации о соответствующей характеристике, отнесенной к затратам на получение такой информации. Звездочкой в таблицах отмечены те показатели, которые имеют одну из следующих высоких оценок:

— по крайней мере ПВ с точки зрения корреляции с качеством программного обеспечения;
— по крайней мере ранг 3 исходя из потенциальных выгод использования;
— ранг АЛ или КО в отношении возможности количественного представления;
— ранг Л в отношении легкости автоматического получения оценки;
— по крайней мере ранг Ч по степени полноты автоматической оценки.

Восклицательным знаком в скобках отмечены показатели, которые удовлетворяют почти всем этим критериям одновременно.

Примеры наличия или отсутствия соответствующих показателей качества у программ приведены в приложении А.

Отсутствие информации в двух последних столбцах приводимых ниже таблиц свидетельствует о невозможности автоматизации процедур получения оценок соответствующих показателей.

(Данные по характеристикам ТОЧНОСТЬ и РАСШИРЯЕМОСТЬ, содержащие всего по одному показателю, приведены в табл. 1.1.)

4.3. РАЗРАБОТКА АЛГОРИТМИЧЕСКИХ МЕТОДОВ ОЦЕНКИ КАЧЕСТВА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В предыдущем разделе был проведен целый ряд полезных вопросников, которые хотя и не являются исчерпывающими, но позволяют судить о качестве программного продукта, определенно характеризуя конкретные его свойства. Была также установлена связь рассмотренных показателей с конечной оценкой полезности и установлена их относительная важность, а также оценена степень легкости автоматического получения

ответов на указанные в таблицах вопросы в количественной форме. Следующим шагом должна быть разработка детальных алгоритмов поиска ответов на сформулированные вопросы и определение способов использования конкретных результатов такой количественной оценки для организации разработки программных средств и вынесения суждений об их качестве с точки зрения целесообразности приобретения. Хотя разработка значительной части подобных алгоритмов не представляет большого труда, к сожалению, жесткие рамки проводившихся нами исследований не позволили осуществить эту работу полностью. Поэтому ниже детально рассматривается лишь алгоритм, касающийся одного из показателей, приведенных в разд. 4.2, а именно показателя информативности (ИН-1), связанного с начальным блоком комментариев.

4.3.1. Программные изделия, связанные с программами для ЭВМ

Программа для ЭВМ представляет собой нечто большее, чем просто колода перфокарт или бобина магнитной ленты. Это по существу идея, отлитая в форму машинной программы, и, для того чтобы ее успешно применять, пользователю необходимо знать о программе все подробности. Отсюда возникает потребность в таком документе, как ИНСТРУКЦИЯ для ПОЛЬЗОВАТЕЛЯ или ОПИСАНИЕ ПРОГРАММЫ. Поскольку очень редко приходится встречаться с программами, не нуждающимися ни в каких изменениях для приспособления их к конкретному, имеющемуся у пользователя типу ЭВМ, то дополнительно требуется еще и такая документация, как функциональная блок-схема, блок-схема исходной программы и программа-эталон. Необходимо также наличие описаний оверлейной структуры и отдельных этапов выполнения программы, форматов наборов данных и требований, касающихся инициализации программы; распределения устройств ввода-вывода, схем загрузки и трансляции, карт управления заданиями. Последние виды программных изделий совсем не сложны и потому не требуют применения какой-либо

метрики для оценки их качества. Они либо бывают сильно зависящими от конкретного оборудования, либо автоматически генерируются операционной системой или транслятором языка, на котором написана исходная программа.

Таким образом, программное изделие включает по меньшей мере четыре элемента программного обеспечения:

1) программу на входном языке,
2) описание программы,
3) блок-схемы,
4) эталонную программу.

Ниже приводятся результаты разработки алгоритма для определения величины показателя ИН-1, характеризующего исходную программу.

4.3.2. Принципы разработки алгоритмов оценки показателей

Как отмечалось в гл. 1, основная ценность анализа исходной программы, написанной на входном языке, для выявления показателей качества программного обеспечения заключается не в получении каких-то численных параметров, а в использовании этих показателей в качестве индикаторов дефектов программы, которые подлежат дальнейшему изучению и исправлению. Таким образом, хотя описываемые ниже алгоритмы и дают на выходе некоторые численные характеристики, основное их назначение состоит в обнаружении дефектов.

Обсуждаемый далее показатель НАЧАЛЬНЫЙ КОММЕНТАРИЙ в табл. 4.8 охарактеризован как допускающий частичную автоматическую оценку. Обычно степень автоматизации, к которой следует стремиться при разработке алгоритмических и программных средств оценки исходных программ, бывает нелегко определить заранее. Поэтому ниже принят широко распространенный практический подход, при котором имеют место полуавтоматические процедуры: вначале определяются формальные операции, приводящие к накоплению опыта на небольших выборках программ с ручной обработкой результатов, а затем рассматривается вопрос о возмож-

ности и целесообразности автоматической оценки соответствующих параметров. При этом устанавливается, какие из процедур пригодны и готовы для автоматизации, а какие требуют дальнейшей ручной работы с информацией для уточнения этой возможности.

4.4. ПРИМЕР ДЕТАЛЬНОЙ РАЗРАБОТКИ МЕТРИКИ ПОКАЗАТЕЛЯ ИН-1

4.4.1. Сущность показателя

Оценка показателя ИН-1 предполагает получение ответа на следующий вопрос: содержит ли каждый программный модуль (под которым понимается функция, подпрограмма, оператор-функция или блок программы со входом от пар операторов ASSIGN GO TO) начальный блок комментариев, описывающий

1) имя программы,
2) дату ввода в действие,
3) требуемую точность,
4) назначение,
5) ограничения и условия применения,
6) хронологию модификаций,
7) входные и выходные данные,
8) метод решения задачи,
9) принятые допущения,
10) процедуры восстановления счета при всех возможных и предвидимых ошибочных входных данных.

4.4.2. Алгоритм

1а. Выделить в явном виде модули программы: функции и подпрограммы.

1б. Выделить неявные программные модули: блоки, вход в которые осуществляется из разных точек.

2. Проверить, предшествует ли каждому модулю совокупность карт с комментариями, имеющих стандартный формат для дескриптивных элементов (например, порядковые номера в третьей и четвертой колонках).

3. Для каждого описываемого элемента проверить, соответствует ли текст описания требуемому и отвечает ли его содержание установленным принципам, например:

— количество входных и выходных данных, указанное между запятыми, должно быть не меньше числа элементов в описании параметров подпрограммы или функции;

— назначение объекта должно описываться максимум в три строки;

— в исходных данных и выдаваемых результатах в скобках должны указываться блоки, к которым они относятся.

3б. По каждому элементу оценить понятность и полезность комментария.

4а. Если требуемый элемент отсутствует, присвоить ему оценку нуль.

4б. Для каждого присутствующего элемента установить конкретную численную оценку, исходя из следующей шкалы:

0 — элемент есть, но качество его неприемлемо;

0,5 — элемент присутствует и обладает приемлемым качеством;

1,0 — элемент присутствует и обладает очень высоким качеством.

Возможно присвоение промежуточных значений в соответствии с мнением оценивающего лица относительно полезности того или иного элемента для понимания сути модуля и работы с ним. После анализа всех элементов данного модуля их оценки суммируются и полученное значение фиксируется. Если рассмотренный модуль — не последний, то вернуться к пункту 2, в противном случае перейти к пункту 5.

5. Запомнить число модулей, на чем и кончается работа алгоритма.

4.4.3. Выдаваемые результаты

Результатом работы алгоритма является:

1. Перечень всех описываемых элементов с оценкой 0 с указанием причин такой оценки.

2. Количество нулевых оценок по типам элементов.

3. Гистрограмма или аналогичная ей табличная форма, показывающая распределение элементов по ин-

тервалам оценок (например, с границами 0; 0,2; 0,4; 0,6; 0,8; 1,0).

4. Среднее значение оценки качества модуля по данному показателю:

$$\text{СОМ} = \frac{\sum (\text{Оценка элемента})\,(\text{Вес элемента})}{\text{Общее число модулей}}.$$

Веса устанавливаются таким образом, чтобы их сумма равнялась единице.

4.4.4. Возможности автоматизации процедур оценивания

Шаг 1а автоматизируется легко посредством простого сканирования программы для выявления операторов, начинающихся со слов SUBROUTINE или FUNCTION. Шаг 1б хотя и с большими трудностями, но тоже может быть автоматизирован с использованием структурного анализатора, подобного программе NODAL фирмы TRW. Шаг 2 предполагает простой анализ текста, но могут быть исключения из этого правила при появлении неподходящих элементов, нарушений порядка номеров карт с комментариями, а также в случаях попадания в соответствующие колонки перфокарт цифровой информации комментария вместо порядкового номера карты. Шаг может быть автоматизирован, но с некоторым риском, поскольку, например, в некоторых описаниях возможно использование в качестве разделителей не запятых, а каких-либо иных знаков пунктуации. Шаг 3б пока что автоматизировать невозможно. Операции шага 4а поддаются автоматизации. Шаг 4б можно осуществить на базе контрольных вопросников и руководящих принципов, но не иначе.

Большая часть оценок, полученных на первых порах вручную, должна способствовать выработке принципов, которые в дальнейшем образуют основу для автоматизации процесса оценивания. А когда этот процесс автоматизирован, автоматизация процедур подсчета и выдачи результатов не составит никакого труда.

ГЛАВА 5

Руководящие принципы разработки качественного программного обеспечения

В этой главе рассматриваются проблемы, возникающие в связи с необходимостью внедрения обсуждавшейся в гл. 4 метрики в практику проектирования и реализации программных средств. Разд. 5.1 посвящен поиску метода такого внедрения; в нем показано, как руководящие принципы или детальные вопросники могут использоваться разработчиками, руководителями разработок и пользователями программного обеспечения для того, чтобы их программные изделия в большей степени обладали желаемыми характеристиками.

Введение дополнительного процесса оценивания программного изделия может потребовать и дополнительных «накладных расходов», касающихся календарного времени разработки, количества встреч разработчиков друг с другом для согласования проектных решений, машинного времени, документации, планирования разработок и т. д. Однако эти дополнительные расходы окупаются благодаря уменьшению частоты контрольных проверок, меньшему количеству ошибок, обнаруживаемых при системных испытаниях программных средств, а следовательно, и меньшему количеству изменений в программах документации, что сокращает число повторных испытаний и обеспечивает более высокую эксплуатационную надежность программного обеспечения. Понимание проблем и способов реализации качественных программных средств на всех уровнях их разработки и тщательное планирование этого процесса может гарантировать чистую экономию издержек по сравнению с такой практикой, когда рекомендуемые принципы обеспечения высокого качества программного обеспечения не используются.

В некоторых случаях, особенно на этапе программирования, не ограничивающемся, однако, простым кодированием программы, средства автоматического обнаружения дефектов программного обеспечения столь же необходимы, сколь и эффективны. В разд. 5.2 рассматриваются некоторые из таких средств, применяемые в фирме TRW, либо находящиеся в стадии разработки. В результате проведенных нами исследований выявилась необходимость в целом ряде новых средств автоматизации и были определены некоторые из них.

В разд. 5.3 анализируется зависимость затрат и выгод от использования метрики программного изделия и ее роль в создании программных средств, а также приводится простая аналитическая модель, используемая для анализа реальных данных об ошибках, допущенных в конкретном проекте военного назначения.

5.1. ПРИМЕНЕНИЕ РУКОВОДЯЩИХ ПРИНЦИПОВ РАЗРАБОТКИ ПРОГРАММНОГО ИЗДЕЛИЯ НА РАЗЛИЧНЫХ СТАДИЯХ ЕГО СОЗДАНИЯ

В этом разделе даются рекомендации относительно целесообразных принципов создания программного изделия на каждой фазе его разработки, которые представлены в форме вопросников или перечней необходимых действий на данной фазе. Основной целью этих руководящих принципов должно быть предотвращение или сокращение числа возможных ошибок, дефектов или других случаев несоответствия программного изделия выдвигаемым требованиям и проникновения их в поток программной продукции. Вторая цель состоит в том, чтобы повысить экономическую эффективность процесса контроля программного изделия на соответствие техническим условиям. Если дополнительные затраты, связанные с более тщательным анализом и оценкой качества, управленческим контролем и т. п. и необходимые для достижения основной цели, компенсируются сокращением издержек на контрольные проверки, то при заданном уровне затрат будет всегда иметь место чистый выигрыш в надежности программного обеспечения.

В ряде случаев этапы, выделяемые в данном разделе, представляют собой совокупность нескольких видов деятельности, обсуждавшихся в гл. 2. Мы рассматриваем здесь следующие этапы:

1. Этап выработки требований, объединяющий в себе ранее выделенные этапы установления требований к системе и ее программному обеспечению.
2. Этап проектирования, включающий как техническое, так и рабочее проектирование.
3. Этап кодирования и отладки, ничем не отличающийся от рассматривавшегося в гл. 2.
4. Этап усовершенствования и системных испытаний.
5. Этап ввода в действие и эксплуатации.

5.1.1. Руководящие принципы этапа выработки требований

Одной из главных причин ошибок, нарушения ограничений, отказов и неэффективности программных средств является отсутствие формальных методов анализа требований к ним и проектирования.

На эту проблему существует по крайней мере две равноправные точки зрения:

1. Процессуальная, в соответствии с которой система рассматривается как совокупность взаимодействующих операций со своим множеством характеристик входов, выходов, процессов и условий функционирования.
2. Качественная, в соответствии с которой отправными пунктами для выработки требований к проектированию являются такие общие свойства системы, как защищенность от несанкционированного доступа, надежность, удобство эксплуатации, а также ограничения на количество персонала и машинные ресурсы и т. п.

Большинство применяемых в настоящее время подходов к решению проблемы выработки требований к системе опирается на процессуальную точку зрения, вследствие чего получаются результаты, пока что в недостаточной степени позволяющие гарантировать полное удовлетворение выдвигаемым требованиям относительно желаемых свойств программного обеспечения. Однако фирмой TRW предпринято исследование с целью

расширения ранее разработанных формализованных методов проектирования программных средств — матрицы «Требования — свойства» и связанных с ней «Технических условий» (примеры показаны в табл. 5.1 и 5.2) — и объединения их в систему с другими методами анализа требований, применяемыми в рамках фирмы [1]) или вне ее. Основная задача этого проекта — разработка автоматических средств контроля требуемых свойств, проверки соответствия техническим условиям и полноты реализации необходимых функций [25].

Самое главное достоинство матрицы «Требования — свойства» заключается в том, что она заставляет разработчика размышлять над способами реализации каждого из требований исходя из свойств, обеспечиваемых системой, по крайней мере в течение времени, необходимого для занесения в матрицу соответствующей записи. В табл. 5.1 символ «А» означает, что соответствующие аспекты требуемого свойства проанализированы и признаны приемлемыми для реализации (как, например, аспекты машинонезависимости, связанные с хранением информации). Символ «П» означает, что соответствующий аспект подлежит дальнейшему анализу, а нуль говорит о том, что данное свойство не имеет отношения к соответствующему требованию либо ведет к самоочевидным решениям, как, например, аспекты надежности, связанные с машинонезависимостью.

Есть в табл. 5.1 и такие требования, которые выливаются в разработку конкретных технических условий, иллюстрируемых табл. 5.2. В обеих таблицах соответствующие условия имеют одинаковые номера. Некоторые из приведенных технических условий очевидны, другие могли бы в случае отсутствия формализованного метода ускользнуть от внимания разработчика. Например, могло быть не предусмотрено условие У4, согласно которому испытания программных средств должны проводиться более чем на одной машине; выявление этого условия позволяет своевременно спланировать тести-

[1] Для примера можно назвать диаграмму последовательности действий, матрицу функциональных характеристик и схему взаимодействия функций.

рование программ на разных машинах. После завершения разработки перечня технических условий он становится документом, содержащим совокупность конкретных легко модифицируемых технических требований, которые могут подвергаться дальнейшему анализу на согласованность и легко группируются в руководящие принципы проектирования различных компонентов системы.

Одним из главных необходимых условий формирования матрицы «Требования — свойства» является определение состава ее элементов, которые на практике оказываются важными для обеспечения желаемых характеристик программного изделия. Хорошей основой для выполнения этой работы может служить рассмотренная в предыдущих главах совокупность показателей качества программного обеспечения. Еще двумя полезными средствами являются вопросники, позволяющие гарантировать проверяемость соответствия программного изделия требованиям, и средства анализа ключевых слов, предназначенные для контроля непротиворечивости исходных требований и технических условий.

В качестве примера, демонстрирующего выгоды использования таких средств, ниже приводится образец выходных данных, которые могли бы иметь место в результате применения анализатора ключевых слов в каком-нибудь крупномасштабном проекте:

```
.
.
.
доступ
У147 Подсистема терминального доступа должна обеспечивать легкий и естественный доступ ко всем
     программным средствам планирования.
У266 Система должна иметь совершенную защиту от
     несанкционированного доступа.
.
.
.
```

Даже такое элементарное средство обработки текстовой информации дает в руки системного аналитика очень

ценный способ проверки согласованности технических условий. Разумеется, окончательное суждение относительно противоречивости или непротиворечивости условий У147 и У266 и им подобных должно принадлежать самому разработчику. Однако если не применять предварительного сканирования, обеспечиваемого анализатором ключевых слов, то конфликтующие требования могут оказаться своевременно невыявленными и всплывут гораздо позже, когда разрешение противоречий станет делом дорогостоящим и трудным; а вероятность такой ситуации велика вследствие наличия большого числа требований, устанавливаемых в крупномасштабных разработках различными группами специалистов.

Таким образом, метрика, разработанная в гл. 4, должна оказаться полезной не только для анализа уже созданных программ, но и для предварительного анализа условий, которым такая программа должна удовлетворять. Вся практика разработки программных средств показывает, что гораздо легче бывает с самого начала придать программе какое-либо свойство, чем пытаться добавить его впоследствии.

Применение метрики качества программного обеспечения представляется наиболее ценным, хотя и трудно оценить эту полезность количественно.

Ниже кратко излагаются еще несколько принципов этапа выработки требований, которые предполагают затрату существенных усилий по управлению разработками и анализу системы (особенно первый из них). Каждый из сформулированных принципов мог бы быть в свою очередь разбит на ряд более детальных вопросов. Второй из принципов, хотя и кажется банальным, все же заслуживает гораздо большего внимания, чем ему обычно оказывается. Третий вопрос, кажущийся не менее банальным, позволяет применять в какой-то мере количественную оценку, называемую «Индексом непонятности» [30]. Этот показатель вычисляется по следующему алгоритму:

Шаг 1

А. Взять выборочно фрагмент текста документации (лучше всего размером в 100 слов, но не более 200).

Таблица 5.1

Матрица «Требования — свойства»

Свойства системы / Функциональные требования	Быстродействие	Память	Ввод-вывод	Надежность	Удобство эксплуатации	Возможность расширения	Измеряемость характеристик	Стандарты программирования	Тестирование
T1 Машинонезависимость	П	А	У1	0	У2, 3	0	0	У2, 3	У4
T2 Должно быть занято не более 25К слов оперативной памяти	0	У5	У5	0	0	У5, 6	У7	0	У7, 8
T3 Секционирование программ	А	У9	У9	У10, 11	У12	У12	0	0	У13, 14
T4 Многократное использование секций в итеративном процессе вычислений	У15	У16, Т7	У16, 17	У18	У19	У19	У15	У20	У21, 22

Условные обозначения:

0 — свойство не связано с требованием или самоочевидно,
А — свойство проанализировано,
П — свойство подлежит дальнейшему изучению,
Уi — свойство определяется техническим условием Уi,
Тj — свойство учтено требованием Тj.

Б. Подсчитать количество предложений в выбранном фрагменте.

В. Подсчитать среднее количество слов в одном предложении, разделив общее число слов на общее количество предложений.

Шаг 2

А. Подсчитать количество длинных слов, состоящих из четырех [1]) или более слогов, исключая состав-

[1] В переводе приведены цифры и правила словообразования, характерные для русского языка и отличные от английских,— *Прим. перев.*

Таблица 5.2

Перечень технических условий на проектирование

У1	Отмечать каждый машинозависимый оператор ввода-вывода специальной комментирующей картой.
У2	Использовать стандартный Фортран.
У3	В тех случаях, когда по соображениям эффективности программа пишется на ассемблере, иметь эквивалентную программу на Фортране.
У4	Провести тестирование не менее чем на двух машинах.
У5	Организовать внутреннее резидентное ядро в оперативной памяти и библиотеку на внешнем запоминающем устройстве.
У6	Обеспечить возможность обновления и расширения библиотеки.
У7	Оставить в оперативной памяти место для стандартной программы измерения характеристик системы и для тестовых драйверов.
У8	Провести тестирование библиотечных программ и проверку возможности обновления библиотеки.
У9	Резидентная область памяти для хранения параметров программной секции должна быть ограниченной или переменной величины.
У10	Предусмотреть средства идентификации альтернативных условий окончания работы программных секций, если основные условия не достигаются.
У11	Предусмотреть механизм обнаружения ситуации, когда больше нет программных секций, подлежащих исполнению.
У12	Предусмотреть расширяемый перечень условий окончания работы программной секции.
У13	Разработать контрольные примеры для проверки совместной работы разных программных секций.
У14	Разработать контрольные примеры для условий У10, У11 и У12.
У15	Построить процесс вычислений так, чтобы пользователь мог сам принимать решения относительно продолжения или прекращения итераций.
У16	Иметь ограниченную резидентную область памяти для хранения параметров программной секции. Уточнение условия У9.
У17	Предусмотреть средства определения зависимых и независимых переменных, условий окончания вычислительного процесса, допусков и начальных значений частных производных для организации итеративного вычислительного процесса.
У18	Предусмотреть резервные средства на случай, если итеративный процесс не сходится.
У19	Предусмотреть расширяемый список переменных, участвующих в итеративном процессе.

Продолжение

У20	Состояние резидентной области памяти для параметров программной секции не должно изменяться в ходе выполнения программы.
У21	Разработать контрольные примеры для многократного использования программной секции в итеративном процессе.
У22	Разработать контрольные примеры для условий У15, У18, У19.

ные слова из коротких компонентов (например, машино-час, противодействие и т. п.), имена собственные и причастия, образованные от глаголов с помощью суффиксов (например, оформленный, разрешенный, выбираемый и т. п.), когда сам глагол содержит менее четырех слогов.

Б. Определить процент длинных слов, разделив количество длинных слов на общее число слов в анализируемом фрагменте и умножив результат на 100.

Шаг 3

Найти численное значение «Индекса непонятности» по формуле:

$$\text{Индекс непонятности} = (\text{Среднее число слов в предложении} + \text{Процент длинных слов}) \cdot 0{,}4.$$

Для обычных деловых документов приемлемым должно считаться значение этого индекса в пределах от 10 до 12, для научных отчетов, технической документации и спецификаций — от 12 до 16.

Итак, принципы проектирования на стадии выработки требований состоят в следующем:

1. Анализ и обсуждение требований к системе должны быть организованы таким образом, чтобы каждый участник разработки программного обеспечения (заказчик, руководитель проекта, проектировщик, программист) мог быть гарантирован от неправильного толкования требований и мог письменно зафиксировать свое согласие с обсужденными и утвержденными спецификациями.

2. Для каждого пункта требований должны быть сформулированы количественные критерии оценки степени их выполнения.

3. Требования должны описываться ясным, доходчивым языком без излишнего многословия.

5.1.2. Руководящие принципы этапа проектирования системы

В ходе этого этапа разработки исходные требования трансформируются в детальные описания способов их реализации. Это значит, что руководящие принципы предыдущего этапа должны быть переформулированы в терминах анализа проектной деятельности, с тем чтобы гарантировать в любой момент соответствие проектных решений выработанным требованиям. Вот основные принципы этапа проектирования:

1. Проектные решения должны обсуждаться со всеми участниками разработки, которые должны подтвердить, что проектирование ведется в соответствии с техническими требованиями и что критерии приемлемости проектных решений удовлетворительны.

2. Необходимо обеспечивать возможность проверки соответствия системы требованиям после внесения в нее изменений и следить за тем, чтобы технические требования имелись у всех участников разработки.

3. Программы должны разбиваться на модули по функциональному признаку.

4. Необходимо стремиться к использованию уже существующих, опробованных на практике модулей.

5. Работа программных средств должна моделироваться с целью определения затрат машинного времени и способа организации распределения памяти.

6. Модули, которым присущи критические требования, должны оцениваться путем реализации их в составе некоторой системы-прототипа.

7. Предварительный план испытаний должен строиться в соответствии с пунктами технических требований, должен содержать критерии приемлемости результатов испытаний, должен указывать необходимую глубину тестирования каждого модуля, поскольку в не-

которых случаях может оказаться достаточным лишь повторное тестирование тех блоков, в которые вносились изменения, но в других необходимо исчерпывающее тестирование каждого элемента модуля в связи с ожидаемой сложностью их взаимодействия; кроме того, предварительный план испытаний должен описывать действия на каждом этапе их проведения и содержать необходимую информацию для руководства, касающуюся ресурсов, планов, финансирования, способов учета издержек и т. п.

Первые четыре принципа направлены на обеспечение ПОНЯТНОСТИ, ЗАВЕРШЕННОСТИ, СТРУКТУРИРОВАННОСТИ и УДОБСТВА ЭКСПЛУАТАЦИИ системы программного обеспечения. Следование пятому принципу зависит от возможности и необходимости выполнения моделирования в условиях ограничений по срокам и денежным ресурсам. Шестой принцип является в какой-то степени дискуссионным, однако те или иные предварительные варианты программ в сущности необходимо создавать во имя обеспечения реализуемости системы. Можно рекомендовать создание прототипа вне рамок основного проекта, но столь же целенаправленное, планомерное, документированное и обеспеченное финансированием, а не распыленное по другим задачам, операциям проектирования и направлениям разработок. После выполнения своей роли прототип должен быть ликвидирован.

Седьмой принцип касается разработки плана испытаний системы, в ходе которых должна будет проверяться способность модулей к совместному функционированию в составе разрабатываемого программного обеспечения. Этот принцип предполагает наличие специальной отдельной независимой группы испытателей, в задачи которой входит проверка системы на соответствие требованиям, и наличие формализованных методов контроля.

5.1.3. Руководящие принципы этапа программирования и отладки

Как отмечалось в гл. 4, этот этап является в книге предметом наиболее подробного обсуждения. Рассмотренные в предшествующих главах характеристики программного изделия не представляется возможным всегда выбирать однозначно, поскольку выбор определяется конкретными интересами пользователя, системными требованиями, имеющимися вычислительными ресурсами или техническими средствами, которые должны быть установлены в будущем, и рядом других ограничений. Так, например, потребность в быстрой загрузке программ или жесткие ограничения на время выдачи результатов в сочетании с ограничением на стоимость технических средств может привести к исключению из рассмотрения таких показателей качества программного обеспечения, которые хотя и важны с точки зрения обеспечения других желаемых характеристик, но вступают в противоречие с требованием ЭФФЕКТИВНОСТИ (например, в смысле частоты появления ошибок, их обнаружения и устранения). Как правило, однако, процесс создания программного изделия не бывает подвержен сразу всем указанным ограничениям, благодаря чему показатели не становятся конфликтующими. Таким образом, разработчик программных средств стоит лишь перед вопросом экономической целесообразности и возможности применения метрики качества исходя из установленных сроков. Поскольку некоторые из показателей невозможно использовать при отсутствии автоматических средств их анализа, мы отложим рассмотрение принципов этапа программирования и отладки до разд. 5.1.6.

5.1.4. Руководящие принципы этапа усовершенствования и системных испытаний

Рабочие испытания программных средств в процессе проектирования осуществляются в целях проверки правильности функционирования модулей по отдельности

и совместно и гарантирования верной реализации межмодульных сопряжений. Проверки первого типа часто называют индивидуальными испытаниями, а второго — стыковочными.

Цель системных испытаний состоит в том, чтобы гарантировать соответствие системы требованиям, и подход к проведению этих испытаний может быть совершенно иным, чем к рабочим проверкам. Основное внимание в ходе системных испытаний уделяется тому, чтобы в условиях реальных данных и рабочих процедур, предусмотренных техническими условиями, найти ситуации, в которых система перестает удовлетворять требованиям. Считается, что лучше всего эта цель может быть достигнута независимой группой испытателей.

5.1.4.1. Принципы индивидуальных испытаний

1. Тесты должны предусматривать проверку той точности вычислений, которая установлена требованиями и отражена в критерии приемлемости результатов тестирования.

2. Все логические ветви должны пройти успешное тестирование не менее одного раза. Тестированием должно быть охвачено все множество характеристических путей в алгоритме, т. е. минимальная совокупность путей, содержащая все возможные разветвления.

3. Функции должны реализовываться на основе модулей, прошедших предварительную проверку при экстремальных значениях входных переменных. Результаты должны соответствовать утвержденным требованиям. Должна предусматриваться проверка всех ситуаций, ведущих к возникновению неопределенностей, и гарантироваться работоспособность метода выявления ошибок.

4. Необходимо постоянное ведение базы данных о ходе испытаний, включающей контрольные примеры, критерии приемлемости результатов тестирования, статистику использования ЭВМ, кумулятивные оценки глубины тестирования и другую информацию, необходимую для обеспечения эффективных и глубоких проверок.

5.1.4.2. Принципы стыковочных испытаний

1. До начала стыковочных испытаний необходимо иметь все специальные подпрограммы-утилиты.
2. Необходимо обеспечить наличие всех внешних объектов, к которым предусматриваются обращения.
3. Совместно используемые области памяти должны иметь надлежащую защиту.
4. Необходимо проверить, что подпрограммы защищают и восстанавливают общие участки памяти и регистры так, как это предусмотрено проектом.

5.1.4.3. Документация процесса испытаний

На основе базы тестовых данных должны составляться в удобочитаемой форме отчеты, содержащие следующую информацию:

— идентификатор контрольного примера,
— идентификатор программного модуля,
— имена переменных,
— критерии приемлемости результатов испытаний с указанием допустимых отклонений,
— достигнутые результаты тестирования,
— оценку степени завершенности проверки модуля,
— резюме, касающееся узких мест программного обеспечения.

5.1.5. Руководящие принципы этапа ввода в действие и эксплуатации

На этом этапе разработчик программных средств или группа, проводящая испытания на договорной основе, получает информацию, характеризующую затруднения, которые встретились при испытаниях и использовании системы. Здесь могут потребоваться изменения во входных данных и выдаваемых результатах. Это особенно легко сделать, если на всех предыдущих этапах применялись изложенные выше принципы разработки программного изделия.

5.2. ВОЗМОЖНОСТИ ИСПОЛЬЗОВАНИЯ СРЕДСТВ АВТОМАТИЗАЦИИ В ПРОЦЕССЕ СОЗДАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В табл. 5.3 приведены описания целого ряда автоматических программных средств, которыми в настоящее время располагает фирма TRW или с помощью которых осуществляет процесс разработки. Все указанные средства доступны в системах, создаваемых фирмой. В таблице осуществлена привязка соответствующих средств к показателям качества программного обеспечения, рассмотренным в гл. 4 и к конкретным этапам его разработки, на которых данные средства применимы (указывается номер соответствующего подраздела). Некоторые из средств, приведенных в табл. 5.3, реализованы в нескольких вычислительных системах, таких, как UNIVAC 1108, IBM 370, CDC 6000 и CDC 6600, используемых для обработки информации космических исследований, а также в Системе CDC 3800, функционирующей в г. Саннивейле (шт. Калифорния).

Указанный в таблице уровень реализации автоматических средств характеризует степень готовности и доступности их для использования. Различия уровней реализации заключаются в следующем:

Уровень I

А. Написанная программа реализована и испытана с применением того или иного языка и операционной системы из числа используемых в фирме TRW.

Б. Имеется «Руководство для программистов» и «Инструкция по использованию программы».

В. Применяются современные методы контроля программы и ведения документации.

Уровень II

А. Написанная программа реализована и испытана с применением того или иного языка и операционной системы из числа используемых в фирме TRW.

Б. Использование программы требует участия ее разработчика или другого лица, способного выполнять те же функции.

В. Применяются неформализованные методы контроля структуры программы и документации.

Таблица 5.3

Автоматические программные средства и их связь с руководящими принципами разработки и метрикой программного изделия

Имя программы	Уровень реализации	Принципы и показатели, в отношении которых применима данная программа	Описание программы	Примечания
DOCEDT	I	Текущее ведение и обновление технических требований и любых других спецификаций (см. подразд. 5.1.1—5.1.5)	Позволяет формировать и затем вести документацию на магнитной ленте. Первоначально документ наносится на перфокарты, после чего переносится на ленту с исправлениями и дополнениями на основе использования карт DOCEDT, относящихся к изменениям	Эффективно применена в проекте NASA/JSC, касающемся управления траекторией стрельбы. В проекте использован язык Фортран и вычислительная система UNIVAC 1108. Программа может быть легко настроена на любую другую версию Фортрана
COMPSIM	II	Этап проектирования (подразд. 5.1.2)	Основана на использовании универсальной системы моделирования SALSIM и позволяет имитировать различные вычислительные комплексы, в том числе аппаратуру пользователя, прикладные программы, операционную систему и условия функционирования	Нашла конкретное применение в ряде проектов NASA, использующих Системы UNIVAC 1108, IBM 360/ и 370, а также XDS-SIGMA 5. Подобна программе SPRINT, которая может использоваться в Системе CDC 6500 в режиме разделения времени

Продолжение

Имя программы	Уровень реализации	Принципы и показатели, в отношении которых применима данная программа	Описание программы	Примечания
PPE	II	Программирование и отладка, рабочая проверка (подразд. 5.1.3 и 5.1.4), показатель РЦ-1 (см. табл. 4.4)	Выявляет участки программы или ее характеристики, приводящие к нерациональному использованию машинного времени	Используется в Системах IBM 360/370 с операционной системой OS/MV 7
TAFIRM	III	Программирование и отладка (подразд. 5.1.3), показатели ЗВ-3, ЗВ-6 из табл. 4.2 и ТЧ-1 из табл. 1.1	Проверяет выполнимость и точность вычислений в небольших логических блоках программ. Требует от разработчика разделения программы на тестируемые логические блоки	Применена в Системе CDC 800 с языком JOVIAL J4 и операционной системой IIB. Подобна программе ASIST-1, используемой в Системе XDS-SIGMA 5
FLOWGEN	I	Программирование и отладка, рабочая проверка (подразд. 5.1.3, 5.1.4)	Формирует блок-схемы программ по исходному тексту, написанному на языке Фортран	Применена в Системе CDC 6500, допускающей режим разделения времени, и Системе UNIVAC 1108. Предшествующий вариант программы был модифицирован в целях работы с более крупными программами и выдачи результатов постранично. Модифицированный вариант предоставляется за арендную плату фирмой CALCOMP

Продолжение

Имя программы	Уровень реализации	Принципы и показатели, в отношении которых применима данная программа	Описание программы	Примечания
DEPCHT	I	Программирование и отладка, рабочие и стыковочные испытания, системные испытания, ввод в действие и эксплуатация (подразд. 5.1.3—5.1.5), показатель ЗВ-4 (табл. 4.2)	Формирует в виде листинга список перекрестных ссылок на подпрограммы, к которым происходит обращение из анализируемой программы и наоборот. Формирует в виде блок-схемы описание связей анализируемой программы, начиная с задаваемой пользователем «исходной точки»	Применена в проекте NASA/JSC, касающемся управления траекторией стрельбы. Язык — Фортран V, ЭВМ — UNIVAC 1108
BLKGEN/ SPECPN	I	Программирование и отладка, рабочие и стыковочные испытания, системные испытания, ввод в действие и эксплуатация (подразд. 5.1.3—5.1.5), показатель ИН-5 (табл. 4.8)	Выявляет и обслуживает общие блоки данных COMMON в больших Фортран-программах	Использована в проекте NASA/JSC, касающемся управления траекторией стрельбы, в котором применены Фортран V и UNIVAC 1108. Реализована в качестве стандартного средства в языке JOVIAL J4, где управляется оператором COMPOOL

Продолжение

Имя программы	Уровень реализации	Принципы и показатели, в отношении которых применима данная программа	Описание программы	Примечания
DPNDCY	I	Программирование и отладка, рабочие и стыковочные испытания, системные испытания, ввод в действие и эксплуатация (подразд. 5.1.3—5.1.5), показатель ЗВ-11	Выполняет анализ совместимости входов в базу данных через оператор COMMON. Результирующая распечатка показывает перекрестные ссылки переменных с конкретными именами на подпрограммы и наоборот. Указываются ошибки и несоответствия	Использована в системе, где применена ЭВМ UNIVAC 1108 и язык Фортран, совместно с программой BLKGEN/SPECPN
FREF	I	Программирование и отладка, рабочие и стыковочные испытания, системные испытания, ввод в действие и эксплуатация (подразд. 5.1.3—5.1.5), показатель СГ-10 (табл. 4.3)	Генерирует список перекрестных ссылок для номеров операторов, имен переменных, имен массивов и имен функций. Есть блок, определяющий частоту обращений к каждому имени	Доступна в Системе CDC 6500 в режиме разделения времени для программ, написанных на языке Фортран IV. Подобна программе BREF для языка BASIC

Продолжение

Имя программы	Уровень реализации	Принципы и показатели, в отношении которых применима данная программа	Описание программы	Примечания
FSEQ	I	Программирование и отладка, рабочие и стыковочные испытания, системные испытания, ввод в действие и эксплуатация (подразд. 5.1.3—5.1.5), показатель ОТ-3 (табл. 4.10)	Переупорядочивает метки операторов (номера) по возрастанию значений в Фортран-программах; помогает отмечать операторы FORMAT посредством занесения в последний разряд номера оператора двойки	Доступна в Системе CDC 6500 в режиме разделения времени
CODE AUDITOR (Программный ревизор)	II	Программирование и отладка, рабочие и стыковочные испытания, системные испытания, ввод в действие и эксплуатация (подразд. 5.1.3—5.1.5), показатели ИН-1, СТ-3, ОТ-5, СТ-4, ОТ-10, СТ-5, СТ-2, РЦ-2	Предоставляет стандартные программные средства для оценивания программ, написанных на Фортране	Эффективно применена для Системы CDC 7600 в проекте местной системы обороны. В настоящее время модифицируется для расширения возможностей оценки стандартных средств программирования

Имя программы	Уровень реализации	Принципы и показатели, в отношении которых применима данная программа	Описание программы	Примечания
TDEM	I	Рабочие испытания (подразд. 5.1.4), показатели ИН-5, ИН-6 (табл. 4.8), КМ-9 (табл. 4.6)	Измеряет эффективность контрольных примеров путем выдачи сведений о проверенных ветвях исходной Фортран-программы, о логически связанных сегментах и о фактической частоте проверок всех сегментов и ветвей	Эффективно использована в проекте NASA/JSC, касающемся управления траекторией стрельбы, в котором применены язык Фортран V и UNIVAC 1108. Родственные программы действуют в других системах: PAGE 1, CDC 6600, CDC 7600, NODAL CDC 6000 (в режиме разделения времени), TATTLE, CDC 3800 с использованием языка JOVIAL J4
ATDG	III	Рабочие испытания (подразд. 5.1.4)	Выявляет логические пути в программах и генерирует оптимальные тесты для	Имеется прототип программы, который проходит стадию испытаний и оценки.

Продолжение

Имя программы	Уровень реализации	Принципы и показатели, в отношении которых применима данная программа	Описание программы	Примечания
			модулей и подпрограмм, написанных на языке Фортран	Используется в системе с UNIVAC 1108 и языком Фортран V
TCOCO	III	Стыковочные испытания, системные испытания, ввод в действие и эксплуатация (подразд. 5.1.4, 5.1.5)	Комплекс программных средств для контроля конфигурации больших систем программного обеспечения	Применена в Системе CDC 3800 с языком JOVIAL J4, которая представляет собой реализацию обобщенной управленческой информационной системы (GIM), используемой непосредственно для конструирования средств контроля конфигурации других вычислительных систем

Уровень III

А. Имеется исходный вариант программы, служащий прототипом.

Б. Документация несовершенна.

5.3. ПРИМЕНЕНИЕ МЕТРИКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ ОБНАРУЖЕНИЯ И УСТРАНЕНИЯ ОШИБОК

Представленная в гл. 4 система показателей была всесторонне проанализирована с использованием обширной базы данных об ошибках в программных средствах и опыте их обнаружения и устранения в системе, создававшейся фирмой TRW для ВВС США по проекту CCIP-85. Об этой базе данных частично уже говорилось в гл. 1, где указывалось, что она содержит сведения о 224 типах программных ошибок, сгруппированных в 13 основных категорий. Для примера в гл. 1 приводился фрагмент этой базы данных. В табл. 5.4 она отображена полностью. Как и прежде, буква В в той или иной графе отмечает стадию, на которой ошибки данного типа обычно возникают, а буква О — стадию, на которой они обычно обнаруживаются и исправляются. Шифр одного из показателей в графе указывает стадию, на которой ошибки конкретного типа могли бы наверняка обнаруживаться и устраняться в случае использования данного показателя. Таким образом, табл. 5.4 характеризует эффективность показателей по отношению к ошибкам различных типов. Два последних столбца этой таблицы содержат номера этапов, на которых ошибка обнаруживается и устраняется соответственно с применением и без применения метрики. В табл. 5.5 оценки эффективности сгруппированы по показателям и для каждого показателя перечислены шифры типов ошибок, в отношении которых он оказался ключевым с точки зрения обнаружения и устранения, а также характерная степень сдвига стадии обнаружения в сторону более ранних этапов. В табл. 5.6 та же информация приведена в ином разрезе: она отображает оценки относительной полезности каждого показателя для обнаружения и устранения ошибок. Данные этой таблицы позволяют сделать

общий вывод, что более раннее применение автоматических и полуавтоматических средств контроля показателей завершенности и согласованности дает наибольший выигрыш в смысле повышения эффективности процесса обнаружения и устранения ошибок. Среди 94 типов ошибок (из 224), обнаруженных и устраненных благодаря использованию метрики, успех по 37 типам определялся показателями, подобными завершенности, а 34 — показателями вида согласованности. Из суммарного зафиксированного выигрыша в 200 этапов (каждый из которых длится в проекте средних масштабов около четырех месяцев) 62 были получены благодаря метрике, подобной завершенности, и 89 — благодаря показателям наподобие согласованности. Это важный вывод, но ему не следует слишком уж удивляться, поскольку завершенность и согласованность представляют собой две из трех элементарных характеристик свойства надежности.

Другим полезным аспектом анализа этих показателей является сравнение их исправляющей способности с характеристиками потенциальной выгодности использования из табл. 4.1. Можно с удовлетворением отметить, что практически все показатели, оказавшиеся важными с точки зрения возможностей обнаружения и исправления ошибок, имеют и максимальную оценку выгодности, равную 5 баллам, причем ни один из показателей, способствующих исправлению ошибок, не имеет значения этой оценки ниже 3 баллов.

Разумеется, приведенные данные представляют собой лишь частичные оценки, так как исправление ошибок и надежность являются только двумя из множества аспектов обеспечения высокого качества программных средств. Однако, если иметь ввиду, что тестирование и испытания поглощают львиную долю общих усилий по разработке программного обеспечения, приведенные в рассмотренных таблицах оценки представляются чрезвычайно полезными для принятия решения о том, каким показателям следует отдать предпочтение в смысле их дальнейшей отработки и использования.

Дополнительные показатели, присутствующие в табл. 5.4—5.6, представляют собой модификации рассмотренных ранее и имеют следующий смысл:

ИН-1++ *характеризует согласованность начального блока модуля и является разновидностью показателя СГ.*

Не противоречат ли списки входных и выходных переменных, операторов и форматов в программе их описаниям, данным в начальном блоке?

СГ-2+ *характеризует согласованность типов данных.*

Является ли указание типа переменной (действительная, целая, только для чтения, упакованная и т. п.) всюду одинаковым?

СГ-12+ *характеризует информативность базы данных и представляет собой показатель типа ИН.*

Включает ли программа собственное описание базы данных, в том числе спецификации структур данных и их элементарных блоков, характер доступа, номинальные значения, типы данных, приемлемые границы области изменения и т. п.?

СГ-12++ *характеризует согласованность описаний базы данных в программе и в спецификации.*

Совместимы ли структуры данных, предусматриваемые программой, с их описанием в спецификации базы данных?

ЗВ-1+ *характеризует способность программы устанавливать память в исходное состояние и правильно заканчивать работу с памятью.*

Имеются ли в программе средства, позволяющие надлежащим образом приводить в исходное положение, повторно устанавливать в исходное состояние и прекращать использование запоминающих устройств?

ЗВ-2+ *характеризует способность программы воспринимать сигналы от устройств, работающих в режиме онлайн.*

Есть ли в программе средства, обеспечивающие возможность распознавания факта работы с аппаратурой, функционирующей в режиме онлайн, и способна ли она предпринимать правильные действия при отсутствии информации от таких устройств?

Таблица 5.4

Оценка возможностей метрики с точки зрения выявления ошибок в программном обеспечении на различных этапах его создания

Этапы создания программного обеспечения / Категории и типы ошибок	Разработка требований	Проектирование	Кодирование и отладка	Рабочие испытания	Утверждение	Приемочные испытания	Стыковка с другими программами	Сдача пользователю	k_i	t_i
А. ОШИБКИ ПОДГОТОВКИ И ВВОДА ДАННЫХ С ПЕРФОКАРТ										
1. Программа получает параметр в формате, отличающемся от того, который предусмотрен спецификацией требований		В ИИ-1+				О			2	6
2. Программа не получает или не воспринимает требуемый параметр		В ИИ-1+					О		2	7
3. Программа получает параметры в порядке, отличающемся от того, который предусмотрен спецификацией		В ИИ-1+					О		2	7
4. Программа не способна работать с данными во всем предусмотренном диапазоне их изменения		В ИИ-1+				О			2	6
5. Программа рассчитана на работу с элементами параметра, отличающегося от того, который предусмотрен спецификацией		В ИИ-1+				О			2	6
6. Номинальные значение параметра или значение, присваиваемое по умолчанию, отличаются от предусмотренного спецификацией		В					О		НМ	7

7. Программа воспринимает данные как правильные за пределами допустимого диапазона изменения		В						О	НМ	8
8. Программа не воспринимает данные, находящиеся в границах допустимых изменений		В						О	НМ	8
9. Программа вызывает переполнение таблиц в оперативной памяти, занося в них данные, лежащие в допустимых пределах		В		ЗВ-9				О	4	8
10. Программа переполняет отведенную ей область внешней памяти данными, лежащими в допустимых границах		В		ЗВ-9				О	4	8
11. Программа проходит первый тест успешно, но при последующих тестах не работает		В				О			НМ	6
12. Программа предусматривает местоположение параметра, отличающегося от того, которое предусмотрено спецификацией		В ИН-1+				О			2	6
13. Программа не отвергает неправильный параметр		В		ЗВ-7		О			4	6
14. Программа отвергает правильный параметр		В				О			НМ	6
15. Программа некорректно преобразует правильный параметр		В				О			НМ	6
16. Программа читает данные за пределами нужного поля (например, порядковый номер считывает в качестве исходных данных для проведения контроля, не относящегося к анализу упорядоченности перфокарт)		В		ЗВ-7	О				4	5
17. Программа время от времени отвергает данные с незаполненным полем в зависимости от его местоположения		В					О		НМ	7
18. Программа не может распознать разделителей полей данных			В	ЗВ-7	О				4	5

Продолжение

Категории и типы ошибок \ Этапы создания программного обеспечения	Выработка требований	Проектирование	Кодирование и отладка	Рабочие испытания	Утверждение	Приемочные испытания	Стыковка с другими программами	Сдача пользователю	k_j	i_j
Б. ОШИБКИ ОБРАБОТКИ ДАННЫХ ПРИ ВВОДЕ С МАГНИТНОЙ ЛЕНТЫ										
1. Программа предусматривает слишком малые интервалы времени между двумя последовательными считываниями		В					О		НМ	7
2. Программа читает данные за пределами логической границы ленты		В		КМ-2			О		4	7
3. Программа читает данные за пределами конца файла		В		О КМ-2					4	4
4. Программа считывает данные некорректным способом		В ИН-1+					О		2	7
5. Программа пытается считать блок, который не должен читаться в данный момент		В		О ЗВ-2+					4	4
6. Программа не распознает во время считывания ленты ошибку при контроле по четности		В		ЗВ-2+				О	4	8
7. Программа игнорирует при записи физический конец ленты		В		КМ-2+				О	4	8
8. Программа записывает данные неверным способом		В ИН-1+					О		2	7

9. Программа записывает данные в неверном формате	В ИН-1+					О		2	7
10. Программа предусматривает слишком малые интервалы времени между последовательными операциями записи на ленту	В					О		НМ	7
11. Программа записывает данные при наличии ошибки по четности	В						О	НМ	8
12. Программа портит нужные данные	В				О			НМ	6
13. Программа не записывает признак конца файла	В		ЗВ-1+	О				4	5
14. Программа не записывает признак логического конца ленты	В		ЗВ-1+	О				4	5
15. Программа не перезаписывает внутреннюю метку ленты	В			О				НМ	5
16. Программа пытается записывать информацию на устройство, не готовое к работе	В		О ЗВ-2+					4	4
17. Программа пытается считывать информацию не с того накопителя на магнитной ленте	В ИН-1+		О					2	4
18. Программа не пропускает без чтения или записи нужного количества файлов	В		О					НМ	4
19. Программа не обнаруживает неверную внутреннюю метку ленты	В						О	НМ	8
20. Программа не перематывает ленту перед считыванием данных	В		ЗВ-2+				О	4	8
21. Программа считывает с ленты больше полей данных, чем имеется в записи	В		О КМ-2					4	4
22. Программа не считывает с ленты полную запись	В		КМ-2		О			4	6
23. Программа не считывает информацию с ленты, содержашей продолжение файла	В		КМ-2				О	4	8

Продолжение

Этапы создания программного обеспечения / Категории и типы ошибок	Выработка требований	Проектирование	Кодирование и отладка	Рабочие испытания	Утверждение	Приемочные испытания	Стыковка с другими программами	Сдача пользователю	k_j	i_j
24. Программа не производит разгрузку ленты после завершения подготовительных операций		В		ЗВ-1+				О	4	8
В. ОШИБКИ ПРИ РАБОТЕ С ДИСКАМИ										
1. Программа предусматривает недостаточный интервал времени между двумя последовательными операциями записи		В					О		НМ	7
2. Программа предусматривает недостаточный интервал времени между двумя последовательными обращениями к данным		В					О		НМ	7
3. Программа получает доступ не к тому файлу, который нужен			В	ИН-1+	О				4	5
4. Программа получает доступ не к той записи, которая нужна			В		О				НМ	5
5. Программа делает попытки читать более длинную запись, чем существующая		В		КМ-2	О				4	5
6. Программа совершает попытки доступа к неопределенному файлу		В ИН-1+		О					2	4
7. Программа получает доступ не к тому устройству, которое нужно		В ИН-1+		О					2	4

8. Программа не запрашивает устройство и не открывает файл перед получением доступа	В		О ЗВ-2					4	4
9. Ошибка в данных проникает с диска в оперативную память	В						О	НМ	8
10. Ошибка в данных проникает из оперативной памяти на диск	В						О	НМ	8
11. Программа не закрывает файл после завершения операций над ним	В		ЗВ-1+				О	4	8
12. Программа пытается получить доступ к файлу, который защищен от несанкционированного доступа	В		ИН-1+		О			4	6
13. Программа пытается получить доступ к неопределенному устройству	В ИН-1+		О					2	4
Г. ОШИБКИ ВЫВОДА ДАННЫХ И СООБЩЕНИЙ									
1. Программа выдает в качестве результатов отладочную информацию		Б			О			НМ	6
2. Программа неправильно подсчитывает число выдаваемых на печать строк	В					О		НМ	7
3. Программа неправильно управляет движением каретки печатающего устройства		В	О					НМ	4
4. Программа выделяет мало места в памяти под выводимые данные		В	О ИН-1+					4	4
5. Программа пытается печатать в строке больше данных, чем это возможно	В		О					НМ	4
6. Программа выводит информацию не на то внешнее устройство	В ИН-1+			О				2	5

Продолжение

Категории типы ошибок \ Этапы создания программного обеспечения	Выработка требований	Проектирование	Кодирование и отладка	Рабочие испытания	Утверждение	Приемочные испытания	Стыковка с другими программами	Сдача пользователю	k_j	t_j
7. Программа неправильно преобразует данные для выдачи		В			О				НМ	5
8. Программа правильно печатает шапку документа, но не печатает данные		В		ИН-1++		О			4	6
9. Программа правильно печатает данные, но не печатает шапку документа		В		ИН-1++		О			4	6
10. При выдаче страницы теряются результаты		В					О		НМ	7
11. Выданные результаты содержат лишние данные или «абракадабру»			В		О				НМ	5
12. Выдаются лишние страницы		В					О		НМ	7
13. Неправильно выполняется перевод строки		В					О		НМ	7
14. Вверху и внизу страницы не выпечатываются грифы секретности информации		В		ИН-1++		О			4	6
15. Выдаются неверные грифы секретности		В		ИН-1++		О			4	6
16. Грифы секретности появляются в середине страницы		В				О			НМ	6
17. Блоки выдаваемой информации или формат не подходят для выполнения нужных операций	В							О	НМ	8
18. Большое число раз выдается одна и та же строка			В	О ИН-1++					4	4

19. Информация одной строки распределена по нескольким строкам			В	О					НМ	4
20. Данные в столбцах не располагаются соответственно их заголовкам			В	ИН-1++	О				4	5
21. Программа выдает неправильный список исходных данных			В	О					НМ	4
Д. НЕПРАВИЛЬНЫЕ СООБЩЕНИЯ ОБ ОШИБКАХ										
1. Сообщение об ошибке не соответствует описанному в документации			В	О					НМ	4
2. Сообщение об ошибке неполно		В		КМ-4		О			4	6
3. Сообщение об ошибке неинформативно		В		КМ-4				О	4	8
4. Сообщение об ошибке выдается при ее отсутствии			В			О			НМ	6
5. Сообщение не выдается при наличии ошибки		В		КМ-4			О		4	7
6. Сообщение об ошибке не описано в документации		В				О			НМ	6
7. Сообщение об ошибке не выдается в режиме онлайн			В	КМ-4		О			4	6
8. Сообщение об ошибке выдается несвоевременно		В		КМ-4		О			4	6
9. При возникновении ошибки выдается неправильное сообщение		В		О КМ-4					4	4
10. Сообщение об ошибке неправильно ориентирует оператора в его действиях		В				О			НМ	6
11. Сообщение об ошибке указывает на возможность ее устранения, но соответствующие средства не работают		В				О			НМ	6

Продолжение

Категории и типы ошибок \ Этапы создания программного обеспечения	Выработка требований	Проектирование	Кодирование и отладка	Рабочие испытания	Утверждение	Приемочные испытания	Стыковка с другими программами	Сдача пользователю	k_f	t_f
Е. ОШИБКИ СОПРЯЖЕНИЯ ПРОГРАММ										
1. Список аргументов подпрограммы содержит количество переменных, не согласующееся с оператором вызова			В	О					4	4
2. Список аргументов подпрограммы имеет иную упорядоченность, чем в операторе вызова			В	ПП-1++ ПН-1++	О				4	5
3. Переменные в списке аргументов подпрограммы и в операторе вызова определены не одинаково			В	ПН-1++	О				4	5
4. Программа вызывает для конкретной функции не ту подпрограмму			В		О				НМ	5
5. Подпрограмма изменяет значение константы, используемой для ее вызова			В	СГ-2+		О			4	6
6. Две подпрограммы, выполняющие одну и ту же функцию, дают разные результаты		В						О	НМ	8
7. Результаты работы подпрограммы зависят от того, в какой очередности она выполняется		В					О		НМ	7

8. Подпрограмма не восстанавливает исходное состояние регистров, которое было ею изменено			В	ЗВ-1+	О				4	5
9. Вызывающая программа не использует вызванную подпрограмму		В						О	НМ	8
10. Подпрограмма выдает сообщение об ошибке		ИН-1+						О	2	8
11. Подпрограммой не предусматривается возможность появления неверных результатов		В		ЗВ-14	О				4	5
12. Вызывающая программа не допускает возможности ошибки в подпрограмме		В		ИН-1++			О		4	7
13. Условие возникновения ошибки в подпрограмме неправильно интерпретируется вызывающей программой		В					О		НМ	7
14. Подпрограмма сигнализирует о неверном определении условий появления ошибки		В			О				НМ	5
15. Подпрограмма не сообщает о возникновении ошибки		В		ЗВ-14 КМ-4+	О				4	5
Ж. ОШИБКИ ОБРАЩЕНИЯ К ВНЕШНИМ УСТРОЙСТВАМ										
1. Имеется попытка использовать внешние устройства, не приведенные в состояние готовности		В		ЗВ-1				О	4	8
2. Ошибка возникает во время работы внешнего устройства		В						О	НМ	8
3. Используемое внешнее устройство выдает неверные данные		В				О			НМ	6

Продолжение

Категории и типы ошибок \ Этапы создания программного обеспечения	Выработка требований	Проектирование	Кодирование и отладка	Рабочие испытания	Утверждение	Приемочные испытания	Стыковка с другими программами	Сдача пользователю	k_j	t_j
4. Логическое отображение реальных аппаратных средств не изменяется при их замене другими		В						О	НМ	8
5. Программа прекращает работу в случае неготовности периферийного оборудования к использованию		В		ЗВ-2+				О	4	8
6. При занесении информации в регистр возникает программная ошибка		В						О	НМ	8
7. Модель аппаратных средств имеет иные временные характеристики, чем реальные технические средства		В						О	НМ	8
8. Модель аппаратных средств правильна для номинальных значений характеристик, но при крайних значениях работает неверно		В						О	НМ	8
9. Логическая модель аппаратных средств не позволяет восстановить предшествующее состояние системы или смоделировать будущее состояние		В		О					НМ	5

3. ОШИБКИ ОБРАЩЕНИЯ К БАЗЕ ДАННЫХ										
1. При использовании данных переменной длины значения указателей не увеличиваются			В	О					НМ	4
2. При использовании данных переменной длины значения указателей не уменьшаются			В	О					НМ	4
3. Указатели не инициируются при обработке данных переменной длины		В		О ЗВ-1					4	4
4. При обработке данных переменной длины превышается емкость буферной области памяти		В				О			4	6
5. Размеры полей данных переменной длины не согласуются с ожидаемыми		В				О			НМ	6
6. Буфер не очищается перед использованием			В	О ЗВ-9					4	4
7. Не работает механизм удвоения размера буфера			В	О					НМ	4
8. В базе данных имеются неверные номинальные значения данных или значения, присваиваемые по умолчанию		В					О		НМ	7
9. Значения присваиваются по умолчанию даже тогда, когда во входных данных указываются конкретные величины		В			О				НМ	5
10. Программа присваивает по умолчанию неверное значение переменной		В						О	НМ	8
11. В базе данных имеются посторонние переменные		В		СГ-11				О	4	8
12. В базе данных имеется дублирование переменных		В		СГ-11				О	4	8

Продолжение

Категории и типы ошибок \ Этапы создания программного обеспечения	Выработка требований	Проектирование	Кодирование и отладка	Рабочие испытания	Утверждение	Приемочные испытания	Стыковка с другими программами	Сдача пользователю	k_j	i_j
13. В базе данных отсутствуют требуемые переменные		В			О				НМ	5
14. Условия координатной системы несовместимы	В						О		НМ	7
15. Условия работы с блоками данных противоречивы	В						О		НМ	7
16. Матрица описана неверно			В				О		НМ	7
17. Условия, отмечаемые путем занесения единиц в соответствующие разряды, противоречивы		В					О		НМ	7
18. Данные стираются, но указатели своевременно не изменяются		В					О		НМ	7
19. Указатели изменяются, но данные своевременно не стираются		В					О		НМ	7
20. Данные разрушаются под влиянием ошибок и сбойных ситуаций		В		ИН-1++				О	4	8
21. Между программами существуют противоречивые операторы COMMON			В	СГ-1	О				4	5
22. Между программами существуют противоречивые операторы EQUIVALENCE			В	СГ-1	О				4	5

23. Таблицы расположены так, что сведения об одних переменных разрушаются другими		В			О				НМ	5
24. Переменным придаются неверные значения			В				О		НМ	7
25. Переменные не инициализируются			В	ЗВ-1			О		4	7
26. Переменные инициализируются должным образом лишь при первом входе			В	ЗВ-1	О				4	5
27. Для приема данных отводится недостаточный размер буфера		В		О					НМ	4
28. Для приема данных отводится слишком большой размер буфера		В						О	НМ	8
29. В качестве констант используются в разных местах разные величины		В		СГ-5				О	4	8
30. При обновлении базы данных происходит разрушение информации		В					О		НМ	7
31. Базу данных не удается обновить, когда это требуется		В						О	НМ	8
32. Работа ведется с неверной базой данных	В							О	НМ	8
И. ОШИБКИ ВЗАИМОДЕЙСТВИЯ С ПОЛЬЗОВАТЕЛЕМ										
1. Сообщение, требующее реакции оператора, непонятно		В		КМ-4++		О			4	6
2. Программа не отвергает вводимые вручную неверные данные		В		ЗВ-7		О			4	6
3. Программа отвергает вводимые вручную верные данные		В		О					НМ	4
4. Программа не укладывается в ограничения по времени прогона		В					О		НМ	7

Продолжение

Категории и типы ошибок \ Этапы создания программного обеспечения	Выработка требований	Проектирование	Кодирование и отладка	Рабочие испытания	Утверждение	Приемочные испытания	Стыковка с другими программами	Сдача пользователю	k_j	i_j
5. Программа чувствительна к последовательности ввода данных		В		КМ-2			О		4	7
6. Программа чувствительна к очередности ее выполнения		В		ИН-7				О	4	8
7. В ответ на реакцию оператора программа работает неправильно				В		О			НМ	6
8. Операторские интерфейсы документированы неполностью		В					О		НМ	7
9. Требуемая реакция оператора противоречит документации			В				О		НМ	7
10. Программа предлагает оператору неверные действия		В		КМ-4++		О			4	6
11. Программа не воспринимает правильных действий оператора		В				О			НМ	6
12. Программа требует сложных действий оператора		В				ДС-5		О	6	8
К. ОШИБКИ ВЫЧИСЛИТЕЛЬНОГО ХАРАКТЕРА										
1. Проверка на равенство дает неверный результат вследствие ошибок округления значений переменных			В			О			НМ	6

2. В результате преобразований элементов данных теряется точность		В				О			НМ	6
3. Происходит потеря точности в результате преобразования чисел в форму записи с плавающей точкой и наоборот		В				О			НМ	6
4. Теряется точность при переходе от двойной точности к обычной		В		ТЧ-1		О			4	6
5. Происходит потеря точности при переходе от обычной точности к записи в виде целого числа		В				О			НМ	6
6. Происходит потеря точности в результате перехода от записи с фиксированной точкой к той же форме записи, но с меньшим количеством дробных разрядов		В				О			НМ	6
7. Получается неверный результат из-за игнорирования скобок			В	О					НМ	4
8. Получается неверный результат из-за неправильного местоположения скобок			В	О					НМ	4
9. Получается неверный результат из-за использования неподходящего арифметического оператора			В	О					НМ	4
10. Получается неверный окончательный результат из-за отбрасывания промежуточных результатов			В	О					НМ	4
11. В результате вычислений с целыми числами происходит переполнение			В	О					НМ	4
12. В результате операций над числами с плавающей точкой происходит переполнение			В	О					НМ	4
13. В результате операций над числами с плавающей точкой происходит обнуление			В	О					НМ	4

Продолжение

Категории и типы ошибок \ Этапы создания программного обеспечения	Разработка требований	Проектирование	Кодирование и отладка	Рабочие испытания	Утверждение	Приемочные испытания	Стыковка с другими программами	Сдача пользователю	k_i	t_i
14. Вычислительными процедурами допускается деление на нуль			В	ЗВ-3	О				4	5
15. Вычислительными процедурами допускается извлечение квадратного корня из отрицательного числа			В	ЗВ-3	О				4	5
16. Арифметическая ошибка связана со смешанной формой представления чисел			В	О					НМ	4
17. Числа неправильно преобразуются из формы с фиксированной точкой в форму с плавающей точкой			В	О					НМ	4
18. Допущена ошибка в написании имени переменной			В	О					НМ	4
19. В операциях над матрицей теряется точность вследствие малых различий между большими числами		В		О					НМ	4
20. Ошибка возникает из-за неправильной индексации матрицы в операциях над ней			В		О				НМ	5
21. В операциях над матрицами участвуют матрицы неподходящей размерности			В		О				НМ	5
22. Для переменной, принимающей множество значений, неправильно указаны индексы			В		О				НМ	5

23. Результат вычислений не сохраняется			В		О			НМ	5
24. Используется неверный логический оператор			В	О				НМ	
25. Неправильно вычисляется индекс			В		О			НМ	5
26. Программа работает с ошибками, когда вычисления производятся на стыке двух годов		В					О	НМ	8
27. Программа работает с ошибками, когда вычисления производятся на стыке двух суток		В					О	НМ	8
28. Нуль с отрицательным знаком используется для контроля неправильно			В			О		НМ	6
Л. ОШИБКИ МОДИФИЦИРОВАНИЯ И ИНДЕКСАЦИИ									
1. Условия, отображаемые индексами, несовместимы		В			О			НМ	5
2. В индексном регистре по ошибке устанавливается отрицательное число			В	ЗВ-8			О	4	7
3. Содержимое индексного регистра не изменяется			В				О	НМ	7
4. Указатели очереди не устанавливаются в исходное положение			В	ЗВ-6	О			4	5
5. Указатели очереди устанавливаются в исходное положение неправильно			В	О				НМ	4
6. Значение указателя очереди не увеличивается			В		О			НМ	5
7. Значение указателя очереди не уменьшается			В		О			НМ	5
8. Значения указателей очереди обновляются неправильно			В			О		НМ	6
9. Значения указателей очереди обновляются несвоевременно			В			О		НМ	6

Продолжение

Категории и типы ошибок \ Этапы создания программного обеспечения	Выработка требований	Проектирование	Кодирование и отладка	Рабочие испытания	Утверждение	Приемочные испытания	Стыковка с другими программами	Сдача пользователю	k_i	t_j
10. Переполнение очереди			В	ЗВ-8		О			4	6
11. Происходит обращение к пустой очереди			В	ЗВ-8		О			4	6
12. Входные сообщения ставятся в очередь неправильно			В			О			НМ	6
13. Сообщения исключаются из очереди неправильно			В			О			НМ	6
14. В качестве индекса использована неподходящая переменная			В	О					НМ	4
15. Индексы использованы не в том порядке			В	О					НМ	4
16. Нуль использован в качестве индекса неверно			В	ЗВ-8					4	4
17. Индексы, использованные в операциях над матрицами, имеют большее значение, чем размерность самой матрицы			В	ЗВ-8	О				4	5
18. Индексы вычисляются неправильно			В	ЗВ-8	О				4	5
19. Индексы внутри цикла изменяются неправильно			В		О				НМ	5

20. В результате изменений индекса процесс выполнения программы не доводится до конца			В		О				НМ	5
М. ОШИБКИ ИТЕРАТИВНЫХ ПРОЦЕДУР										
1. Выполняется излишне большое число итераций			В		О				НМ	5
2. Выполняется очень мало итераций			В		О				НМ	5
3. Возможно бесконечное повторение операций			В		О				5	5
4. Не выполняется ни одной итерации			В	О	ЗВ-3+				НМ	4
5. Неверно передается управление в цикле		В		СТ-1	О				4	5
6. Происходит изменение переменной цикла			В	СТ-2+		О			4	6
7. Неправильно делается вложение циклов		В		СТ-1	О				4	5
8. Не определен критерий сходимости итерационного процесса		В				О			НМ	6
9. Критерий сходимости определен неверно		В				О			НМ	6
10. После завершения цикла предпринимаются неверные действия		В				О			НМ	6
11. Неправильно определена величина шага		В				О			НМ	6
12. Неправильно выполняется проверка условий выхода из цикла			В		О				НМ	5
13. Индексы цикла обновляются неправильно			В		О				НМ	5

Продолжение

Категории и типы ошибок \ Этапы создания программного обеспечения	Выработка требований	Проектирование	Кодирование и отладка	Рабочие испытания	Утверждение	Приемочные испытания	Стыковка с другими программами	Сдача пользователю	h_j	i_j
14. Цикл выполняется с неверными данными			В	О					НМ	4
15. Вычисления внутри цикла не контролируются посредством индекса цикла		В		ОС-2				О	4	8
Н. ОШИБКИ ПОРАЗРЯДНОЙ ОБРАБОТКИ ДВОИЧНЫХ СЛОВ										
1. Разряды формируются неправильно			В		О				НМ	5
2. Разряды изменяются неправильно			В		О				НМ	5
3. Многоразрядное слово не устанавливается в исходное положение		В		ЗВ-1	О				4	5
4. Многоразрядное слово устанавливается в начальное положение неверно		В			О				НМ	5
5. Для управления процессом используется не тот разряд		В			О				НМ	5
6. Для проверки содержимого разрядов используется не та процедура		В			О				НМ	5

Таблица 5.5

Степень корректирующей способности различных показателей качества программного обеспечения с точки зрения ускорения обнаружения ошибок различных типов

Показатели завершенности		Показатели согласованности		Прочие показатели	
ЗВ-1	ЗВ-3+	СГ-1	ИН-1+	КМ-2	СТ-1
Ж1 8/4	Л20 5/5	321 5/4	В6 4/2	Б2 7/4	М5 5/4
З3 4/4	М3 5/5	322 5/4	В7 4/2	Б3 4/4	М7 5/4
З25 7/4	ЗВ-6	СГ-2+	В12 6/2	Б21 4/4	ИН-7
З26 5/4	Л4 5/4	Е5 6/4	В13 4/2	Б22 6/4	И6 8/4
Н3 5/4	ЗВ-7	М6 6/4	Г4 4/4	Б23 8/4	ДС-5
ЗВ-1+	А13 6/4	СГ-5	Г6 5/2	В5 5/4	И12 8/6
Б13 5/4	А16 5/4	329 8/4	Е10 8/2	И5 7/4	ТЧ-1
Б14 5/4	А18 5/4	СГ-11	ИН-1++	КМ-2+	К4 6/4
Б24 8/4	И2 6/4	312 8/4	Г8 6/4	Б7 8/4	ОС-2
В11 8/4	ЗВ-7+	ИН-1+	Г9 6/4	КМ-4	М15 8/4
Е8 5/4	Б6 8/4	А1 6/2	Г14 6/4	Д2 6/4	
ЗВ-2	ЗВ-8	А2 7/2	Г15 6/4	Д3 8/4	
Б20 8/4	Л2 7/4	А3 7/2	Г18 4/4	Д5 7/4	
В8 4/4	Л10 6/4	А4 6/2	Г21 5/4	Д9 4/4	
ЗВ-2+	Л11 6/4	А5 6/2	Е1 4/4	Е15 5/4	
Б5 4/4	Л16 4/4	А12 6/2	Е2 5/4	КМ-4+	
Б16 4/4	Л17 5/4	Б4 7/2	Е3 5/4	Д7 6/4	
Ж5 8/4		Б8 7/2	З20 8/4	Д8 6/4	
ЗВ-3	ЗВ-9	Б9 7/2		КМ-4++	
К14 5/4	А9 8/4	Б17 7/2		И1 6/4	
К15 5/4	А10 8/4	Б3 5/4		И10 6/4	
	З4 6/4				
	З6 4/4				
	ЗВ-10				
	З11 8/4				
	ЗВ-14				
	Е11 5/4				
	Е15 5/4				

Примечание. В каждой колонке буквенно-цифровое обозначение соответствует шифру категории и типу ошибки из табл. 5.4, а дробь, стоящая справа, содержит в числителе номер этапа создания программного обеспечения, на котором ошибка обнаруживается без применения метрики, а в знаменателе — номер этапа, на котором она обнаруживается при использовании данного показателя.

Таблица 5.6

Эффективность различных показателей с точки зрения возможностей исправления ошибок различных типов

Показатели	Число типов обнаруживаемых ошибок	Общее выигрываемое количество этапов	Потенциальная выгодность применения показателя (табл. 4.1—4.10)
Показатели завершенности			
ЗВ-1	5	9	5
ЗВ-1+	5	11	5
ЗВ-2	2	4	5
ЗВ-2+	3	4	5
ЗВ-3	2	2	5
ЗВ-3+	2	0	5
ЗВ-6	1	1	5
ЗВ-7	4	6	5
ЗВ-7+	1	1	5
ЗВ-8	5	8	4
ЗВ-9	4	10	4
ЗВ-10	1	4	3
ЗВ-14	2	2	5
Итого по группе	37	62	
Показатели согласованности			
СГ-1	2	2	4
СГ-2+	2	4	5
СГ-5	1	4	5
СГ-11	1	4	3
ИН-1+	18	61	5
ИН-1++	10	14	5
Итого по группе	34	89	
Прочие показатели			
КМ-2	7	13	5
КМ-2+	1	4	5
КМ-4	5	10	5
КМ-4+	2	4	5
КМ-4++	2	4	5
СТ-1	2	2	5
ИН-7	1	4	5
ДС-5	1	2	4
ТЧ-1	1	2	5
ОС-2	1	4	4
Итого по группе	23	49	
Всего	94	200	

ЗВ-3+ *характеризует способность работы программы в условиях, отклоняющихся от номинальных.*

Предусмотрены ли в программе действия «по запасному пути», если номинальные условия не достигаются?

ЗВ-7+ *характеризует возможность контроля по четности.*

Проверяются ли исходные данные путем контроля по четности?

КМ-2+ *характеризует способность программы к автоматическому распознаванию граничных признаков.*

Может ли программа самостоятельно распознавать признаки конца файла, конца ленты и т. п., не требуя от пользователя задания этих границ в явном виде?

КМ-4+ *характеризует качество сообщений об ошибках, выдаваемых программой.*

Предусмотрена ли в программе выдача диагностических сообщений об ошибках с пользой и своевременно?

КМ-4++ *характеризует качество сообщений, выдаваемых программой оператору в процессе работы.*

Своевременно ли выдаются сообщения оператору в рабочем режиме и обладают ли они необходимой четкостью и полезностью?

Пометка НМ в предпоследнем столбце табл. 5.4 означает, что для автоматического обнаружения соответствующего типа ошибок в настоящее время нет метрики.

5.4. ОЦЕНКА ЭФФЕКТИВНОСТИ МЕТРИКИ С ТОЧКИ ЗРЕНИЯ ЭКОНОМИИ ЗАТРАТ

Применение рассмотренной метрики программного обеспечения в процессе разработки крупномасштабных проектов должно приносить не только текущие, но и долгосрочные выгоды. Последние означают экономию затрат на перспективные проекты. Текущие выгоды — это возможное сокращение расходов на текущий проект в ходе отладки программ, рабочих испытаний, ввода в

действие и эксплуатации. Долгосрочные выгоды вкратце рассматриваются в подразд. 5.4.1, а в подразд. 5.4.2 достаточно подробно обсуждаются непосредственные выгоды использования метрики программного обеспечения. В заключение главы в подразд. 5.4.3 приводится аналитическая модель, позволяющая оценивать экономию затрат и выгоды от применения рассмотренных в предшествующих главах показателей качества программного обеспечения.

5.4.1. Долгосрочные выгоды

Основная ценность использования метрики для оценивания определенных свойств больших систем программного обеспечения в ходе разработки множества программ заключается в том, что о каждой программе накапливаются сведения, которые полезны уже сами по себе. Эта полезность состоит в возможности предсказывать необходимые затраты (например, удельную стоимость разработки одной команды в программе для системы реального времени в отличие от системы, работающей не в реальном времени) на основании собранных и соответствующим образом классифицированных данных. Точно так же сбор и классификация данных о конкретных значениях различных показателей помогают экстраполировать и предсказывать свойства и характеристики будущих программных средств. Что же касается использования метрики как средства сокращения затрат, необходимых для удовлетворения требований к программному обеспечению (это и есть текущие выгоды), то согласованное применение показателей качества на различных этапах создания отдельных программ должно в значительной мере способствовать достижению этой цели.

Мы, однако, больше не будем касаться возможных методов изменения, уточнения или разработки заново показателей для оценки свойств программных средств в ходе их создания и эксплуатации. Речь пойдет далее о проблеме оценивания возможной экономии затрат при разработке каждого конкретного проекта.

5.4.2. Текущие выгоды

Выше уже констатировалось, что использование метрики на ранних этапах создания программного обеспечения, а также на этапе ввода его в действие и эксплуатации должно значительно уменьшить количество программных ошибок. Стоимость устранения таких ошибок обычно возрастает по мере продвижения процесса разработки от написания программы к созданию модулей и пакетов программ, объединенных в рамках единой системы. Это происходит потому, что на более поздних стадиях приходится исследовать и документировать последствия ошибок глубже, чем на ранних. В случае применения метрики можно надеяться, что степень возрастания затрат на устранение ошибок удастся уменьшить благодаря разумной оценке необходимой сложности межмодульных сопряжений.

Экономия затрат, о которой идет речь, достигается ценой некоторых дополнительных расходов по применению метрики. Поэтому необходимо проанализировать взаимосвязь между общими затратами на проектирование и приращением расходов по применению метрики. Как отмечалось в ходе обсуждения долгосрочных выгод использования показателей качества программного обеспечения, только оценки, выработанные в результате анализа данных по многим проектам программного обеспечения, могут создать реальную основу для формулирования критериев приемлемости тех или иных систем показателей. Таким образом, мы предполагаем, что соответствующая метрика определена; дополнительные предположения будут введены в последующих рассуждениях.

Затраты по использованию метрики могут включать в себя следующие элементы:

1. Выраженные в человеко-часах трудозатраты по анализу и численной оценке показателей.

2. Для показателей, определяемых автоматически, затраты машинных ресурсов (памяти и процессорного времени), связанные с использованием специальных вспомогательных программных средств на этапах рабочих проверок и тестирования.

3. Затраты труда руководителей, программистов и системных аналитиков в случае низких значений показателей и принятия решения о необходимости внесения существенных изменений в проект, программу и документацию. Обычно низкие оценки показателей должны наряду с другими критериями способствовать принятию таких решений. Если при этом низкие оценки показателей играют в таком решении главенствующую роль, то соответствующие дополнительные затраты должны учитываться при расчете издержек применения метрики.

4. Нарушения сроков выполнения проектных работ, влекущие за собой потерю прогрессивных вознаграждений за своевременное выполнение заданий вследствие дополнительных затрат времени на использование метрики.

5. Управленческие расходы, связанные с гарантированием качества программного изделия и направленные на поддержание должной дисциплины проектирования. Сюда входят затраты, связанные с нормированием процедур проектирования, проведением теоретических занятий, подготовкой отчетов, с обеспечением взаимодействия с функциональным руководством и заказчиками и с оценкой возможных последствий реализации проектных решений. Затраты, связанные с анализом фактических данных по конкретному проекту для использования их в процессе разработки многих других проектов, должны, как это отмечалось выше, частично относиться к каждому из проектов. Что касается экономии затрат, то она должна рассчитываться с учетом как тех ошибок, которых удалось избежать благодаря использованию метрики, так и тех, которых можно было бы избежать в случае применения метрики.

Одним из возможных путей оценивания экономии затрат, связанных с программными ошибками, достигаемой благодаря применению метрики, может быть сравнительный анализ двух или более специально выбранных программ примерно одинаковых по трудоемкости и величине. В процессе их разработки, начиная с этапа выработки требований к программному обеспечению, к некоторым из этих программ применяется соответствующая метрика, а к некоторым нет. Если это воз-

можно, следует избегать существенных различий в квалификации и опыте программистов и других факторов, которые могут существенно исказить результаты сравнительного анализа. Такой подход создает основу для проведения надежного реального эксперимента.

Необходимым условием является сбор более детальных сведений о затратах, чем это имеет место в обычной практике проектирования. Данные о трудозатратах каждого участника разработки должны классифицироваться по видам деятельности, так как обычно один и тот же инженер или программист выполняет в течение дня самые разнообразные задания. Это нужно для того, чтобы отличать изменения, вносимые в программу по результатам оценки с помощью метрики, от изменений, которые вносятся в результате обнаружения ошибки.

Другой возможный способ оценивания сокращения затрат при применении метрики состоит в параллельной разработке двух или более программ двумя разными группами разработчиков на основе одной и той же спецификации требований. Эти программы могут быть встроены в «систему» программного обеспечения, состоящую из действующей операционной системы, библиотечных программ, некоторых других системных программ и подпрограмм, к которым обращаются наблюдаемые программы. Вероятнее всего, многие из подпрограмм будут вызываться каждой из сопоставляемых программ. Программы, участвующие в эксперименте, должны быть выполнены на основе существенно различающихся принципов программирования: например, к одной программе может предъявляться требование минимума команд, а другая создаваться без каких-либо ограничений. Следует ожидать, что программы, созданные в условиях разных ограничений, будут характеризоваться существенно различной хронологией ошибок (это результат подобного наблюдения в одном из недавних исследований, выполненных фирмой TRW, которое касалось количественного измерения безопасности и надежности программного обеспечения [1]).

---

[1]) TRW Systems, Rpt.No.SDP-1776, The Quantitative Measurement of Software Safety and Reliability, 24 August 1973.

5.4.3. Модель для анализа экономии затрат, получаемой в результате применения метрики качества программного обеспечения

Приведенная ниже простая модель была применена к данным, собранным в ходе исследования, касавшегося изучения характеристик процесса разработки программного обеспечения и выполненного фирмой TRW по заказу исследовательской группы проекта CCIP-85 ВВС США. Эти данные были описаны и проанализированы в разд. 5.3. При выполнении анализа был принят параметрический подход, в соответствии с которым стоимость ошибок считается зависящей от этапа разработки и от затрат по применению метрики. Кроме того, было сделано предположение о том, что в 100% случаев использование метрики позволяет обнаружить ошибку не позже, чем это делается в отсутствие какой-либо метрики. Существует возможность оценки также и процента случаев, в которых метрика обнаруживает ошибки конкретного типа, но такой анализ будет более надежным после дальнейшего совершенствования и оценки системы показателей, являющейся предметом рассмотрения в данной книге. Во многих случаях применялись именно эти показатели, в ряде случаев требовалось их расширение. В некоторых случаях был сделан вывод о невозможности использования в настоящее время какого-либо показателя, позволяющего с достаточной уверенностью гарантировать выявление ошибки конкретного типа.

5.4.3.1. Анализ модели

В модели выделяются следующие 8 этапов создания программного обеспечения:

1. Выработка требований.
2. Проектирование.
3. Программирование.
4. Отладка и рабочее тестирование.
5. Утверждение.
6. Приемочные испытания.
7. Стыковочные системные испытания.

8. Передача заказчику — ввод в действие и эксплуатация.

Эти наименования этапов несколько отличаются от тех, которые рассматривались нами выше: причина заключается в том, что данные, подлежавшие анализу, были первоначально упорядочены именно по таким этапам.

Затраты, связанные с обнаружением всех ошибок данного типа на каждом из перечисленных этапов, равны C_i, $i=1, 2, \ldots, 8$, и обычно

$$C_1 < C_2 < \ldots < C_8.$$

Хотя модель может быть легко распространена на случай различных стоимостей ошибок разных типов, это улучшение нами не рассматривается.

Предположим, что ошибка j-го типа при отсутствии метрики возникает на этапе i_j. Считается, что использование метрики приводит к обнаружению ошибки типа j на этапе k_j. Издержки по применению метрики на этапах до k_j включительно равны

$$M_{k_j} = \sum_{i=1}^{k_j} m_i,$$

где m_i — затраты по применению метрики на этапе i для каждого типа ошибок.

Экономия затрат, получаемая благодаря применению метрики к ошибкам j-го типа, равна $C_{i_j} - C_{k_j}$. Из этой суммы экономии должны быть вычтены затраты M_{k_j}. Следовательно, чистая экономия по j-му типу ошибок составит:

$$K_j = C_{i_j} - C_{k_j} - M_{k_j},$$

а по всем типам ошибок:

$$K_T = \sum_{j=1}^{N} K_j,$$

где N — общее количество различных типов ошибок.

Вероятно, наиболее удобно иметь дело с безразмерной оценкой эффективности метрики, равной отношению

экономии затрат по применению метрики к стоимости ошибки в случае отсутствия метрики:

$$C_{R_j}=\frac{C_{i_j}-C_{k_j}-M_{k_j}}{C_{i_j}}$$

Очевидно, что

$$C_{R_j}\leqslant 1{,}0$$

и может быть отрицательным. Более высокие значения C_{R_j} означают более высокую эффективность. Средняя величина экономии определяется соотношением

$$\bar{C}_T=\frac{1}{N}\sum_{i=1}^{N}C_{R_i}.$$

Для простоты примем допущение, что стоимости ошибок каждого типа возрастают линейно в зависимости от номера этапа разработки, так что

$$C_i=C_0 i.$$

Предполагается также для простоты, что затраты по применению метрики не зависят от этапа создания программного обеспечения и потому

$$m_i=m, \quad i=1, 2, \ldots, 8,$$

откуда

$$M_{k_j}=mk_j$$

и, следовательно,

$$K_j=C_0(i_j-k_j)-mk_j=C_0 i_j-k_j(C_0+m).$$

Подставив этот результат в формулу для C_{R_j}, получим

$$C_{R_j}=\frac{C_0 i_j-k_j(C_0+m)}{C_0 i_j}=$$
$$=1-\left(\frac{C_0+m}{C_0}\right)\left(\frac{k_j}{i_j}\right).$$

откуда

$$\overline{C}_T = 1 - \frac{1}{N}\left(1 + \frac{m}{C_0}\right)\sum_{j=1}^{N}\frac{k_j}{i_j} = 1 - \frac{1}{N}(1+r)\sum_{j=1}^{N}\frac{k_j}{i_j},$$

где $r = m/C_0$.

Основой для проверки модели послужили данные, приведенные в табл. 5.4. Типы ошибок, для которых нет метрики, в расчет не принимались. Например, для выполнения расчетов по первому типу ошибок из категории ошибок подготовки и ввода данных с перфокарт следовало бы принять $k_1 = 2$, $i_1 = 6$. Заметим, что если какой-то тип ошибок не может быть обнаружен или исправлен посредством применения метрики на более ранних этапах, чем без нее, то затраты по ее применению на всех предшествующих этапах становятся чистыми потерями и экономия оказывается отрицательной:

$$K_j = C_{i_j} - C_{i_j} - M_{i_j} = -M_{i_j}.$$

Так, например, обстоит дело с ошибками типа 5 из категории ошибок обработки данных при чтении с магнитной ленты.

5.4.3.2. Оценка на основе использования приведенных данных

Из данных табл. 5.4 следует, что

$$\sum_{j=1}^{N}\frac{k_j}{i_j} = 62{,}5857,$$

где $N = 94$. Отсюда

$$\overline{C}_T = 1 - \frac{1}{94}(1+r)\,62{,}5857,$$

или

$$\overline{C}_T = 0{,}33419 - 0{,}66580r.$$

Таким образом, ключевым параметром рассмотренного анализа является отношение метрика/тестирование:

$$r = \frac{m}{\overline{C}_0},$$

в числителе которого стоит стоимость применения метрики, отнесенная к одному этапу и одному типу ошибок, а в знаменателе — стоимость традиционного тестирования, отнесенная к одному этапу и одному типу ошибок.

Полученное соотношение говорит о том, что при $r < \frac{0,33419}{0,66580} = 0,5019$ возможна экономия за счет использования метрики. Следовательно, при сделанных выше предположениях можно утверждать, что, если применение метрики обходится менее чем в 50,19% стоимости обычного тестирования, ее надо использовать, так как затраты оказываются оправданными.

ЗАКЛЮЧЕНИЕ

Мы рассмотрели конкретную совокупность мероприятий, подлежащих осуществлению на различных стадиях жизненного цикла программного обеспечения, которые оказывают значительное влияние на уровень его качества. Эти мероприятия включают:

— установление в явном виде целевых параметров качества и их относительных приоритетов;

— осуществление нормирования характеристик качества программных средств;

— применение контрольных таблиц и вопросников по анализу качества;

— введение специальных мер, гарантирующих высокое качество программного изделия;

— использование таких стандартов разработки технических требований, которые пригодны для машинного анализа;

— разработка матриц «Требования — свойства»,

— установление стандартов работы, особенно в отношении структурного программирования;

— применение автоматического «Программного ревизора» для контроля программ на соответствие установленным стандартам;

— осуществление периодических контрольных проверок качества проектных решений и программ.

В книге приведено большое количество показателей качества программного обеспечения, которые соответ-

ствующим образом упорядочены и оценены с точки зрения их потенциальной выгодности, возможности количественного выражения и автоматизации процесса их получения.

На основе этих показателей построена четкая иерархическая структура непересекающихся характеристик качества программного обеспечения. Ее верхний уровень непосредственно отражает потребности пользователей, а нижний — содержит элементарные характеристики, тесно коррелированные с реальными свойствами программного обеспечения, которые могут оцениваться по соответствующим показателям.

И хотя современное состояние проблемы оценки качества программных средств таково, что мы имеем лишь ограниченные возможности автоматизации этого процесса, все же специальное внимание к характеристикам качества программного обеспечения может дать значительную экономию затрат на всех этапах его создания.

ПРИЛОЖЕНИЕ А

Примеры наличия (положительные) или отсутствия (отрицательные) у программ свойств, оцениваемых предложенной метрикой

ЗАВЕРШЕННОСТЬ

ЗВ-1 Отрицательный пример

```
      DO 100 I=1,99
      A(I+1)=10.*A(I)+Y(I)
  100 CONTINUE
```

Ошибка состоит в том, что A(1) не устанавливается в исходное состояние.

ЗВ-2 Во многих вычислительных системах при открытии файла или обращении к совокупности данных не предусматривается установка их в начальную позицию. Поэтому программа должна сама выбирать нужное положение носителя информации, с тем чтобы гарантировать считывание нужных данных.

ЗВ-3 Положительный пример 1

```
      J=0
      .
      .
      .
C     ВЕТВЬ ДЛЯ J<1
      IF (J. LT. 1) GO TO 5
      DO 100 I=1,J
      .
      .
      .
```

```
      100 CONTINUE
      5 CONTINUE
```

Положительный пример 2

```
C     ЕСЛИ СОДЕРЖИМОЕ СЧЕТЧИКА РАВНО
      НУЛЮ, ВЫДАТЬ СООБЩЕНИЕ
      IF (COUNT. GI. B) GO TO 100
      WRITE (SOUT, 9901)
   9901 FORMAT (1H,*COUNT IS. LE. O, C SET TO
      1000.0*)
      C=1000.0
      GO TO 200
    100 C=SUM/COUNT
    200 .
        .
        .
```

ЗВ-4 Описание всех обращений к внешним программам в начальном блоке комментариев облегчает задачу контроля и отладки.

ЗВ-6 Положительный пример

```
      DATA ANGLE/90/, DIST/100/
      READ (SIN, 10) TANGLE, TDIST
   10 FORMAT (...)
C     КОНТРОЛЬ ВВОДИМЫХ ДАННЫХ
      IF(EOF) GO TO 100
C     ЗНАЧЕНИЕ ВВОДИТСЯ, А НЕ ПРИСВАИ-
      ВАЕТСЯ ПО УМОЛЧАНИЮ
      ANGLE=TANGLE
      DIST=TDIST
  100     .
          .
          .
```

ЗВ-7 Положительный пример

```
      READ, WIDTH
      IF (WIDTH. GT. 0) GO TO 100
      WRITE (SOUT, 9901) WIDTH
9901  FORMAT (*WIDTH IS. LE.0*, F20.8)
      CALL ABORT
100   .
      .
      .
```

ЗВ-8 Положительный пример

```
      SUBROUTINE SUMN (N)
      COMMON/ARRAY/NIX, X (10)
C     SUMN СУММИРУЕТ ПЕРВЫЕ N ЭЛЕ-
      МЕНТОВ X
      IF (N.GT. 0) GO TO 100
      SUMN=0
      RETURN
C     КОНТРОЛЬ ПРЕВЫШЕНИЯ В ЗАПРО-
      СЕ ЧИСЛА ЭЛЕМЕНТОВ X
 100  IF (N. GT. NIX) N=NIX
      SUM=0.0
      DO 200 I=1,N
           SUM=SUM+X (I)
 200  CONTINUE
      SUMN=SUM
      RETURN
      END
```

ЗВ-9 Лучше всего проверка допустимых границ осуществляется тогда, когда используется транслятор, обладающий дополнительной способностью

генерировать такую программу, которая контролирует каждое обращение на соответствие объявленным границам параметров. Такой способ может вылиться в некоторый перерасход памяти и машинного времени, но зато будет гарантироваться правильность всех ссылок и обращений.

ЗВ-10 Отрицательный пример

```
      .
      .
      .
      DIMENSION CHEK (8,8), RSUM (8)
      READ (SIN, 1) (RSUM(I), I=1,8)
      READ (SIN,2) ((CHEK(I, J), I=1,8), J=1,8)
    1 FORMAT (8F10.0)
    2 FORMAT (4F20.5/)
      GRAND=0.0
      DO 10 I=1,8
          SUM=0.0
C     СУММИРОВАНИЕ ПО СТРОКЕ
      DO 20 J=1,8
      SUM=SUM+CHEK (I, J)
   20 CONTINUE
      RSUM(I)=SUM
C     СУММИРОВАНИЕ ВСЕХ ЭЛЕМЕНТОВ
      GRAND=GRAND+SUM
   10 CONTINUE
      .
      .
      .
```

В этом примере RSUM вычисляется по входной переменной CHEK, поэтому нет никакой необходимости вводить RSUM. Однако, в тех

случаях, когда точность исходных данных является критическим фактором, может быть оправдан ввод избыточных данных, чтобы обеспечить контроль их совместимости и точности.

ЗВ-11 Применения «запрещенных» или «ключевых» слов в качестве имен переменных следует избегать даже в тех случаях, когда используемый транслятор это разрешает.

ЗВ-12 Отрицательный пример

```
SUBROUTINE DUMMY
RETURN
END
```

ЗВ-13 В начальном блоке комментариев должны быть особо отмечены те блоки и программы, которые специфичны для конкретной ЭВМ.

ЗВ-14 Необходимо обеспечивать возможность повторения выполнения программы от некоторых контрольных точек, а в программах, работающих с большими массивами исходных данных, следует предусматривать средства восстановления данных и анализа ошибок.

СОГЛАСОВАННОСТЬ

СГ-1 Отрицательный пример

```
COMMON/A/A, B, C          COMMON/A/D, C, B
COMMON/A/A(15), B(22)     COMMON/A/DUM(24),
                          KEY
```

СГ-2 Отрицательный пример

REAL*4 VEL(18) в подпрограмме AMAX

REAL*8 VEL(18) в подпрограмме TOTAL

СГ-3 Отрицательный пример

Обращение к первому элементу массива AR, описанное как AR вместо AR(1).

СГ-4 Отрицательный пример

```
X=LOC(I)
```

При такой записи, если LOC нигде не имеет описания размерности и никогда не встречается в левой части операторов присваивания, LOC будет интерпретироваться как обращение к внешней функции. В ходе загрузки программы будет происходить обращение к библиотеке стандартных подпрограмм для отыскания подпрограммы с именем LOC. Если такой подпрограммы не окажется, будет выдано на печать соответствующее сообщение, но, если подпрограмма с именем LOC будет найдена, может произойти все, что угодно.

Примечание: большинство загрузчиков, редакторов связей и компиляторов способны выявлять почти все подобные ошибки.

СГ-5 Положительный пример

Значения физических констант должны задаваться однократно в начале работы программы, а затем обращение к ним должно осуществляться по именам (например, PI для π).

СГ-6 Отрицательный пример

```
      XL=100
      DELX=1.0E—10
      X=XL
      DO 15 J=1, K
         Y(J)=A*(X**2)
         X=X+DELX
   15 CONTINUE
```

Прибавление DELX к X не приводит к существенному изменению его значения.

СГ-7 Отрицательный пример

```
INTEGER FUNCTION R (J, K)
      .
      .
      .
RETURN
```

В вызывающей программе

```
      .
      .
      .
I=R (J, K)
```

СГ-9 Отрицательный пример

Использование в одном месте выражения

```
X=(A*B*C*D)/Q
```

а в другом

```
X=A*B
X=X*C
X=X*D
X=X/Q
```

СГ-10 Отрицательный пример

```
PI=3.14159
PI=Произведение целых чисел
```

СГ-11 Отрицательный пример

```
TOTAL
SUM   } Обозначения одной и той же суммы в
CUM   } разных модулях
```

СГ-12 Отрицательный пример

```
A(1) — значение высоты полета самолета
A(2) — значение скорости полета
A(3) — дата вылета
```

Такое размещение снижает согласованность и понятность.

ДОСТУПНОСТЬ

ДС-1 Положительные примеры

1. Возможна выдача на печать промежуточных результатов.
2. Доступны специальные отладочные средства, позволяющие контролировать правильность структур базы данных, таких, как список связей, стеки, очереди, области приема информации по каналам связи и т. п.

ДС-2 Отрицательный пример

```
CIRCUM=3.14*DIAM
```

Положительный пример

```
DATA/PI/3.14
      .
      .
      .
CIRCUM=PI*DIAM
```

КОММУНИКАТИВНОСТЬ

КМ-1 Для осуществления контроля правильности ввода данных пользователю должна предоставляться дополнительная возможность «эхо-печати».

КМ-2 Положительный пример

Программа, в которой используется свободный формат ввода или оператор NAMELIST.

КМ-3 Положительный пример

Начало программы:

```
INTEGER NAME(8)
READ(5, 1) NAME
```

```
      FORMAT (A80)
      PRINT 1, NAME
```

(Пользователь указывает имя на первой входной карте.)

КМ-4 Положительный пример

```
C     АВАРИЙНЫЙ ОСТАНОВ ПРИ ОТРИЦАТЕЛЬНОМ ЗНАЧЕНИИ СКОРОСТИ
      IF (VEL. GT. 00) GO TO 100
      WRITE (SOUT, 9901) VEL
9901  FORMAT (*VELOCITY IS LESS THAN 0=*,F15.2)
      CALL ABORT
100   CONTINUE
        .
        .
        .
```

КМ-5 Программы, предназначенные для многократной прогонки контрольных примеров, должны использовать операторы BLOCK DATA и/или DATA для задания значений переменных, вероятность изменения которых мала.

КМ-6 Отрицательный пример

```
      IF (DEBUGF .EQ.1) WRITE (SOUT,9901)
9901  FORMAT (...)
```

Здесь DEBUGF является отладочным оператором, относящимся к дополнительным возможностям.

КМ-7 Положительный пример

```
      PRINT 9904, BABAGE
9904  FORMAT (1H, *AGEOF BABY IS*, F10.2)
```

КМ-8 Многие языки программирования и их расширения имеют в своем составе средства, обеспечивающие свободный формат ввода.

КМ-9 Правильное решение состоит в том, чтобы
а) использовать возможности трассировки, имеющиеся в данной системе программирования или языке (например, отслеживание входов в подпрограмму, передач управления, последовательности событий, обращений к переменным и т. п.),
б) при моделировании дискретных процессов обеспечивать возможность трассировки всех событий, событий определенного класса и событий, происходящих в заданном интервале времени;
в) иметь одну или несколько специальных трассировочных переменных и присваивать им конкретный смысл в зависимости от трассируемой программы (хорошим способом создания разнообразия переменных является умножение предыдущего номера на простое число);
г) выпечатывать в соответствующих местах программы необходимые сообщения и значения переменных и/или сами трассировочные переменные.

КМ-10 Положительный пример

Сведения, выдаваемые в виде форматированных таблиц.

СТРУКТУРИРОВАННОСТЬ

СТ-1 Отрицательный пример

Программа, предназначенная для выполнения в Системе IBM 360, в которой не учтено, что в ассемблере регистр 8 используется для указания защищенной области, регистр 9 используется для указания адреса возврата, регистр 10 используется для указания подпрограммы, регистры 0, 1, 2 используются для передачи параметров, регистр 1 используется для возврата вычисленной величины.

СТ-6 Положительный пример

```
      C НАЧАЛЬНАЯ УСТАНОВКА СЧЕТЧИКА I01
        50 J=BUFFP—1                  I01
           CALL AOWTOST (CBCRR, CBCJID) I01
      C    ПЕРЕХОД, ЕСЛИ БУФЕР ОСВОБОЖ- I01
           ДАЕТСЯ
        60 IF (J.GT.NB) GO TO 9903    I01
           IF (J.EQ.NB) GO TO 7006    I01
      C    ИЗМЕНЕНИЕ СОСТОЯНИЯ СЧЕТЧИКА
           БУФЕРА                     I01
           J=J+2                      I01
      C      ПЕРЕХОД В СЛУЧАЕ ЗАНЯТОСТИ
             БУФЕРА                   I01
           IF (BUFFT (J).LE.0) GO TO 60 I01
      C    ПЕРЕХОД ПРИ НЕПРАВИЛЬНОМ   I01
           УКАЗАНИИ БУФЕРА
           IF (TABS (BUFFT (J)).NE.CBCBT) I01
           GO TO 60
      C      ЗАНЯТИЕ БУФЕРА           I01
      7051 BUFFT (J)=—BUFFT (J)       I01
```

СТ-8 Положительный пример

В основной программе:

```
PROGRAM MAIN
COMMON . . .
DATA . . .
CALL INPUT
CALL CMPUT 1
CALL CMPUT 2
CALL CMPUT 3
CALL OUTPUT
END
```

ИНФОРМАТИВНОСТЬ

ИН-2 Положительный пример

```
C     ЕСЛИ УГОЛ КУРСА СОСТАВЛЯЕТ
      90 ГРАДУСОВ ВЫЗЫВАЕТСЯ СПЕ-
C     ЦИАЛЬНАЯ ПРОГРАММА ИНТЕГРИ-
C     РОВАНИЯ УРАВНЕНИЯ ДВИЖЕНИЯ
C     В ПРОТИВНОМ СЛУЧАЕ ИСПОЛЬ-
C     ЗУЕТСЯ ПОЛУЧЕННАЯ ВЕЛИЧИНА
C     INTGR
      IF (FPANGL.EQ.90) GO TO 100
      CALL INTGR
      GO TO 200
  100 CALL INTGR90
  200 .
      .
      .
```

ИН-3 Положительный пример

```
SUBROUTINE SAM (A, B, C, D)
COMMON/TEST/ITEST
IF(ITEST.EQ.1)PRINT 1, A, B, C, D
  .
  .   } АЛГОРИТМ ВЫЧИСЛЕНИЙ
  .
IF (ITEST.EQ.1) PRINT 2, A, B, C, D
  .
  .   } ПРОДОЛЖЕНИЕ АЛГОРИТМА ВЫ-
  .   } ЧИСЛЕНИЙ
  IF (ITEST.EQ.1) PRINT 3, A, B
  RETURN
1 FORMAT (...)
2 FORMAT (...)
```

```
      3 FORMAT (...)
        END
```

Для выполнения тестирования вызывающей программе достаточно установить в «1» значения TEST COMMON и ITEST.

ИН-5 Начальный блок комментариев модуля должен содержать описание условий его связи с другими подпрограммами.

ИН-6 Положительный пример

Наличие специальных пояснений к изменяемым блокам программ и подпрограмм:

```
C ***ОБЩИЙ БЛОК COM1***
C ***5 СЛОВ ИСПОЛЬЗУЮТСЯ ТАКЖЕ
  В SUBR1, SUBR4***
C ***ОБЩИЙ БЛОК COM2***
C ***2463 СЛОВА ИСПОЛЬЗУЮТСЯ ТАКЖЕ
  В SUBR5***
```

ИН-7 Положительный пример

```
      SUBROUTINE RAND(X)
C ДО ЗАПУСКА RAND ЧИСЛО ДОЛЖНО
  БЫТЬ ПРЕДВАРИТЕЛЬНО СФОРМИРО-
C ВАНО ПОДПРОГРАММОЙ INITIA В
C ДАЛЬНЕЙШЕМ БЕРЕТСЯ ПОСЛЕДНЕЕ
C ЧИСЛО СФОРМИРОВАННОЕ ПОДПРО-
C ГРАММОЙ RAND
      COMMON/RAND/NUM
      .
      .  } Блок, использующий сформированное
      .  } число
      END
```

ИН-8 Положительный пример

```
PI обозначает π
V обозначает скорость
и т. п.
```

ИН-9 Положительный пример

```
C     ВЫЧИСЛЕНИЕ СРЕДНЕГО ВРЕМЕНИ ОБСЛУЖИВАНИЯ
      SUM=0.0
      DO 100 I=1, NOFCAR
  100 SUM=SUM+SERT(I)
      AVSERT=SUM/NOFCAR
```

ИН-10 Положительный пример тот же, что и для ИН-2.

ИН-11 Отрицательный пример

```
      GO TO 5
      A=B+C
    5 CONTINUE
```

Оператор A=B+C недостижим и потому должен быть исключен. (Примечание: большинство компиляторов, как правило, выявляют подобную ошибку.)

Положительный пример

```
      .
      .
      .
C БЛОК, СЛЕДУЮЩИЙ ЗА ОПЕРАТОРОМ
C 10, ПРЕДНАЗНАЧЕН ДЛЯ ТРАССИРОВКИ
   10 CONTINUE
      .
      .
      .
      END
```

Отметим, что передачи управления оператору 10 может и не быть, однако назначение этого оператора объяснено.

ОСМЫСЛЕННОСТЬ

ОС-1 Отрицательный пример

```
GO TO 100
N=3
.
.
.
N=2
IF (N. EQ. 2) GO TO 200
N=3
```

Два оператора, присваивающие N значение 3, нереализуемы. Отметим, что большинством компиляторов первый из этих нереализуемых операторов выявляется.

ОС-2 Отрицательный пример

```
DO 10 I=1,10
PI=4.0*ATAN (1.0)
X (I)=2.*PI*R (I)*R (I)
10 CONTINUE
```

должно кодироваться так:

```
PI=4.0*ATAN (1.0)
DO 10 I=1, 10
X (I)−2.*PI*R (I)*R (I)
10 CONTINUE
```

ОС-3 Отрицательный пример

```
P=Z* ((A*B)/D)
Q=X* ((A*B)/D)
```

Положительный пример

```
T=(A*B)/D
P=Z*T
Q=P*T
```

ОС-4 См. алгоритм вычисления Индекса непонятности на стр. 123

ОС-5 Отрицательный пример

```
    DO 10 I=1, N
5   X(I)=A(I)+B(I)
10  CONTINUE
```

Оператор X(I)=A(I)+B(I) не нуждается в метке. Метки должны использоваться только в точках ветвления, точках сопряжения и в операторах формата.

ОС-6 Отрицательный пример

```
DIMENSION ALTVEL (100,4)
```

Положительный пример

```
DIMENSION ALT (100), VELX (100), VELY
(100), VELZ (100)
```

ОС-7 Отрицательный пример

```
    WRITE (SOUT, 10) I
10  FORMAT (I10)
    WRITE (SOUT, 20) I, J
20  FORMAT (2I10)
```

Положительный пример

```
WRITE (SOUT, 20) I
WRITE (SOUT, 20) I, J
FORMAT (2I10)
```

ОС-8 Отрицательный пример

```
      IF (A—B) 20, 20, 30
```

лучше кодировать так:

```
      IF (A.LE.B) GO TO 20
30
      .
      .
      .
```

Многие компиляторы в первом случае генерируют более эффективную программу, однако вторая запись отличается большей понятностью.

ОТКРЫТОСТЬ

ОТ-1 Положительный пример

Фрагмент цикла с оператором DO:

```
C     НАЧАЛО ВЛОЖЕННОГО ЦИКЛА
      DO 10 I=1, 10
         K=J (I)*L (I)
         ANS=0.0
         DO 20 N=1, K
            ANS=ANS+B (N)*C (N)
20    CONTINUE
         X (I)=ANS/Y (I)
C     КОНЕЦ ЦИКЛА
10    CONTINUE
```

ОТ-2 Отрицательный пример

В одних машинах A**B**C трактуется как (A**B)**C, а в других — как A**(B**C).

ОТ-3 Положительный пример

```
      SUBROUTINE HIQUAL (ALPHA, BETA,
      GAMMA, DELTA)
```

```
1 BETA=ALPHA+1
2 GAMMA=3*BETA+2
3 DELTA=ALPHA+BETA+GAMMA
  RETURN
  END
```

Отрицательный пример

```
  SUBROUTINE LOQUAL (ALPHA, BETA,
  GAMMA, DELTA)
3 BETA=ALPHA+1
2 GAMMA=3*BETA+2
1 DELTA=ALPHA+BETA+GAMMA
  RETURN
  END
```

ОТ-4 Положительный пример

При написании программы на языке ассемблера необходимо присваивать операторам легко различимые мнемонические метки; в приводимом примере такой меткой является PLUS.

```
PL      X6, PLUS    Передача управления
        .           оператору PLUS,
        .           если X6— положи-
        .           тельное число
        .
        .
PLUS    SAG X6
```

В текстах Фортран-программ (или написанных на любом другом языке свободного формата) передачи управления должны выноситься вправо, чтобы при необходимости их можно было быстро обнаруживать, например:

```
IF (N.GT.10)    GO TO 20
IF(A—B)         20, 30, 40
```

ОТ-5 Отрицательный пример

В одной подпрограмме:

```
REAL A(15), B(15), C(75)
```

в другой подпрограмме:

```
REAL C(75), A(15)
REAL B(15)
```

Положительный пример

Когда программа рассчитана на работу с большой по размерам структурированной базой данный, в которой многократно используются описательные операторы COMMON (в больших программах это частое явление), глобальный описательный оператор COMMON должен быть тщательно продуман. До начала трансляции подпрограммы препроцессор (или программа-утилита) должен вставить в нее глобальный описатель COMMON. Такой метод исключает ошибки, причиной которых является неправильное размещение блока COMMON, многократные описания переменных и т. п. При этом одновременно обеспечивается большее удобство эксплуатации, бóльшая понятность, согласованность и осмысленность. Вот пример такого описания:

```
SUBROUTINE I01                                          I01
      INTEGER SP,T                                      I01
C///EVT              БЛОК СОБЫТИЙ                      GLOBE
      INTEGER EVBLK, RTNST, EVENT, STIME                GLOBE
      COMMON/EVT/EVBLK, IT, RTNST, ITR, JTR,            GLOBE
      LCOPY, EVENT, STIME                               GLOBE
C///ENT      ЭЛЕМЕНТЫ ОПИСАНИЯ СОБЫТИЙ                  GLOBE
      DIMENSION JID(1), QET(1), RETB(1), CPNO(1),       GLOBE
     *CEPU(1), RIFL(1), CALR(1), CALRB(1)               GLOBE
      INTEGER QET, RETB, CPNO, CEPU, RIFL,              GLOBE
     *CALR, CALRB                                       GLOBE
      EQUIVALENCE (JID, WORD (2)),                      GLOBE
```

```
     (QET,   WORD(3)),   (RETB,   WORD(4)), GLOBE
     *(CPNO,  WORD(5)),  CEPU,    WORD(6)), GLOBE
     *(RIFL,  WORD(7)),  (CALR,   WORD(8)), GLOBE
     *(CALRB, WORD(9))                      GLOBE
C///MEM   ДИНАМИЧЕСКИЙ БЛОК ПАМЯТИ          GLOBE
     DIMENSION IV (5200)                    GLOBE
     COMMON/MEM/TIME, IV                    GLOBE
```

ОТ-7 Отрицательный пример

В некоторых языках программирования и ряде вычислительных систем имеются модифицированные трансляторы, позволяющие располагать в одной строке более одного оператора. Это значит, что

D=A*E

A=B*C

может быть закодировано как

D=A*E $ A=B*C

Подобной возможностью пользоваться не следует.

ОТ-8 Отрицательный пример

```
C    ВЫБОР УГЛА СНОСА В ЗАВИСИМОСТИ
C    ОТ ВЫСОТЫ
     IF (ALT—CHEIGHT) 30, 40, 50
30   ANGLE=30.0       $ GO TO 60
40   ANGLE=45.0       $ GO TO 60
50   ANGLE=60.0       $ GO TO 60
60   CONTINUE
                .
                .
                .
```

В случаях, подобных этому примеру, увеличивается понятность, но это достигается ценой потери независимости от конкретного транслятора.

ОТ-9 Требуемые правила могут быть такими:

1. Применять разделители и описательные операторы.
2. В программах, написанных на языке ассемблера, использовать возможности пространственного разделения, разбиения на страницы и присвоения заголовков.
3. Внутри одного блока присваивать операторам увеличивающиеся номера.
4. Операторам формата присваивать номера из отдельного ряда чисел.
5. Точки входа в программу и обращения к подпрограммам снабжать комментариями в целях более легкого распознавания.

ОТ-10

Положительный пример

```
C///RESTAB          ТАБЛИЦА РЕСУРСОВ                    GLOBE
      DIMENSION NRO(25), NRA(25), NIQ(25)               GLOBE
      INTEGER RQS, SRQE                                 GLOBE
      GOMMON/RESTAB/NRS, RQS, SPQE, NRO,                GLOBE
     *NRA, NIQ, ICPQS, PRFLA, PJSRS, PICSA,             GLOBE
      PRO, PRI
C///NEWQ    ФОРМАТ НОВОЙ ОЧЕРЕДИ ЗАДАНИЙ               GLOBE
      DIMENSION LTPA(1), TIN(1), PIN(1), JP(1),         GLOBE
      UPR(1)                                            GLOBE
      INTEGER TIN, PIN, UPR                             GLOBE
      EQUIVALENCE (LTPA, WORD(4)), (TIN,                GLOBE
     *WORD(9)), (PIN, WORD(10)), (JP, WORD(11)),        GLOBE
     *(UPR, WORD(12))                                   GLOBE
C///ORQE  ФОРМАТ СВЕРТЫВАНИЯ ОЧЕРЕДИ ЗА-               GLOBE
          ДАНИЙ
C///ROWTBL          ТАБЛИЦА СВЕРТЫВАНИЯ                 GLOBE
      DIMENSION CPN(35), JCRO(35), FL(35)               GLOBE
      INTEGER CPN, FL                                   GLOBE
      COMMON/ROWTBL/ CPN, JCRO, FL                      GLOBE
C///SCHWTBL РАБОЧАЯ ТАБЛИЦА ПЛАНИРОВЩИ-                 GLOBE
          КА ЗАДАНИЙ
```

ПРИЛОЖЕНИЕ Б

Аннотированная библиография

Abernathy D. H., et al., Survey of Design Goals for Operating Systems, Georgia Institute of Technology Report GTIS 72—04.

В работе обсуждаются некоторые из характеристик качества программного обеспечения, рассмотренных в данной книге, и проблема обеспечения секретности. Анализируются компромиссные решения и возможные конфликтные ситуации при разработке операционных систем.

Alberts D. S., The Economics of Software Quality Assurance, Proceedings 1976 NCC, pp. 433—442.

В статье дается широкое толкование термина «гарантия качества», который сопутствует любым методам управления разработкой программного обеспечения, проектирования, программирования и оценивания программных средств, которые направлены на уменьшение вероятности возникновения ошибок. Рассматриваются экономические аспекты, оправдывающие деятельность по обеспечению качества на более ранних этапах создания программного обеспечения и усиление внимания к этой деятельности.

Boehm B. W., Some Steps Toward Formal and Automated Aids to Software Requirements Analysis and Design, Proceedings, IFIP Congress 1974, pp. 192—197.

В статье обсуждается методология проектирования с использованием матриц «Требования — свойства», которая была рассмотрена нами в гл. 5. Излагаются также вопросы разработки автоматических средств анализа требований к создаваемым программным средствам и их проектирования.

Boehm B. W., McClean R. L., Urfrig D. B., Some Experience with Automated Aids to the Design of Large-Scale Reliable Software, *IEEE Trans. Software Engr.*, March 1975, pp. 125—133.

В статье обсуждаются вопросы обоснования целесообразности использования, проектирования и усовершенствования, а также результаты успешного применения анализатора согласованности проектных решений (DACC) — автоматического средства разработки программного обеспечения, основой которого служат принципы автоматического анализа ошибок, изложенные в гл. 5.

Brown J. R., et al., The Quantitative Measurement of Software Safety and Reliability, TRW Systems Group, One Space Park, Redondo Beach, CA, August 24, 1973.

Отчет об исследовательской работе, выполненной с целью разработки методологии оценки вероятности возникновения тактических

ошибок в системе управления местной противоракетной обороной и/или ядерными ударами. Была разработана предварительная методика, охватывающая модель свойств программного обеспечения и их метрику для априорной оценки степени подверженности программных средств ошибкам, измерение глубины структурного анализа программного обеспечения в процессе испытаний, математическую трактовку характеристик надежности и безопасности программных средств и способы их измерения.

Brown J. R., et al., Automated Software Quality Assurance, Program Test Methods, Prentice-Hall, 1973, Chapter 15.

Работа посвящена рассмотрению ряда автоматических средств, предназначенных для обеспечения качества крупных проектов и относящихся к функциям анализа конфигурации программного обеспечения и глубины его тестирования. Обсуждаются результаты успешного применения этих средств.

Brown J. R., et al., Impact of Modern Programming Practices on System Development, RADC-TR-77-121.

Отчет, в котором дается краткий обзор современной практики программирования, использованной при разработке очень крупного проекта программного обеспечения; приведены результаты обследования 67 разработчиков с точки зрения влияния применяемых ими методов работы на стоимость и качество программных средств. Авторами сделан вывод, что использование таких методов обеспечения качества, как автоматическая проверка на соответствие нормативам и стандартам, оказывает заметное положительное влияние на характеристики удобства эксплуатации и надежности.

Culpepper L. M., A. System for Reliable Engineering Software, *IEEE Trans. Software Engr.*, 174—178 (June 1975).

В статье описывается программа «Ревизор», которая автоматически проверяет Фортран-программы на наличие нарушений стандартных требований к модульности, мобильности и согласованности.

Dijkstra E. W., GO TO Statement Considered Harmful, *Communications of the ACM*, **11**, No. 3, 147—148 (March 1968).

В статье высказывается ряд соображений относительно характеристики структурированности как свойства программного обеспечения. Автор утверждает, что квалификация программистов есть убывающая функция частоты использования ими в программах оператора GO TO, большое количество которых свидетельствует о беспорядочном написании программы.

Fagan M. E., Design and Code Inspections to Reduce Errors in Program Development, *IBM Syst. J.*, **15**, No. 3, 182—211 (1976).

В статье говорится о том, что существенного повышения качества и производительности труда программистов можно достигнуть благодаря введению формальных проверок проектных решений и программ. Определены процедуры и роли участников таких проверок; приведены контрольные вопросники и форматы подлежащих сбору данных. На основе анализа примеров утверждается, что повышение производительности должно достигаться за счет значительного уменьшения объема работы по устранению ошибок.

Fisher D., A. Common Programming Language for DoD—Background and Technical Requirements, Institute for Defence Analysis, Paper P-1191, June 1976.

Исследовательский отчет, являющийся прекрасным примером заблаговременного планирования уровня качества программного обеспечения и определения технологических требований к нему. Обсуждаются компромиссные решения в отношении обеспечения качества (например, компромисс между эффективностью и легкостью программирования и надежностью), принятые при разработке единого языка программирования для проектов министерства обороны. Приводятся специфические требования к языку, которые должны обеспечивать его свойства.

Gilb T., Software Metrics, Winthrop, Inc., and Studentlitteratur AB, 1976.

В книге дается полезная информация, касающаяся широкого набора показателей качества и характеристик программного обеспечения. Проанализировано около 44 различных возможных показателей, которые сгруппированы в соответствии со свойствами надежности, гибкости, структурированности, эффективности, трудоемкости и др. Обсуждается ряд методов повышения качества программного обеспечения, в том числе некоторые из рассмотренных в нашей книге и в ряде смежных работ Т. Гилба.

Gunning R., How to Take the Fog Out of Writing, Dartuell Press, Inc. Chicago Ill., 1962.

Очень легко читаемая брошюра, убедительно показывающая, как можно количественно оценить понятность письменных текстов.

Halstead M., Elements of Software Science, Elsevier-North-Holland, Inc., 1977.

Книга отмечает значительный прогресс в работе по установлению и утверждению общей меры сложности программного обеспечения. В ней обсуждаются основные компоненты меры сложности и приводятся последние результаты в области анализа корреляции этих компонентов с длиной программы, языком программирования, производительностью разработчиков и частотой ошибок.

Kernighan B. W., Plauger P. J., The Elements of Programming Style, McGraw-Hill, N. Y., 1974.

В книге предпринята попытка определить «хороший стиль программирования» и показать, как программы, написанные в хорошем стиле, могут быть более легкими для чтения и понимания, часто более короткими и более эффективными. Авторы отмечают, что даже «структурированные» программы могут не обладать такими свойствами. Приведено и проанализировано большое количество примеров. Одна из характеристик качества, на которую авторы обращают особое внимание, это «живучесть», которая означает, по мнению авторов, способность программы правильно функционировать при граничных значениях входных данных и защищать себя от неверных данных.

Kosy D. W., Air Force Command and Control Information Processing in the 1980s: Trends in Software Technology, The Rand Corporation, R-1012-PR, June 1974.

Этот отчет представляет собой расширенный вариант отчета по проекту CCIP-85 в отношении методологии оценки технологии разработки программного обеспечения. В нем анализируется текущее состояние и перспективы решения проблемы обеспечения таких характеристик качества программного обеспечения, как надежность, приемлемость, адаптируемость и целостность в аспектах затрат и сроков разработки. Даются рекомендации по усовершенствованию современных методов разработки программного обеспечения посредством проведения более глубоких исследований.

Lientz B. P., Swanson E. B., Tompkins G. E., Characteristics of Application Software Maintenance, CACM, 1977.

В статье приводятся результаты обследования 69 организаций, занимающихся созданием программного обеспечения в связи с проблемой эксплуатации программных средств. Из 24 потенциальных источников затруднений авторы оценивают качество программной документации и программы соответственно баллами 2 и 4.

Manley J. H., Lipow M., Findings and Recommendations of the Joint Logistics Commanders' Software Reliability Work Group, Vols. I, II, AFSC TR-75-05, November 1975.

Отчет, в котором представлены мнения группы из 30 специалистов по вычислительной технике, высказанные в ходе совещаний представителей правительственных, промышленных и научных кругов в период с октября 1974 по май 1975 года. На этих совещаниях определялись проблемы пополнения вычислительных ресурсов министерства обороны США и намечались планы их решения. План, выработанный рабочей группой по надежности программного обеспечения (SRWG) в отношении совершенствования процесса приобретения программных средств, был утвержден министерством обороны почти целиком. Особое внимание уделено проектам усовершенствований технологии разработки программного обеспечения, которые должны привести к созданию эффективных методов обеспечения высокого качества, надежности и удобства эксплуатации.

McCracken D. D., Weinberg G. M., How to Write a Readable FORTRAN Program, *Datamation*, 73—77 (October 1972).

Статья призывает к адекватному документированию программ, с тем чтобы анализирующий их человек мог легко определять основное их назначение. Указывается, что полный комплект документации программного обеспечения должен отображать целый ряд факторов, таких, как назначение программы, полный набор блок-схем, полный набор входных и выходных данных, включая контрольные примеры и тесты, множество рабочих инструкций для оператора ЭВМ. Исследованы и некоторые другие аспекты качества программных средств.

Myers G. J., Software Reliability-Principles and Practices, Wiley, New York, 1976. (Имеется перевод: Майерс Г., Надежность программного обеспечения.— М.: Мир, 1980.)

В этой книге рассматриваются концепции надежности программного обеспечения, принципы проектирования надежных программных средств, их испытания, а также ряд смежных вопросов, касающихся методов управления разработками, эффективности языков программирования, влияния языков и архитектуры ЭВМ на надежность программ, доказательство правильности функционирования программ и моделей надежности. Однако вопросы гарантирования качества программного обеспечения не затрагиваются. Все рассмотренные в книге вопросы характеризуются равной глубиной анализа и степенью сложности, дается много полезных контрольных вопросников и большое количество примеров, однако упражнения для читателей отсутствуют. Книга написана в непринужденной манере и доставляет удовольствие при чтении.

Rubey R. J., Hartwick R. D., Quantitative Measurement of Program Quality, Proceedings, 23rd International Conference, ACM, 1968, pp. 671—677.

В статье определяется и обсуждается множество характеристик программного обеспечения и их показателей. Рассмотрено 57 характеристик, объединенных в семь категорий, отражающих характер математических вычислений, алгоритм программы, время вычислений, использование памяти, степень изменяемости программ и т. п. Метрика реализуется в виде формул и детально анализируется.

Sadowski W. L., Lozier D. W., A. Unified Standards Approach to Algorithm Testing, Program Test Methods, W. C. Hetzel (ed.), Prentice-Hall, Inc., Englewood Cliffs, N. J., 1973, pp. 277—290.

В книге показывается, как достигается мобильность основополагающих алгоритмов в отношении желаемого уровня точности за счет использования основных стандартов в начале деятельности по проектированию, при смежных передачах документации и в работе, что позволяет «градуировать» свойства разрабатываемого программного обеспечения.

Stepczyk F. M., Requirements for Secure Operating Systems, TRW Software Series, TRW-SS-74-05, June 1974.

Отчет о результатах исследований, касающихся составляющих факторов безопасности ЭВМ, в число которых входят изолированность, идентификация пользователей, контроль за доступом, инспектирование, обеспечение целостности и текущее обслуживание. Детально излагаются требования к указанным факторам и анализируются способы их удовлетворения.

Thayer T. A., Lipow M., Nelson E. C., Software Reliability Study, TRW Software Series, TRW-SS-76-03, March 1976.

Это заключительный отчет об исследованиях по проблеме надежности программного обеспечения, выполненных фирмой TPW для Римского исследовательского центра командования ВВС США. В отчете представлены результаты анализа данных об ошибках в программном обеспечении, собранных в ходе разработки четырех проектов. Целью анализа данных было определение возможных характерных типов ошибок, изучение эффективности различных стратегий разработки и испытаний программных средств с точки зрения

предотвращения и обнаружения ошибок соответственно, а также анализ надежности программного обеспечения в процессе его практического использования. [1)]

Warren J. C., Jr, Software Portability, Stanford Digital Systems Laboratory Technical Note No. 48, Stanford University, September 1974.

В работе рассматриваются основные концепции и факторы, определяющие мобильность программного обеспечения (такие, как абстрактные машины, самостоятельная загрузка программ, имитация, эмуляция, языки высокого уровня, псевдопрограммы), современные способы использования этих концепций и выдвигаются предложения по дальнейшим исследованиям в области поиска новых принципов повышения мобильности программного обеспечения.

Weinberg G. M., The Psychology of Improved Programmer Performance, *Datamation*, 82—85 (November 1972).

В статье дается краткий обзор результатов нескольких экспериментов, показывающих, какое большое значение имеет установление целевых параметров качества. Описываются эксперименты, в которых нескольким группам программистов ставились одинаковые задачи, но по-разному оценивалось качество работы. По каждому заданному свойству программного обеспечения наивысшие результаты достигались той группой, для которой обеспечение данного свойства служило критерием оценки качества ее работы.

Wulf W. A., Report of Workshop No.3 'Methodology', Symposium on the High Cost of Software, Carnegie-Mellon University, September 17, 1973, pp. 1—12.

В статье обсуждаются качественные различия программ и исследуются способы последовательного улучшения их качества. Рассматриваются вопросы абсолютной стоимости программного обеспечения, его стоимости по отношению к техническим средствам и изменения его стоимости во времени. Определен ряд необходимых свойств программного обеспечения: удобство эксплуатации, модифицируемость, живучесть, четкость, эффективность, приемлемая стоимость и мобильность.

ЛИТЕРАТУРА

1. Elshoff J. L., An Analysis of Some Commercial PL/1 Programs, *IEEE Trans. Sofware Engineering*, 113—120 (June 1976).
2. Improvements Needed in Managing Automated Decisionmaking by Computers Throughout the Federal Government, U. S. General Accounting Office, April 23, 1976.
3. Rubey R. J., Hartwick R. D., Quantitative Measurement of Program Quality, Proceedings, ACM National Conference, 1968 pp. 671—677.
4. Brown J. R., Lipow M., The Quantitative Measurement of Soft-

[1)] Книга переводится в издательстве «Мир».— *Прим. ред.*

ware Safety and Reliability, revised from TRW Report No. SDP-1776, August 1973.
5. Wulf W. A., Programming Methodology, Proceedings of a Symposium on the High Cost of Software, J. Goldberg (ed.), Stanford Research Institute, September 1973.
6. Abernathy D. H., et al., Survey of Design Goals for Operating Systems, Georgia Institute of Technology, Report GTIS-72-04.
7. Weinberg G. M., The Psychology of Improved Programmer Performance, *Datamation*, 82—85 (November 1972).
8. Kernighan B. W., Plauger P. J., The Elements of Programming Style, McGraw-Hill, 1974.
9. A Study of Fundamental Factors Underlying Software Maintenance Problems, CIRAD, Inc., December 1971.
10. Research Toward Ways of Improving Software Maintenance, CIRAD, Inc., January 1973.
11. Warren J., Software Portability, Stanford University Digital Systems Laboratory, Technical Note No. 48, September 1974.
12. DeRoze B. C., DOD Defence System Software Management Program, Abridged Proceedings from the Software Management Conference, 1976.
13. Kossiakoff A., Sleight T. P., Software Requirements Analysis and Validation, там же.
14. Whitaker W. A., DOD Common High Order Language (HOL) Program, там же.
15. Light W., Software Reliability/Quality Assurance Practices, там же.
16. Myers G. J., Software Reliability: Principles and Practices, Wiley, New York, 1976.
17. Halstead M. H., Elements of Software Science, Elsevier North-Holland, Inc., 1977.
18. Gilb T., Software Metrics, Studentlitteratur AB, Lund, Sweden and Winthrop, Cambridge, MA, USA, 1976.
19. Liskov B. H., Zilles S. N., Programming with Abstract Data Types, ACM SIGPLAN Notices, April 1974, pp. 50—59.
20. Stepczyk F. M., Requirements for Secure Operating Systems, TRW Software Series, TRW-SS-74-05, June 1974.
21. Boehm B. W., McClean R. K., Urfrig D. B., Some Experiences with Automated Aids to the Design of Large-Scale Software, *IEEE Trans. Sofware Engineering*, 125—133 (March 1975).
22. Teichroew D., Sayari H., Automation of System Building, *Datamation*, 25—30 (August 1971).
23. Alford M. W., Jr., A Requirements Engineering Methodology for Real-Time Processing Requirements, *Trans. Software Engineering*, **SE-3**, No. 1, 63—69 (January 1977).
24. Bell T. E., Bixler D. C., Dyer M. E., An Extendable Approach to Computer-Aided Software Requirements Engineering, ibid., pp. 49—63.
25. Boehm B. W., Some Steps Toward Formal and Automated Aids to Software Requirements Analysis and Design, Proceedings, IFIP Congress, 1975, pp. 192—197.
26. Mills H. D., Mathematical Foundations of Structured Programming, IBM-FSD Report 72-6012, 1972.

27. Brown J. R., Controlling and Measuring Software Quality, Proceedings of the AIIE Conference on Software, July 19—21, 1976.
28. Fagan M. E., Design and Code Inspections to Reduce Errors in Program Development, *IBM Syst. J.*, **15**, No. 3, 182—211 (1976).
29. Thayer T. A., Lipow M., Nelson E. C., Software Reliability Study, TRW Software Series, TRW-SS-76-03, March 1976. (Готовится к выходу в свет на русском языке.)
30. Gunning R., How to Take the Fog Out of Writing, Dartuell Press, Inc, Chicago Ill., 1962.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

ОГЛАВЛЕНИЕ

УВАЖАЕМЫЙ ЧИТАТЕЛЬ!

Ваши замечания о содержании книги, ее оформлении, качестве перевода и другие просим присылать по адресу: 129820, Москва, И-110, ГСП, 1-й Рижский пер., д. 2, издательство «Мир».

Б. Боэм, Дж. Браун, Х. Каспар,
М. Липов, Г. Мак-Леод, М. Мерит

ХАРАКТЕРИСТИКИ КАЧЕСТВА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Старший научный редактор А. Харитонов
Младший научный редактор Н. Титова
Художник А. Козловский
Художественный редактор Л. Безрученков
Технический редактор Н. Толстякова
Корректор Н. Гиря

ИБ № 2317

Сдано в набор 01.08.80. Подписано к печати 15.01.81. Формат 84×108$^{1}/_{32}$. Бумага типографская № 1. Гарнитура латинская. Печать высокая. Объем 3,25 бум. л. Усл. печ. л. 10,92. Уч.-изд. л. 10,40. Изд. № 20/0872. Тираж 15 000 экз. Зак. № 920. Цена 80 коп.

ИЗДАТЕЛЬСТВО «МИР»
Москва, 1-й Рижский пер., 2.

Отпечатано в Ленинградской типографии № 2 головном предприятии ордена Трудового Красного Знамени Ленинградского объединения «Техническая книга» им. Евгении Соколовой Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли. 198052, г. Ленинград, Л-52, Измайловский проспект, 29 с матриц ордена Октябрьской Революции и ордена Трудового Красного Знамени Первой Образцовой типографии имени А. А. Жданова Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли. Москва, М-54, Валовая, 28

80 коп.