Тестирование программных продуктов

**I. ВСТУПЛЕНИЕ.**

**ОБЩИЕ ПОНЯТИЯ.**

Многие организации, занимающиеся созданием программного обеспечения, до 50% средств, выделенных на разработку программ, тратят на тестирование, что составляет миллиарды долларов по всему миру в целом. И все же, несмотря на громадные капиталовложения, знаний о сути тестирования явно не хватает и большинство программных продуктов неприемлемо ненадежно даже после “основательного тестирования”.

О состоянии дел лучше всего свидетельствует тот факт, что большинство людей, работающих в области обработки данных, даже не может правильно определить слово “тестирование”, и это на самом деле главная причина неудач.

“Тестирование — процесс, подтверждающий правильность программы и демонстрирующий, что ошибок в программе нет.” Основной недостаток подобного определения заключается в том, что оно совершенно неправильно; фактически это почти определение антонима слова “тестирование”. Читатель с некоторым опытом программирования уже, вероятно, понимает, что невозможно продемонстрировать отсутствие ошибок в программе. Поэтому определение описывает невыполнимую задачу, а так как тестирование зачастую все же выполняется с успехом, по крайней мере с некоторым успехом, то такое определение логически некорректно. Правильное определение тестирования таково:***Тестирование — процесс выполнения программы с намерением найти ошибки.***

Невозможно гарантировать отсутствие ошибок в нетривиальной программе; в лучшем случае можно попытаться показать наличие ошибок. Если программа правильно ведет себя для солидного набора тестов, нет основании утверждать, что в ней нет ошибок; со всей определенностью можно лишь утверждать, что не известно, когда эта программа не работает. Конечно, если есть причины считать данный набор тестов способным с большой вероятностью обнаружить все возможные ошибки, то можно говорить о некотором уровне уверенности в правильности программы, устанавливаемом этими тестами.

Психологические эксперименты показывают, что большинство людей, поставив цель (например, показать, что ошибок нет), ориентируется в своей деятельности на достижение этой цели. Тестовик подсознательно не позволит себе действовать против цели, т. е. подготовить тест, который выявил бы одну из оставшихся в программе ошибок. Поскольку мы все признаем, что совершенство в проектировании и кодировании любой программы недостижимо и поэтому каждая программа содержит некоторое количество ошибок, самым плодотворным применением тестирования будет найти некоторые из них. Если мы хотим добиться этого и избежать психологического барьера, мешающего нам действовать против поставленной цели, наша цель должна состоять в том, чтобы найти как можно больше ошибок. Сформулируем основополагающий вывод:

Если ваша цель — показать отсутствие ошибок, вы. их найдете не слишком много. Если же ваша цель — показать наличие ошибок, вы найдете значительную их часть.

Надежность невозможно внести в программу в результате тестирования, она определяется правильностью этапов проектирования. Наилучшее решение проблемы надежности — с самого начала не допускать ошибок в программе. Однако вероятность того, что удастся безупречно спроектировать большую программу, бесконечно мала. Роль тестирования состоит как раз в том, чтобы определить местонахождение немногочисленных ошибок, оставшихся в хорошо спроектированной программе. Попытки с помощью тестирования достичь надежности плохо спроектированной программы совершенно бесплодны.

Тестирование оказывается довольно необычным процессом (вот почему оно и считается трудным), так как этот процесс разрушительный. Ведь цель проверяющего (тестовика) — заставить программу сбиться. Он доволен, если это ему удается; если же программа на его тесте не сбивается, он не удовлетворен.

Еще одна причина, по которой трудно говорить о тестировании — это тот факт, что о нем известно очень немногое. Если сегодня мы располагаем 5% тех знании о проектировании и собственно программировании (кодировании), которые будут у нас к 2000 г., то о тестировании нам известно менее 1%.

ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ.

Хотя в тестировании можно выделить несколько различных процессов, такие термины, как тестирование, отладка, доказательство, контроль и испытание, часто используются как синонимы и, к сожалению, для разных людей имеют разный смысл. Хотя стандартных, общепринятых определений этих терминов нет, попытка сформулировать их была предпринята на симпозиуме по тестированию программ. Нашу классификацию различных форм тестирования мы начнем с того, что дадим эти определения, слегка их дополнив и расширив их список.

***Тестирование (testing)****,* как мы уже выяснили,—процесс выполнения программы (или части программы) с намерением (или целью) найти ошибки.

***Доказательство (proof)*** *—* попытка найти ошибки в программе безотносительно к внешней для программы среде. Большинство методов доказательства предполагает формулировку утверждений о поведении программы и затем вывод и доказательство математических теорем о правильности программы. Доказательства могут рассматриваться как форма тестирования, хотя они и не предполагают прямого выполнения программы. Многие исследователи считают доказательство альтернативой тестированию — взгляд во многом ошибочный; более подробно это обсуждается в гл. 17.

***Контроль (verification)*** *—* попытка найти ошибки, выполняя программу в тестовой, или моделируемой, среде.

***Испытание (validation)*** *—* попытка найти ошибки, выполняя программу в заданной реальной среде.

***Аттестация (certification)*** *—* авторитетное подтверждение правильности программы, аналогичное аттестации электротехнического оборудования Underwriters Laboratories. При тестировании с целью аттестации выполняется сравнение с некоторым заранее определенным стандартом.

***Отладка (debugging)*** не является разновидностью тестирования. Хотя слова “отладка” и “тестирование” часто используются как синонимы, под ними подразумеваются разные виды деятельности. Тестирование — деятельность, направленная на обнаружение ошибок; отладка направлена на установление точной природы известной ошибки, а затем — на исправление этой ошибки. Эти два вида деятельности связаны — результаты тестирования являются исходными данными для отладки.

***Тестирование модуля, или автономное тестирование (module testing, unit testing)*** *—* контроль отдельного программного модуля, обычно в изолированной среде (т. е. изолированно от всех остальных модулей). Тестирование модуля иногда включает также математическое доказательство.

***Тестирование сопряжении (integration testing)*** *—* контроль сопряжении между частями системы (модулями, компонентами, подсистемами).

***Тестирование внешних функций (external function testing)*** *—* контроль внешнего поведения системы, определенного внешними спецификациями.

***Комплексное тестирование (system testing)*** *—* контроль и/или испытание системы по отношению к исходным целям. Комплексное тестирование является процессом контроля, если оно выполняется в моделируемой среде, и процессом испытания, если выполняется в среде реальной, жизненной.

***Тестирование приемлемости (acceptance testing)*** *—* проверка соответствия программы требованиям пользователя.

***Тестирование настройки (installation testing)*** — проверка соответствия каждого конкретного варианта установки системы с целью выявить любые ошибки, возникшие в процессе настройки системы.

Отношения между этими типами тестов и проектной документацией, на которой основывается тест, показаны на рис.3,



Рис. 2. Спектр подходов к проектированию тестов,



Рис. 3. Процессы тестирования и их связь с процессами проектирования.

**II. ОСНОВНАЯ ЧАСТЬ.**

**ФИЛОСОФИЯ ТЕСТИРОВАНИЯ**

Тестирование программного обеспечения охватывает целый ряд видов деятельности, весьма аналогичный последовательности процессов разработки программного обеспечения. Сюда входят постановка задачи для теста, проектирование, написание тестов, тестирование тестов и, наконец, выполнение тестов и изучение результатов тестирования. Решающую роль играет проектирование теста. Возможен целый спектр подходов к выработке философии, или стратегии проектирования тестов, изображенный на рис.2. Чтобы ориентироваться в стратегиях проектирования тестов, стоит рассмотреть два крайних подхода, находящихся на границах спектра. Следует отметить также, что многие из тех, кто работает в этой области, часто бросаются в одну или другую крайность.

Сторонник (или сторонница) подхода, соответствующего левой границе спектра, проектирует свои тесты, исследуя внешние спецификации или спецификации сопряжения программы или модуля, которые он тестирует. Программу он рассматривает как черный ящик. Позиция его такова: “Меня не интересует, как выглядит эта программа и выполнил ли я все команды или все пути. Я буду удовлетворен, если программа будет вести себя так, как указано в спецификациях”. Его идеал — проверить все возможные комбинации и значения на входе.

Приверженец подхода, соответствующего другому концу спектра, проектирует свои тесты, изучая логику программы. Он начинает с того, что стремится подготовить достаточное число тестов для того, чтобы каждая команда была выполнена по крайней мере один раз. Если он немного более искушен, то проектирует тесты так, чтобы каждая команда условного перехода выполнялась в каждом направлении хотя бы раз. Его идеал — проверить каждый путь, каждую ветвь алгоритма. При этом его совсем (или почти совсем) не интересуют спецификации.

Ни одна из этих крайностей не является хорошей стратегией. Читатель, однако, уже, вероятно, заметил, что первая из них, а именно та, в соответствии с которой программа рассматривается как черный ящик, предпочтительней. К сожалению, она страдает тем недостатком, что совершенно неосуществима. Рассмотрим попытку тестирования тривиальной программы, получающей на входе три числа и вычисляющей их среднее арифметическое. Тестирование этой программы для всех значений входных данных невозможно. Даже для машины с относительно низкой точностью вычислений количество тестов исчислялось бы миллиардами. Даже имей мы вычислительную мощность, достаточную для выполнения всех тестов в разумное время, мы потратили бы на несколько порядков больше времени для того, чтобы эти тесты подготовить, а затем проверить. Такие программы, как системы реального времени, операционные системы и программы управления данными, которые сохраняют “память” о предыдущих входных данных, еще хуже. Нам потребовалось бы тестировать программу не только для каждого входного значения, но и для каждой последовательности, каждой комбинации входных данных. Поэтому исчерпывающее тестирование для всех входных данных любой разумной программы неосуществимо.

Эти рассуждения приводят ко второму фундаментальному принципу тестирования: тестирование — ***проблема в значительной степени экономическая****.* Поскольку исчерпывающее тестирование невозможно, мы должны ограничиться чем-то меньшим. Каждый тест должен давать максимальную отдачу по сравнению с нашими затратами. Эта отдача измеряется вероятностью тою, что тест выявит не обнаруженную прежде ошибку. Затраты измеряются временем и стоимостью подготовки, выполнения и проверки результатов теста. Считая, что затраты ограничены бюджетом и графиком, можно утверждать, что искусство тестирования, по существу, представляет собой искусство отбора тестов с максимальной отдачей. Более того, каждый тест должен быть представителем некоторого класса входных значений, так чтобы его правильное выполнение создавало у нас некоторую убежденность в том, что для определенного класса входных данных программа будет выполняться правильно. Это обычно требует некоторого знания алгоритма и структуры программы, и мы, таким образом, смещаемся к правому концу спектра.

**ИНТЕГРАЦИЯ МОДУЛЕЙ.**

Вторым по важности аспектом тестирования (после проектирования тестов) является последовательность слияния всех модулей в систему или программу. Эта сторона вопроса обычно не получает достаточного внимания и часто рассматривается слишком поздно. Выбор этой последовательности, однако, является одним из самых жизненно важных решении, принимаемых на этапе тестирования, поскольку он определяет форму, в которой записываются тесты, типы необходимых инструментов тестирования, последовательность программирования модулей, а также тщательность и экономичность всего этапа тестирования. По этой причине такое решение должно приниматься на уровне проекта в целом и на достаточно ранней его стадии.

Имеется большой выбор возможных подходов, которые могут быть использованы для слияния модулей в более крупные единицы. В большинстве своем они могут рассматриваться как варианты шести основных подходов, описанных в следующих шести разделах. Сразу же за ними идет раздел, где предложенные подходы сравниваются по их влиянию на надежность программного обеспечения.

**ВОСХОДЯЩЕЕ ТЕСТИРОВАНИЕ.**

При восходящем подходе программа собирается и тестируется снизу вверх. Только модули самого нижнего уровня (“терминальные” модули; модули, не вызывающие других модулей) тестируются изолированно, автономно. После того как тестирование этих модулей завершено, вызов их должен быть так же надежен, как вызов встроенной функции языка или оператор присваивания. Затем тестируются модули, непосредственно вызывающие уже проверенные. Эти модули более высокого уровня тестируются не автономно, а вместе с уже проверенными модулями более низкого уровня. Процесс повторяется до тех пор, пока не будет достигнута вершина. Здесь завершаются и тестирование модулей, и тестирование сопряжении программы.

При восходящем тестировании для каждого модуля необходим *драйвер:* нужно подавать тесты в соответствии с сопряжением тестируемого модуля. Одно из возможных решении — написать для каждого модуля небольшую ведущую программу. Тестовые данные представляются как “встроенные” непосредственно в эту программу переменные и структуры данных, и она многократно вызывает тестируемый модуль, с каждым вызовом передавая ему новые тестовые данные. Имеется и лучшее решение: воспользоваться программой тестирования модулей — это инструмент тестирования, позволяющий описывать тесты на специальном языке и избавляющий от необходимости писать драйверы.

**НИСХОДЯЩЕЕ ТЕСТИРОВАНИЕ.**

Нисходящее тестирование (называемое также нисходящей разработкой не является полной противоположностью восходящему, но в первом приближении может рассматриваться как таковое, При нисходящем подходе программа собирается и тестируется сверху вниз. Изолировано тестируется только головной модуль. После того как тестирование этого модуля завершено, с ним соединяются (например, редактором связей) один за другим модули, непосредственно вызываемые им, и тестируется полученная комбинация. Процесс повторяется до тех пор, пока не будут собраны и проверены все модули.

При этом подходе немедленно возникает два вопроса: что делать, когда тестируемый модуль вызывает модуль более низкого уровня (которого в данный момент еще не существует), и как подаются тестовые данные. Ответ на первый вопрос состоит в том, что для имитации функций недостающих модулей программируются *модули-заглушки”,* которые моделируют функции отсутствующих модулей. Фраза “просто напишите заглушку” часто встречается в описании этого подхода, но она способна ввести в заблуждение, поскольку задача написания заглушки” может оказаться трудной. Ведь заглушка редко сводится просто к оператору RETURN, поскольку вызывающий модуль обычно ожидает от нее выходных параметров. В таких случаях в заглушку встраивают фиксированные выходные данные, которые она всегда и возвращает. Это иногда оказывается неприемлемым, так как вызывающий модуль может рассчитывать, что результат вызова зависит от входных данных. Поэтому в некоторых случаях заглушка должна быть довольно изощренной, приближаясь по сложности к модулю, который она пытается моделировать.

Интересен и второй вопрос: в какой форме готовятся тестовые данные и как они передаются программе? Если бы головной модуль содержал все нужные операции ввода и вывода, ответ был бы прост:

Тесты пишутся в виде обычных для пользователей внешних данных и передаются программе через выделенные ей устройства ввода. Так, однако, случается редко. В хорошо спроектированной программе физические операции ввода-вывода выполняются на нижних уровнях структуры, поскольку физический ввод-вывод — абстракция довольно низкого уровня. Поэтому для того, чтобы решить проблему экономически эффективно, модули добавляются не в строго нисходящей последовательности (все модули одного горизонтального уровня, затем модули следующего уровня), а таким образом, чтобы *обеспечить функционирование операций физического ввода-вывода как можно быстрее.* Когда эта цель достигнута, нисходящее тестирование получает значительное преимущество: все дальнейшие тесты готовятся в той же форме, которая рассчитана на пользователя.

Остается еще один вопрос: в какой форме пишутся тесты до тех пор, пока не будет достигнута эта цель? Ответ: они включаются в некоторые из заглушек.

Нисходящий метод имеет как достоинства, так и недостатки по сравнению с восходящим. Самое значительное достоинство — в том, что этот метод совмещает тестирование модуля, тестирование сопряжении и частично тестирование внешних функций. С этим же связано другое его достоинство — когда модули ввода-вывода уже подключены, тесты можно готовить в удобном виде. Нисходящий подход выгоден также в том случае, когда есть сомнения относительно *осуществимости* программы в целом или если в проекте программы могут оказаться серьезные дефекты.

Преимуществом нисходящего подхода очень часто считают отсутствие необходимости в драйверах; вместо драйверов вам просто следует написать “заглушки”. Как читатель сейчас уже, вероятно, понимает, это преимущество спорно.

Нисходящий метод тестирования имеет, к сожалению, некоторые недостатки. Основным из них является тот, что модуль редко тестируется досконально сразу после его подключения. Дело в том, что основательное тестирование некоторых модулей может потребовать крайне изощренных заглушек. Программист часто решает не тратить массу времени на их программирование, а вместо этого пишет простые заглушки и проверяет лишь часть условий в модуле. Он, конечно, собирается вернуться и закончить тестирование рассматриваемого модуля позже, когда уберет заглушки. Такой план тестирования определенно не лучшее решение, поскольку об отложенных условиях часто забывают.

Второй тонкий недостаток нисходящего подхода состоит в том, что он может породить веру в возможность начать программирование и тестирование верхнего уровня программы до того, как вся программа будет полностью спроектирована. Эта идея на первый взгляд кажется экономичной, но обычно дело обстоит совсем наоборот. Большинство опытных проектировщиков признает, что проектирование программы — процесс итеративный. Редко первый проект оказывается совершенным. Нормальный стиль проектирования структуры программы предполагает по окончании проектирования нижних уровней вернуться назад и подправить верхний уровень, внеся в него некоторые усовершенствования или исправляя ошибки, либо иногда даже выбросить проект и начать все сначала, потому что разработчик внезапно увидел лучший подход. Если же головная часть программы уже запрограммирована и оттестирована, то возникает серьезное сопротивление любым улучшениям ее структуры. В конечном итоге за счет таких улучшений обычно можно сэкономить больше, чем те несколько дней или недель, которые рассчитывает выиграть проектировщик, приступая к программированию слишком рано.

МОДИФИЦИРОВАННЫЙ НИСХОДЯЩИЙ МЕТОД

Нисходящий подход имеет еще один существенный недостаток, касающийся полноты тестирования. Предположим, что есть большая программа и где-то ближе к нижнему ее уровню находится модуль, предназначенный для вычисления корней квадратного уравнения. Для заданных входных переменных *А*, *В* и С он решает уравнение .



При проектировании и программировании модуля с такой функцией всегда следует понимать, что квадратное уравнение может иметь как действительные, так и комплексные корни. Для полной реализации этой функции необходимо, чтобы результаты могли быть действительными или комплексными числами (или, если дополнительные затраты на нахождение комплексных корней не оправданы, модуль должен по крайней мере возвращать код ошибки в случае, когда входные коэффициенты задают уравнение с комплексными корнями). Предположим, что конкретный контекст, в котором используется модуль, исключает комплексные корни (т. е. вызывающие модули никогда не задают входных параметров, которые привели бы к комплексным корням). При строго нисходящем методе иногда бывает невозможно тестировать модуль для случая комплексных корней (или тестировать ошибочные условия). Можно попытаться оправдывать это тем, что, поскольку такое уравнение никогда не будет дано модулю, никого не должно заботить, работает ли он и в этих случаях. Да, это безразлично сейчас, но окажется важным в будущем, когда кто-то попытается использовать модуль в новой программе или модифицировать старую программу так, что станут возможными и комплексные корни.

Эта проблема проявляется в разнообразных формах. Применяя нисходящее тестирование в точном соответствии с предыдущим разделом, часто невозможно тестировать определенные логические условия, например ошибочные ситуации или защитные проверки. Нисходящий метод, кроме того, делает сложной или вообще невозможной проверку исключительных ситуаций в некотором модуле, если программа работает с ним лишь в ограниченном контексте (это означает, что модуль никогда не получит достаточно полный набор входных значений). Даже если тестирование такой ситуации в принципе осуществимо, часто бывает трудно определить, какие именно нужны тесты, если они вводятся в точке программы, удаленной от места проверки соответствующего условия.

Метод, называемый модифицированным нисходящим подходом, решает эти проблемы: требуется, чтобы каждый модуль прошел автономное тестирование перед подключением к программе. Хотя это действительно решает все перечисленные проблемы, здесь требуются и драйверы, и заглушки для каждого модуля.

МЕТОД БОЛЬШОГО СКАЧКА.

Вероятно, самый распространенный подход к интеграции модулей — метод “большого скачка”. В соответствии с этим методом каждый модуль тестируется автономно. По окончании тестирования модулей они интегрируются в систему все сразу.

Метод большого скачка по сравнению с другими подходами имеет много недостатков и мало достоинств. Заглушки и драйверы необходимы для каждого модуля. Модули не интегрируются до самого последнего момента, а это означает, что в течение долгого времени серьезные ошибки в сопряжениях могут остаться необнаруженными. Метод большого скачка значительно усложняет отладку.

И все же большой скачок не всегда нежелателен. Если программа мала и хорошо спроектирована, он может оказаться приемлемым. Однако для крупных программ метод большого скачка обычно губителен.

МЕТОД САНДВИЧА

Тестирование методом сандвича представляет собой компромисс между восходящим и нисходящим подходами. Здесь делается попытка воспользоваться достоинствами обоих методов, избежав их недостатков.

При использовании этого метода одновременно начинают восходящее и нисходящее тестирование, собирая программу как снизу, так и сверху и встречаясь в конце концов где-то в середине. Точка встречи зависит от конкретной тестируемой программы и должна быть заранее определена при изучении ее структуры. Например, если разработчик может представить свою систему в виде уровня прикладных модулей, затем уровня модулей обработки запросов, затем уровня примитивных функций, то он может решить применять нисходящий метод на уровне прикладных модулей (программируя заглушки вместо модулей обработки запросов), а на остальных уровнях применить восходящий метод.

Применение метода сандвича - это разумный подход к интеграции больших программ, таких, как операционная система или пакет прикладных программ.

Метод сандвича сохраняет такое достоинство нисходящего и восходящего подходов, как начало интеграции системы на самом раннем этапе. Поскольку вершина программы вступает в строй рано, мы, как в нисходящем методе, уже на раннем этапе получаем работающий каркас программы. Поскольку нижние уровни программы создаются восходящим методом, снимаются те проблемы нисходящего метода, которые были связаны с невозможностью тестировать некоторые условия в глубине программы.

МОДИФИЦИРОВАННЫЙ МЕТОД САНДВИЧА.

При тестировании методом сандвича возникает та же проблема, что и при нисходящем подходе, хотя здесь она стоит не так остро. Проблема эта в том, что невозможно досконально тестировать отдельные модули. Восходящий этап тестирования по методу сандвича решает эту проблему для модулей нижних уровней, но она может по-прежнему оставаться открытой для нижней половины верхней части программы. В модифицированном методе сандвича нижние уровни также тестируются строго снизу вверх. А модули верхних уровней сначала тестируются изолированно, а затем собираются нисходящим методом. Таким образом, модифицированный метод сандвича также представляет собой компромисс между восходящим и нисходящим подходами.

СРАВНИТЕЛЬНАЯ ХАРАКТЕРИСТИКА МЕТОДОВ ТЕСТИРОВАНИЯ.

С точки зрения надежности программного обеспечения эти стратегии можно оценить по восьми критериям, как показано на рис. 10.7. Первый критерий — время до момента сборки модулей, поскольку это важно для обнаружения ошибок в сопряжениях и предположениях модулей о свойствах друг друга. Второй критерий — время до момента создания первых работающих “скелетных” версий программы, поскольку здесь могут проявиться главные дефекты проектирования. Третий и четвертый критерии касаются вопроса о том, необходимы ли заглушки, драйверы и другие инструменты тестирования. Пятый критерий—мера параллелизма, который возможен в начале или на ранних стадиях тестирования. Это интересный вопрос, поскольку необходимость в ресурсах (т. е. программистах) обычно достигает пика на этапах проектирования и программирования модулей.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Восходящий | Нисходящий | Модифицированный нисходящий | Метод большого скачка | Метод сандвича | Модифицированный метод сандвича |
| Сборка | Рано | Рано | Рано | Поздно | Рано | Рано |
| Время до появления работающего варианта программы | Поздно | Рано | Рано | Поздно | Рано | Рано |
| Нужны ли драйверы (новые программы или готовые инструменты) ? | Да | Нет | Да | Да | Частично | Да |
| Нужны ли заглушки | Нет | Да | Да | Да | Частично | Частично |
| Параллелизм в начале работы | Средний | Слабый | Средний | Высокий | Средний | Высокий |
| Возможность тестировать отдельные пути | Легко | Трудно | Легко | Трудно | Средне | Легко |
| Возможность планировать и контролировать последовательность | Легко | Трудно | Трудно | Легко | Трудно | Трудно |

Рис. 10.7. Количественная оценка подход к сборке.

Поэтому важно, чтобы возможность параллельного тестирования появилась ближе к началу, а не концу цикла тестирования.

Шестой критерий связан с ответом на обсуждавшийся ранее вопрос: возможно ли проверить любой конкретный путь и любое условие в программе? Седьмой критерий характеризует сложность планирования, надзора и управления в процессе тестирования. Это связано с осознанием того факта, что тестирование, которым трудно управлять, часто ведет к недосмотрам и упущениям. Время от времени раздаются возражения против нисходящего подхода в связи с тем, что тестирование нижних модулей требует многократных лишних прогонов головных модулей. Однако этот критерий отмечен как несущественный. Хотя лишние прогоны действительно бывают необходимы, возможно также, что во многих случаях, которые кажутся лишними, в действительности воссоздаются несколько разные условия. Эти прогоны могут вскрыть новые ошибки, превращая таким образом недостаток в достоинство. Поскольку этот эффект недостаточно осознан, мы им пренебрегаем.

Теперь оценим наши шесть подходов с помощью перечисленных восьми критериев. Как уже говорилось, такая оценка зависит от конкретного проекта. В качестве исходного приближения для выполнения ваших собственных оценок приведен вариант очень грубой оценки. Прежде всего следует взвесить относительное влияние каждого из восьми критериев на надежность программного обеспечения. Ранняя сборка и раннее получение работающего каркаса программы, а также возможность тестировать любые конкретные условия представляются наиболее важными, поэтому им дается коэффициент 3. Сложность подготовки заглушек, а также сложность планирования и управления последовательностью тестов также важны, поэтому они получают вес 2. Третий критерий, необходимость драйверов, вес 1 ввиду доступности общих инструментов тестирования. Критерий, связанный с параллелизмом работы, также имеет вес 1, потому что, хотя он, может быть, и важен по другим причинам, на надежность сильно не влияет. Восьмой критерий получает коэффициент нуль.

На рис. 10.8 показаны результаты этой оценки. В каждой графе таблицы вес берется со знаком плюс или минус либо не учитывается, в зависимости от того, благоприятно, неблагоприятно или безразлично проявляется соответствующий фактор при рассматриваемом подходе. Модифицированный метод сандвича и восходящий метод оказываются наилучшими подходами, а метод большого скачка— наихудшим. Если способ оценки оказывается близким к вашей конкретной ситуации, следует рекомендовать модифицированный метод сандвича для тестирования больших систем или программ и восходящий подход для тестирования программ малых и средних.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Вес |  | | Восходящий | | Нисходящий | | Модифицированный нисходящий | | Метод большого скачка | | Метод сандвича | | Модифицированный метод сандвича |
| 3 | Сборка | | Рано + | | Рано + | | Рано + | | Поздно - | | Рано + | | Рано + |
| 3 | Время до появления работающего варианта программы | | Поздно - | | Рано + | | Рано + | | Поздно - | | Рано + | | Рано + |
| 1 | Нужны ли драйвера (новые программы u/uли готовые инструменты) ? | | Да - | | Нет + | | Д а - | | Да - | | Частично | | Да - |
| 2 | Нужны заглушки? | | Нет + | | Да - | | Да - | | Да - | | Частично | | Частично |
| 1 | Параллелизм в начале работы | | Средний | | Слабый- | | Средний | | Высокий+ | | Средний | | Высокий + |
| 3 | Возможность тестировать отдельные пути | | Легко + | | Трудно - | | Легло + | | Легко + | | Средне | | Легко + |
| 2 | Возможность планировать и контролировать последовательность | | Легко + | | Трудно - | | Трудно - | | Легко + | | Трудно - | | Трудно - |
| 0 | Неэффективность | |  | |  | |  | |  | |  | |  |
| Всего | | +6 | | -1 | | +4 | | -3 | | +4 | | +7 | |

Рис. 10.8. Взвешенная оценка подходов к сборке.

**III. ИСПЫТАНИЕ ПРОГРАММНЫХ ПРОДУКТОВ (АНАЛИЗ).**

ЦЕЛЬ И ОСОБЕННОСТИ ИСПЫТАНИИ.

Испытания являются важнейшим элементом управления качеством продукции. В соответствии с ГОСТ 16504—81 под *испытанием промышленной продукции* понимают экспериментальное определение количественных и/или качественных характеристик объекта испытания как результата воздействия на него; при его функционировании; при моделировании объекта и/или воздействия. Под ***испытанием программной продукции*** следует понимать экспериментальное определение количественных и/или качественных характеристик свойств продукции при ее функционировании в реальной среде и/или моделировании среды функционирования.

***Целью испытания*** является экспериментальное определение фактических (достигнутых) характеристик свойств испытываемого ПИ. Эти характеристики могут быть как количественными, так и качественными. Важно, чтобы на их основе можно было сделать вывод о пригодности данного ПИ к использованию по своему назначению. Если вывод отрицательный, то образец ПИ возвращается на доработку. Таким образом перекрывается доступ недоброкачественной продукции к пользователю, Непосредственно в ходе испытаний качество ПИ может и не измениться, так как локализация ошибок не является целью испытания. Вместе с тем некоторые дефекты в программах и документации могут устраняться по ходу испытания.

Испытание является завершающим этапом разработки. Ему предшествует этап статической и динамической отладки программ. Основным методом динамической отладки является тестирование. В узком смысле цель тестирования состоит в обнаружении ошибок, цель же отладки—не только в обнаружении, но ив устранении ошибок. Однако ограничиться только отладкой программы, если есть уверенность в том, что все ошибки в ней устранены, нельзя. Цели у отладки и испытания разные. Полностью отлаженная программа может не обладать определенными потребительскими свойствами и тем самым быть непригодной к использованию по своему назначению. Не может служить альтернативой испытанию и проверка работоспособности программы на контрольном примере, так как программа, работоспособная в условиях контрольного примера, может оказаться неработоспособной в других условиях применения. Попытки охватить контрольным примером все предполагаемые условия функционирования сводятся в конечном счете к тем же испытаниям.

В соответствии с ГОСТ 19,004—80 под ***испытанием программ***понимают установление соответствия программы заданным требованиям и программным документам. Это определение построено на предположении, что в техническом задании на разработку программы определены все требования (характеристики), обеспечение которых гарантирует пригодность программы к использованию по своему назначению. Но такое требование редко соблюдается на практике. В некоторых случаях, особенно в автоматизированных системах, ТЗ на ПС либо вообще не пишут, либо в них перечисляют лишь функции, которые возлагаются на ПС, без указания требований к другим потребительским свойствам. При отсутствии ТЗ на разработку ПС или полного и обоснованного перечня требований к характеристикам разрабатываемого ПС задача испытания ПС становится неопределенной и неконструктивной. Что значит установить соответствие программы заданным требованиям, если эти требования формально не заданы? Какая польза от установления такого соответствия, если эти требования заведомо “усечены” и не отражают основных потребительских свойств программы? Пользователю будет не легче, если программа функционирует плохо, но это в явном виде не противоречит требованиям ТЗ.

При наличии в ТЗ требуемых характеристик основных потребительских свойств ПИ приведенные определения термина “испытание” по цели испытания практически совпадают. Однако и в этом случае первое определение является более конструктивным, так как оно формулирует не только цель, но и основной метод проведения испытании — проверка ПИ, функционирующего в реальной или моделируемой, но близкой к реальной среде,

В зарубежной литературе, в том числе в стандартах на программное обеспечение, понятие “испытание” часто отождествляют с понятием “тестирование”. Например, в Std IEEE 829—1983 “Документация тестов программного обеспечения” (США) дано следующее определение тестирования: “...процесс активного анализа ПО на предмет обнаружения расхождения между реальными и требуемыми нормами ПО (т. е. наличия ошибок в программах) и с целью оценки характеристик элементов ПО”. Данное определение объединяет два приведенных определения термина “испытание” с той лишь разницей, что при принятой (см. определения) концепции поиск и локализация ошибок на являются явно выраженными целями испытания. С учетом высказанных соображений термин “тестирование”, используемый в зарубежной литературе, будем интерпретировать как испытание методом тестирования,

Длительность испытания зависит от типа, конфигурации (сложности) ПС, а также от целей и степени автоматизации рассматриваемого технологического процесса. При испытании операционных систем она колеблется от одного до шести месяцев [20]. Сложные программные комплексы после интеграции могут испытываться и более длительное время.

***Основными видами испытания ПП*** являются предварительные, приемочные и эксплуатационные испытания, включая опытную эксплуатацию. Особенности их организации и проведения подробно рассмотрены в книге [18].

***В зависимости от места проведения*** различают стендовые и полигонные испытания. Под ***испытательным стендом*** понимают совокупность технических устройств и математических моделей, обеспечивающих в автоматическом режиме имитацию среды функционирования; поступление входных данных, искажающие воздействия; регистрацию информации о функционировании ПС, а также управление процессом испытания и объектом испытания. Если в основу стендовых испытаний положен принцип моделирования, то соответствующие испытательные стенды называют моделирующими.

***Испытательным полигоном*** называют место, предназначенное для испытаний в условиях, близких к условиям эксплуатации, и обеспеченное необходимыми средствами испытания. Полигонным испытаниям подвергают системы, работающие в реальном масштабе времени. В полигонных условиях обычно сочетают натурные испытания с использованием реальных объектов автоматизируемых систем и моделирование некоторых объектов и процессов их функционирования. В последнее. Время в некоторых разрабатывающих организациях создают испытательные полигоны, представляющие собой совокупность специализированных по профилю данной организации испытательных стендов. Такие полигоны имеют общую техническую и информационную базы, а также программные средства организации испытаний.

***По степени зависимости испытателей от разработчиков*** различают зависимые и независимые испытания. При зависимых испытаниях основные операции с испытываемыми ПС (подготовка к работе, подготовка и ввод исходных данных, регистрация и анализ результатов) выполняют разработчики программ. Оценку результатов испытания производит комиссия при активном участии разработчиков. Независимые испытания проводят специальные подразделения, не несущие ответственности за разработку программ и непосредственно не подчиняющиеся руководителям разработки.

ТЕХНОЛОГИЧЕСКАЯ СХЕМА ИСПЫТАНИЯ.

Для повышения эффективности испытания, его ускорения и удешевления необходимо разработать научно обоснованные методы, средства и методики, позволяющие преодолеть недостатки подхода к испытанию как к своего рода эвристике, недооценку его роли в обеспечении требуемого уровня качества ПП, подмену испытаний процедурами типа проверки работоспособности на контрольном примере и т. п. Эта цель может быть достигнута лишь путем разработки технологической схемы испытаний, предусматривающей;

знание назначения испытываемого ПС, условий его функционирования и требований к нему со стороны пользователей;

автоматизацию всех наиболее трудоемких процессов и прежде всего моделирование среды функционирования, включая искажающие воздействия;

ясное представление цели и последовательности испытания;

целенаправленность и неизбыточность испытания, исключающие или минимизирующие повторение однородных процедур при одних и тех же условиях функционирования испытываемого ПС;

систематический контроль за ходом, регулярное ведение протокола и журнала испытания;

четкое, последовательное определение и исполнение плана испытания;

четкое сопоставление имеющихся ресурсов с предполагаемым объемом испытания;

возможность обеспечения, а также объективной количественной оценки полноты и достоверности результатов испытания навсех этапах.

Любому виду испытаний должна предшествовать тщательная подготовка. В подготовку испытаний ПС входят следующие мероприятия:

* составление и согласование плана-графика проведения испытания;
* разработка, комплектование, испытание и паспортизация программно-технических средств, используемых при испытаниях;
* анализ пригодности испытательных средств, используемых во время предварительных испытаний, для проведения приемочных испытаний;
* анализ пригодности накопленных данных о качестве ПС для использования при окончательном определении значений показателей качества испытываемого ПС;
* проверка и согласование с представителем Заказчика конструкторской документации на ПС, предъявляемой при испытаниях;
* разработка, согласование и утверждение программ и методикиспытаний;
* аттестация специалистов на допуск к проведению испытаний;
* приемка испытываемого опытного образца ПС на носителе данных и документации;
* проведение мероприятий, направленных на обеспечение достоверности испытаний.

Особо следует подчеркнуть необходимость заблаговременной разработки и испытания всех программно-технических средств, которые будут использоваться при проведении испытаний. При этом следует иметь в виду, что уровень точности и надежности измерительной аппаратуры, используемой при испытаниях любого объекта, должен быть значительно выше соответствующих показателен испытываемого объекта. Поэтому реальные характеристики программно-технических испытательных средств необходимо установить заранее, а их приемлемость согласовывать между разработчиками, испытателями и заказчиками ПС. Пренебрежение этим правилом вызывает недоверие к результатам испытания и, как следствие, удлинение сроков испытания.

Сложность программно-технических испытательных средств, требования к их совершенству, а следовательно, и затраты ресурсов на их разработку прямо пропорционально зависят от соответствующих показателей испытываемых ПС. Объем испытательных программных средств, выраженный в машинных командах, может достигать объема испытываемых с их помощью программ. Поэтому разработка программно-технических средств, предназначенных для испытания особо сложной ПП, должна начинаться одновременно с разработкой опытных образцов продукции.

На основании изложенного можно определить следующие пять этапов испытания.

1. Обследование проектируемого ПС, анализ проектной документации.

2. Определение наиболее важных подсистем, функций и путей проектируемого ПС, подлежащих испытанию.

3. Анализ показателей качества ПС и методов определения их значений. Разработка программ и методик испытания.

4. Разработка (освоение) испытательных программно-технических средств, библиотек тестов и баз данных (если они требуются).

5. Непосредственное проведение испытаний, анализ результатов, принятие решения.

На рис. 16 изображена технологическая схема в виде этапов подготовки и проведения испытания и их связи с этапами разработки ПС.

Рис. 16. Технологическая схема испытания ПС.



В зависимости от специфики, условий применения, требований к качеству испытываемых ПС испытания могут проводиться либо путем тестирования, либо путем статистического моделирования среды функционирования, либо на основе натурных и смешанных экспериментов. Часто полезно использование всех этих методов. Значения некоторых показателей качества можно получить экспертным путем.

ПЛАНИРОВАНИЕ И ОЦЕНКА ЗАВЕРШЕННОСТИ ИСПЫТАНИЙ.

План проведения испытаний должен быть ориентирован на обеспечение всесторонней проверки ПС и максимальной (заданной) достоверности полученных результатов при использовании ограниченных ресурсов, выделенных на испытаниях. Принципиально возможны следующие подходы к решению этой задачи:

1) анализируют весь диапазон входных данных. На основе анализа заранее готовят такое множество комбинаций данных (тестовых наборов данных), которое охватывает наиболее характерные подмножества входных данных. Программу рассматривают как черный ящик. Испытания сводятся к последовательному вводу тестовых наборов данных и анализу получаемых результатов;

2) анализируют множество ситуаций, которые могут возникнуть при функционировании ПС. Выбирают наиболее характерные ситуации. Каждую из них выражают через тестовый набор входных данных. Далее сущность испытания и анализа результатов сводится к подходу 1);

3) с помощью графовой модели анализируют микроструктуру ПС. Выбирают множество путей, которое полностью покрывает граф-схему ПС, и такую последовательность тестовых наборов исходных данных, выполнение которой будет проходить по выделенным путям. Организация испытаний аналогична подходам 1) и2);

4) ПС испытывают в реальной среде функционирования;

5) ПС испытывают в статистически моделируемой среде функционирования, адекватной реальной среде.

Ни один из этих подходов не является универсальным. Каждый из них имеет свои преимущества и недостатки, которые в разной степени проявляются в зависимости от специфики испытываемого ПС. Например, подход 1) может оказаться предпочтительным, если диапазон входных данных обозрим, сравнительно легко анализируется и систематизируется, и неприемлемым — в противном случае. Наиболее достоверные результаты получаются при испытаниях в реальной среде функционирования. Но такие испытания редко удается осуществить. Поэтому на практике используют комбинации всех видов. Типичным примером такой комбинации может служить смешанный метод, когда среда функционирования ПС моделируется, а достоверность результатов проверяется путем сравнения с результатами, полученными при функционировании ПС в реальной среде.

Анализ показывает, что абсолютная проверка ПС ни при одном из рассмотренных подходов не осуществима. Поэтому при планировании испытаний необходимо предварительно анализировать структуры испытываемых программ и входных данных. В частности, следует устанавливать те пути граф-схемы программы, использование которых при преобразовании данных наиболее вероятно. Эта задача аналогична подходам 1) и 2). Для сложных программных комплексов она не имеет строго математического решения. Вместе с тем на практике нередко удается заранее установить наиболее вероятные ситуации, которые могут возникнуть в автоматизируемой системе, а следовательно, и наборы входных данных, описывающие эти ситуации.

Методика решения задачи планирования испытания включает в себя следующие этапы: нахождение всех путей реализации;

выделение минимального подмножества путей, обеспечивающих проверку всех участков программы; разработка тестов для проверки выделенных путей. Необходимо отметить, что в результате решения получают не одно подмножество путей, а некоторую совокупность таких подмножеств. Анализируя эти совокупности по критериям минимального времени реализации их на ЭВМ, выбора наиболее вероятных путей, отсутствия в этих совокупностях несовместимых путей (рассмотренным методам присущ этот недостаток), выбирают наиболее приемлемую совокупность. Для формирования входных данных тестирования для каждого выделенного пути реализации составляют специальные таблицы. В таблицах представляют только условные операторы, принадлежащие данному пути, и операторы, в которых вычисляются переменные управления. В результате анализа предписаний, удовлетворяющих условным операторам, вырабатывают входные данные тестирования.

Для установления потребности в машинном времени на проведение испытаний необходимо знать среднее значение абсолютной реактивности ПС. Эта характеристика должна быть задана в ТЗ. Если же она не задана, то можно принять где — минимальное значение абсолютной реактивности; — максимальное значение абсолютной реактивности.



Несмотря на то что проверка всех путей граф-схемы большой программы неосуществима, при планировании испытаний необходимо при заданных ресурсах обеспечить максимальную полноту проверки, особенно проверки модулей решения наиболее ответственных задач. Стремление избежать при этом неэффективного простого перебора приводит к задаче выбора минимального количества путей, покрывающих граф ПС. Под покрытием понимают включение всех дуг графа. Минимальное покрытие, с одной стороны, обеспечивает минимум тестов и контрольных просчетов, а, с другой стороны, гарантирует прохождение каждой дуги графа хотя бы по одному разу.

Рассмотренный метод планирования на этапе автономных статистических испытаний модулей ПИ позволяет значительно уменьшить материальные и временные затраты на испытание программ. Ориентация на тот или иной подход к испытаниям зависит от типа испытываемого ПС.

В общем случае при планировании и организации испытаний следует искать компромиссное решение, учитывающее два противоречивых требования: обеспечение максимальной достоверности обобщенной оценки качества ПИ и выполнение испытания в ограниченное время с использованием ограниченных ресурсов. Следует выделить три стадии испытания: подготовительную; непосредственные испытания; заключительную (подготовка отчетных материалов). Задачи этих стадий очевидны. Подробнее остановимся на задачах подготовительной стадии.

Эта стадия наиболее длительная и наиболее трудоемкая. Основными ее задачами являются: планирование испытаний;

разработка технологической схемы испытаний и испытательных средств; разработка программ и методик испытания; накопление предварительных статистических данных, характеризующих ПС.

Целенаправленность и четкость организации работ по накоплению статистических данных может существенно повысить достоверность оценки качества ПС, исключить дублированные (повторные) проверки и уменьшить сроки испытаний и затрачиваемые материальные ресурсы. Однако в некоторых случаях из-за плохой организации работы результаты тестирования на этапах отладки программ и предварительных испытаниях не регистрируются, поэтому не могут использоваться для окончательной. оценки качества программы.

Между выделенными стадиями испытания ПС имеются прямые и обратные связи, аналогичные связям между этапами жизненного цикла ПС. Это означает, что при выполнении работ заключительной стадии может быть выявлена необходимость возвращения к стадии непосредственных испытаний (или даже к подготовительной стадии) для уточнения отдельных характеристик.

Составлению плана проведения испытаний должен предшествовать анализ Т3 на разработку ПС, структурных и функциональных схем, режимов функционирования, зависимостей между модулями программы, планов-графиков разработки и отладки компонентов ПС, результатов контроля их качества на ранних стадиях разработки. В результате этого анализа необходимо разработать и обосновать общую стратегию испытания, а на ее основе—комплекс документов по организации испытаний, который должен содержать ответы на следующие вопросы: 1) задачи испытаний на каждой фазе, последовательность развития фаз; 2) используемые специальные испытательные средства; 3) количество необходимого машинного времени на каждой фазе испытаний; 4) конфигурация общего технического и программного обеспечения; 5) оцениваемые свойства, критерии оценки, способы их получения; 6) процедуры контроля хода испытания; 7) процедуры регистрации, сбора, обработки и обобщения результатов испытания; 8) условия (критерии) начала и завершения каждой фазы испытаний. По каждому из этих вопросов необходимо определить ответственных исполнителей, сроки выполнения работ, вид конечного результата.

В стандарте IEEE 829—1983 (США) большое внимание уделено документированию процесса испытания ПП. Перечень документов, которые разрабатываются и ведутся в соответствии с планом испытания, приведен в разделе “Документирование”,

Проанализировав содержание выделенных разделов плана испытания/тестирования, можно сделать вывод о целесообразности включения сведений, содержащихся в этих разделах, в программы и методики испытания ПС. Такое включение будет способствовать повышению информативности этих документов и упорядочению самого процесса испытаний.

На проведение испытаний ПП приходится затрачивать значительные трудовые и материальные ресурсы. Сроки проведения испытаний всегда ограничены. Поэтому перед испытателями всегда стоит задача поиска путей минимизации затрат материальных, трудовых и временных ресурсов для достижения цели испытания. Для реализации этой задачи необходимо установить критерии завершенности Испытаний, которые могут служить основой для принятия решения о завершении испытаний.

При оценке уровня завершенности испытаний ПС и достоверности полученных результатов часто возникают серьезные затруднения. Отметим следующие из них:

1) большинство ПС являются уникальными и либо не имеют аналогов для сравнения характеристик, либо имеют аналоги, характеристики которых неизвестны;

2) отсутствие общепринятых показателей, а также методов расчета требуемых и фактических значений приводит к тому, что в ТЗ на разработку ПС требования к характеристикам ПС либо фактически отсутствуют (в количественном выражении), либо не претендуют на полноту.

Рассмотрим пути решения проблемы оценки завершенности испытаний ПС. Но прежде всего обратим внимание на необходимость тщательного документирования процесса испытания. Такое документирование следует начать с момента приобретения ПС свойства работоспособности и вести его непрерывно до момента передачи ПС в промышленную эксплуатацию.

Опыт создания отечественных систем реального времени подтверждает необходимость ведения одного или двух журналов. В одном из них следует регистрировать все эксперименты с ПС, а в другом—обнаруженные ошибки (проблемы) и ход их устранения. Периодически составляют отчеты об испытаниях за определенный период времени. Для ведения журналов необходимо тщательно разработать инструкции, в которых установить общие правила заполнения журналов, в том числе единые правила присвоения регистрационных номеров ошибкам, индексации типов ошибок, классификации ошибок и т. п. В журналах следует предусмотреть отдельные разделы, в которых при необходимости будут даваться подробные комментарии к ошибкам и способы их устранения.

Не всякую ошибку можно быстро идентифицировать, поэтому в стандарте IEEE 829—1983 рекомендовано документировать в виде отчетов тест-инцидент все нестандартные события, происходящие во время испытания и требующие дополнительного анализа. Рекомендуется следующая структура этого отчета: идентификатор отчета тест-инцидент, аннотация, описание инцидента, влияние инцидента на последующий ход испытания. Последние два раздела являются основными. Описание инцидента должно включать следующие элементы: входные данные, ожидаемые и фактические результаты, отклонение от нормы, дата и время испытания, шаг процедуры испытания, среда функционирования, результаты попыток повторения условий эксперимента, испытатели, наблюдатели-регистраторы.

Регистрация экспериментов и ошибок (инцидентов), периодическая обработка данных и анализ результатов позволяют контролировать испытания и управлять этим процессом. Сама процедура регистрации может быть разной, важно лишь предотвратить потерю ценной информации при минимальных трудозатратах на сбор и обработку данных. Данное условие можно обеспечить только путем максимальной автоматизации всех процессов.

**Критерий интенсивности обнаружения ошибок.** Если считать, что во время одного эксперимента обнаруживается не более одной ошибки и каждая ошибка до начала следующего эксперимента устраняется, то можно предположить, что при благоприятном ходе отладки и испытания кривая зависимости: *N* = 1 — *п/К,* где *п —* количество обнаруженных и устраненных ошибок; *К. — .* количество экспериментов, будет асимптотически стремиться к единице (кривая 1 на рис. 17). Кривая *2* свидетельствует о неблагополучном ходе процесса.

Тогда в качестве критерия прекращения испытаний можно принять, например, следующее условие: *N >* 0,95 при обнаружении в последних двухстах экспериментах не более трех несущественных ошибок.

Идея выбора такого критерия основана на том, что частота обнаружения ошибок, выраженная отношением *п/К,* по мере увеличения количества экспериментов должна уменьшаться и к моменту завершения испытаний принять значение, близкое к нулю. Следует иметь в виду, что оценка уровня завершенности испытаний по этому показателю будет достоверной лишь в том случае, если каждый эксперимент проводится в новых условиях и испытатели стремятся обнаружить ошибки, а не доказать их отсутствие. Если же программу проверяют при одних и тех же или близких условиях, то довольно быстро получают кривую вида 1, которая не свидетельствует ни о полноте, ни о глубине проверки программ, ни об отсутствии в ней ошибок.

**Критерий заданного значения средней наработки на отказ** (критерий Дж. Д. Муса). Сделано два предположения. 1. Суммарное количество обнаруженных и устраненных дефектов в про

грамме (под дефектом понимается любая причина неудовлетворенности свойствами программы) описывается показательной функцией времени функционирования



- исходное количество дефектов в программе; - общее количество дефектов, которое может проявиться за время эксплуатации ПС; — средняя наработка на отказ в начале испытаний;



*С—*коэффициент сжатия тестов. Коэффициент *С1* тогда, когда абсолютная реактивность программы при прогоне тестов или статистических испытаниях отличается от абсолютной реактивности при работе программы в реальных условиях. Если, например, за один час испытаний моделируется управляемый процесс, происходящий в реальных условиях в течение десяти часов, то коэффициент сжатия *С* принимается равным 10.



Скорость обнаружения и устранения дефектов, измеряемая относительно времени функционирования программы, пропорциональна интенсивности отказов. Коэффициент пропорциональности *B=n/m* называется коэффициентом уменьшения дефектов.

Количество зарегистрированных отказов *т* зависит от суммарного времени функционирования программы следующим образом:



Значение средней наработки на отказ также зависит от суммарного времени функционирования:



Если в ходе испытания обнаруженные ошибки устраняются, то текущее значение средней наработки на отказ будет увеличиваться. Таким образом, в качестве критерия завершенности испытания можно принять достижение требуемого (заданного) значения средней наработки на отказ Tо. Тогда, определяя периодически текущее значение средней наработки на отказ по этой формуле , можно при планировании дальнейшего хода испытания рассчитать требуемое время для дальнейшего прогона программы по формуле



При планировании отладки и испытания ПО следует учитывать влияние следующих факторов: 1) скорости выявления дефектов; 2) скорости устранения дефектов; 3) удовлетворенности машинным временем. Первый фактор зависит от укомплектованности и квалификации испытателей, второй—от укомплектованности и квалификации группы программистов отладчиков, третий — от фондовооруженности (технической оснащенности) разрабатывающей (испытывающей) организации.

**На начальной стадии отладки программы интенсивность выявления дефектов высока. Программисты-отладчики перегружены работой, приходится даже прерывать тестовые прогоны, делать перерывы в испытаниях. На заключительной стадии интенсивность обнаружения дефектов низкая, но остро ощущается необходимость в машинном времени. Испытатели перегружены в подготовке все новых и новых тестовых исходных данных, в то время как у программистов-отладчиков работы может быть мало.**

СТЕНДЫ ОТЛАДКИ И ИСПЫТАНИЯ ПРОГРАММ.

Идея имитационного моделирования положена в основу создания комплексных имитационно-моделирующих испытательных стендов, используемых для отладки и испытания сложных систем управления в реальном масштабе времени.

***Комплексный имитационно-моделирующий испытательный стенд (КИМИС)*** представляет собой совокупность средств испытываемой системы и их моделей, модели внешней среды и программ обработки результатов моделирования, функционально объединенных на основе испытываемого программного комплекса. Комплексные имитационно-моделирующие испытательные стенды используются при полигонных испытаниях сложных систем.

Общая идея создания КИМИС основана на том, что для испытания (исследования) ПС, реализованного непосредственно на управляющей ЭВМ, необходимо моделировать управляемый процесс и имитировать поступление в ЭВМ информации об этом процессе. Испытываемое ПС безразлично к непосредственным источникам информации. Важно лишь, чтобы вся информация была распределена по реальным физическим каналам ЭВМ и временным тактовым интервалам, а также соответствовала заданному (ожидаемому) диапазону условий внешней среды. Сопряжение моделей с реальными средствами системы необходимо для оценки результатов моделирования путем их сравнения с реальными данными. Использование в составе КИМИС непосредственно самого ПС, а не его модели, позволяет получить более достоверные результаты при моделировании и избежать больших дополнительных трудозатрат на разработку модели ПС.

Для создания КИМИС, помимо основной ЭВМ, на которой реализуется испытываемое ПС, используют ЭВМ примерно такой же производительности для реализации комплекса моделей соответствующего назначения. Первую ЭВМ (ВС) обычно называют технологической, вторую—инструментальной. Инструментальная ЭВМ и программное обеспечение образуют КИМИС. Такие КИМИС являются кроссовой системой (КРОСС-КИМИС). Моделируемые (имитируемые) на инструментальной ЭВМ данные передаются в технологическую ЭВМ, где и обрабатываются как реальные данные. Программное обеспечение КИМИС может быть реализовано и на технологической ЭВМ (Резидент-КИМИС). Но такой вариант используется сравнительно редко из-за дефицита памяти и производительности в технологических (управляющих) ЭВМ.

Автоматизированный технологический комплекс (АТК) состоит из элементов следующих типов: управляемый технологический агрегат (УТА), автоматизированная система управления технологическим процессом (АСУ ТП), датчики информации (ДИ) о состоянии управляемого процесса. На вход АТК поступает объект обработки (00), на выход—результат обработки (РО). Если прекратить доступ информации в ЭВМ от реальных физических объектов АТК, а вместо нее вводить адекватную ин формацию, имитируемую по КИМИС на инструментальной ЭВМ , то процесс функционирования ПО АСУ ТП будет адекватен реальному. Оператор УТА в принципе может участвовать в обеих режимах.

Программное обеспечение КИМИС в общем случае состоит из следующих подсистем: моделирования, анализа результатов испытания, регистрации событий (документирования), планирования и управления и базы данных (рис. 20). В состав *подсистемы моделирования* входят: модель заявок на обработку (МЗ), модель объекта обработки (МОО); модели датчиков информации (МДИ); имитатор помех (ИП); модель управляемого технологического агрегата (МТА).

Модель заявок имитирует поток заявок на обработку (например, поток слябов) исходя из плановых и производственных соображений



В соответствии с заданным приоритетом или случайным образом выбирается 00, принимаемый на обслуживание, из совокупности 00, имитируемой МЗ, и его характеристики. Модели датчиков информации являются информационными моделями конкретных типов датчиков информации, используемых в системе управления АТК. Они имитируют выдачу текущих координат характеризующих состояние технологического процесса. Модель управляемого технологического агрегата (например, прокатного стана) имитирует управляемый технологический процесс (например, прокатки стали) с выдачей соответствующей информации об этом процессе. Имитатор помех в соответствии с заданными вероятностными характеристиками имитирует воздействие случайных факторов на элементы моделируемой системы и управляемый процесс. При этом используется датчик случайных чисел, позволяющий реализовать процедуру “розыгрыш”.

Таким образом, подсистема моделирования, имитируя технологический процесс в управляемом агрегате, обеспечивает воспроизведение потока входной информации в управляющую ЭВМ, адекватного этому потоку в реальных условиях эксплуатации АТК.

Имитируемый поток входной информации подается' на вход испытываемого ПО АСУ и инициирует его функционирование, результатом которого является поток выходной (управляющей) информации, выдаваемый на УТА или его модель. Образуется замкнутый контур управления, адекватный контуру управления в реальном ДТК.

Основными компонентами ***подсистемы анализа результатов испытаний*** являются: программа выборки результатов преобразования входных данных, программы формирования эталонных значений для анализа правильности результатов, программа сравнения фактических результатов с эталонными и оценки их приемлемости (правильности).

***Подсистема регистрации событий*** обеспечивает документирование хода испытаний и регистрацию всех тех характеристик, которые могут быть полезны как для определения значений показателей качества испытываемого ПС, так и для оценки эффективности и состояния самого процесса испытаний.

***Подсистема планирования и управления*** на основе анализа состояния испытаний, полученных результатов, проверенных путей граф-схемы испытываемого ПС и поступающих заданий от программистов-испытателей осуществляет планирование экспериментов и подготовку соответствующих исходных данных для подсистемы моделирования. На эту же подсистему возлагается координация действий (инициализация) всех элементов КИМИС.

Достоинства КИМИС очевидны. Его использование позволяет осуществлять комплексную стыковку объектов испытываемой системы и проверку принципов управления задолго до создания всех элементов системы (элемент системы, разработка которого не завершена, заменяется моделью). Применение моделирования позволяет разнообразить условия испытания и сэкономить материальные ресурсы. Комплексные испытательные моделирующие стенды можно использовать не только для испытания программ, но и для отработки взаимодействия всех элементов системы.

Сопряжение реальных средств испытываемой системы с их моделями позволяет разнообразить условия испытания и провести полунатурные эксперименты. Можно, например, проверить работу автоматизируемого технологического агрегата, моделируя поведение объекта обработки или, наоборот, промоделировать работу технологического агрегата при работе с реальным объектом обработки. Такие вариации позволяют, с одной стороны, проверять адекватность моделей своим оригиналам и тем самым

убеждаться в достоверности результатов статистических испытаний, а, с другой стороны, использовать КИМИС на самых ранних этапах разработки опытного образца ПС для выбора и апробации наилучших проектных решений.

**IV. СЕРТИФИКАЦИЯ ПРОГРАММНЫХ ПРОДУКТОВ.**

**СТАНДАРТИЗАЦИЯ СИСТЕМ КАЧЕСТВА.**

В начале 70-х годов многие специалисты пришли к выводу о необходимости широкого распространения индустриальных (инженерных) методов в области построения программ (см, § 1.1). Индустриальные методы базируются на строгой регламентации и автоматизации технологических процессов. Таким образом, стандартизация и в области построения программ стала жизненной необходимостью.

В рамках Единой Системы Программной Документации (ЕСПД) разработано и введено в действие около тридцати стандартов, упорядочивающих разработку программной документации. Многие виды стандартов для программной продукции еще не разработаны (общие технические требования, общие технические условия, технические условия на виды ПП, номенклатура показателей качества, методы выполнения отдельных видов работ в технологических процессах, порядок проведения этих работ и др.).

При разработке ПМК системы УК ПП приняты следующие исходные положения:

1) разработка ПП осуществляется в соответствии с действующими стандартами, техническими условиями, ТЗ или иными заменяющими его документами, содержащими требования к качеству ПП, установленные на основании анализа требований конкретного и (или) потенциального пользователя к потребительским свойств данного вида ПП;

2) качество ПП обеспечивается преимущественно в процессе его разработки; по завершению каждого этапа разработки проекта должен проводиться документированный, систематический и критический анализы результатов разработки;

3) за качество разрабатываемой ПП ответственность несет разработчик, поставляемой — поставщик;

4) руководство организации — разработчика несет ответственность за определение политики в области качества и за решения, касающиеся разработки, внедрения и ведения системы качества;

5) управление качеством ПП основывается прежде всего на стимулировании заинтересованности разработчиков и поставщиков в обеспечении высокого качества ПП, повышении профессионализма:

6) для обеспечения требуемого качества ПП управление качеством осуществляется на всех стадиях и этапах жизненного цикла ПП, начиная с самых ранних;

7) в разрабатывающей организации должны быть определены система качества (управляющие органы и лица, несущие ответственность за качество), а также политика в области качества ПП; ответственность и полномочия за каждый вид деятельности, влияющей на качество ПП; определение обязанностей и полномочий должно обеспечивать достижение поставленных целей на заданном уровне эффективности;

8) управление качеством ПП базируется на контроле качества в процессе разработки;

9) все формализуемые функции, процедуры и операции по управлению качеством в конечном счете должны быть переданы ЭВМ и реализованы на ней в виде инструментальных программ;

10) в идейном (концептуальном) плане инструментальные программы и методики, входящие в состав ПМК, должны представлять единое целое, согласующееся с принятой технологией программирования и являющееся составной частью этой технологии;

11) в составе ПМК подсистемы У К ПП можно выделить базовую (условно постоянную) и переменную части. Базовая часть-ПМК разрабатывается как типовое проектное решение с использованием принципов модульной структуры и может быть использована в различных организациях, независимо от ведомственной принадлежности и собственной специфики. Переменная часть-ПМК учитывает специфику разрабатывающей организации, структуры и задач подсистемы УК ПП. Она создается в конкретной организации путем настройки базовой части ПМК и разработки новых, недостающих частей подсистемы УК ПП;

12) все компоненты базовой части ПМК должны обладать свойствами автономности (независимости) разработки, настройки и применения. Однако наибольший эффект должен достигаться от комплексного использования всех компонентов ПМК.

Основными методами стандартизации УК ПП в разрабатывающей организации являются: систематизация и классификация: типизация и унификация; регламентирование.

***Систематизация и классификация*** направлены на упорядочение элементов управления (ГКК, СКК и др.), установление их прав и обязанностей, а также взаимодействия между ними.

***Типизация и унификация*** направлены на выявление и формирование сходных компонентов программ и программных комплексов по профилю организации, па создание библиотек унифицированных компонентов, средств генерации программ из этих компонентов, интерфейсных соглашении.

***Регламентирование*** направлено на упорядочение организационных и технологических процедур по обеспечению требуемого уровня качества на всех стадиях жизненного цикла ПП.

В США, например, в середине 80-х годов введены в действие следующие стандарты: ANSI/IEEE “Спецификация требований к ПО” (Guide to Software Requirements Specifications); “Планирование управления конфигурацией ПО” (Software Configuration Management Plans); “Документирование тестов ПО” (Software Test Documentation); “Планирование уровня качества ПО” (Software Quality Assurance Plan?). В качестве проектов апробируются и другие стандарты, в том числе “Справочник гарантии качества”, “Классификация отказов, сбоев и ошибок ПО”.

При организации управления качеством ПП многие полезные рекомендации можно заимствовать из соответствующих стандартов по управлению качеством промышленной продукции.

В 1987 г. утверждено пять международных стандартов ISO, устанавливающих требования к системам обеспечения качества продукции на предприятиях: “Стандарты по управлению качеством и обеспечению качества. Руководство для выбора и применения” (ISO 9000); “Система качества. Модели обеспечения качества при проектировании, разработке, производстве, монтаже и обслуживании” (ISO 900S); “Система качества. Модели обеспечения качества при производстве и монтаже” (ISO 9002); “Система качества. Модели обеспечения качества в процессе контроля и испытания готовой продукции” (ISO 9003); “Управление качеством и элементы системы качества. Основные направления” (ISO 9004).

КЛАССИФИКАЦИЯ ПОКАЗАТЕЛЕЙ КАЧЕСТВА

Под ***показателем качества*** программной продукции в соответствии с ГОСТ 15467—79 следует понимать количественную характеристику одного или нескольких свойств продукции, составляющих ее качество, рассматриваемую применительно к определенным условиям ее создания и эксплуатации. Свойство продукции — это объективная особенность, которая может проявиться при создании или эксплуатации продукции. В определении понятия “Показатель качества” слова “Количественная характеристика” не следует понимать в буквальном смысле. При определении значений показателей качества успешно могут применяться и нечисловые характеристики, хотя в общем случае наличие строго количественных, числовых характеристик предпочтительней.

Показатели качества программной продукции в зависимости от характера решаемых задач по оценке качества продукции можно классифицировать по следующим признакам: характеризуемые свойства; способ выражения; количество характеризуемых свойств; место применения в процедуре оценки; стадии определения значений показателей.

***По способу выражения*** различают показатели, выраженные в натуральных единицах, и показатели, выраженные в стоимостных единицах. В качестве натуральных единиц обычно используют единицы физических величин (килограммы, метры, секунды и т. п.), а также баллы и безразмерные единицы. ПС являются информационными объектами. Какими-либо собственными физическими свойствами они не обладают, поэтому единицы физических величин в традиционном виде при определении значений показателей качества ПС почти не применяются, за исключением единиц времени. Но как составной элемент системы обработки данных ПС вносит определенную долю погрешности в точность выходных результатов. Эта погрешность может измеряться в единицах преобразуемых физических величин. Вместе с тем в программировании широко используют такие натуральные единицы, как бит, байт, условная машинная команда, строка текста и т. п. Стоимостные единицы применяют при определении значений экономических показателей качества программной продукции.

***По количеству характеризуемых свойств*** различают единичные и комплексные показатели. Единичные показатели качества характеризуют одно из свойств ПС, комплексный—несколько. Комплексные показатели могут быть групповыми, обобщенными или интегральными.

В зависимости ***от места применения в процедуре оценки*** уровня качества ПС различают базовые и относительные показатели. Базовым значением показателя качества продукции называют значение показателя, принятое за основу при сравнительной оценке качества продукции. Относительное значение показателя качества продукции представляет собой отношение фактического значения показателя качества оцениваемой продукции к базовому значению этого показателя.

***По стадии определения значений показателей качества*** различают прогнозируемые, проектные, производственные и эксплуатационные показатели. Прогнозируемыми показателями оперируют на стадиях выполнения научно-исследовательских работ и составления ТЗ на разработку ПС, т. е. на тех стадиях, когда нет еще ни детального проекта ПС, ни, тем более, самого ПС. Значения прогнозируемых показателей в основном определяют на основе интуиции и опыта аналогичных разработок, поэтому эти показатели носят субъективный характер.

Значения проектных показателей определяют на основе анализа проектов ПС (эскизного, технического, рабочего), а также путем испытания опытного образца *ПС.* Эти показатели носят более объективный характер. Степень их достоверности зависит от эффективности используемых инструментальных средств анализа и испытания.

Производственные показатели мало отличаются от проектных, особенно если изготовление ПС сводится к простому копированию. Если же копированию предшествуют операции сборки или генерации ПС, то производственные показатели качества таких ПС могут существенно отличаться от проектных.

Значения эксплуатационных показателей определяют по результатам промышленной эксплуатации ПС. При соблюдении определенных правил сбора и обработки данных о качестве ПС в процессе эксплуатации эксплуатационные показатели дают наиболее объективную и достоверную оценку. Только по этим показателям можно произвести действительную оценку научно-технического уровня и качества ПС.

ВЫБОР НОМЕНКЛАТУРЫ ПОКАЗАТЕЛЕЙ КАЧЕСТВА

Выбор номенклатуры показателей качества программной продукции заключается в установлении перечня наименований характеристик свойств продукции, определяющих качество данного вида продукции и обеспечивающих возможность полной и достоверной оценки ее уровня качества. Порядок выбора номенклатуры показателей качества программной продукции должен устанавливаться с учетом специфики этой продукции. Выбор номенклатуры показателей качества конкретного ПС зависит от вида (группы) ПС, цели применения и стадии определения показателей.

Для каждого вида (группы), а иногда и конкретного ПС устанавливают свою номенклатуру показателей качества, учитывающую специфику назначения п условий применения. Номенклатура показателей качества для каждого подкласса, группы и вида ПС оформляется в виде таблиц применяемости показателей качества. Помимо перечня рекомендуемых и обязательных показателей качества для данного подкласса (вида, группы) ПС, в таблицах применяемости следует указывать и коэффициенты (параметры) весомости (значимости) каждого из показателей. Определение коэффициентов весомости показателей качества — наиболее существенная и трудная задача выбора номенклатуры показателей качества. В принципе при решении этой задачи можно использовать либо метод стоимостно-регрессионных зависимостей, либо метод предельных номинальных значений. Но их использование затруднено из-за отсутствия необходимых исходных данных. Поэтому на практике наиболее распространен экспертный метод определения коэффициентов весомости.

Таблицы применяемости являются руководящим или справочным материалом для выбора рабочей номенклатуры показателей качества конкретного ПС. Рабочая номенклатура ПС устанавливается с учетом назначения и условий использования ПС; результатов анализа требований пользователя (заказчика), поставленных задач управления качеством; состава, структуры и специфики характеризуемых свойств.

***Цели применения*** номенклатуры показателен качества устанавливают в соответствии с задачами управления качеством программной продукции. Такими целями, в частности, могут быть следующие: составление технического задания па разработку ПС; составление технических условии на ПС; заполнение карты технического уровня; установление контролируемых показателей при проектировании ПС; установление контролируемых показателей при опытной эксплуатации ПС; аттестация ПС по категориям качества.

***Стадии определения*** значении показателей качества соответствуют стадиям жизненного цикла ПС.

При выделении свойств и соответствующих показателей качества ПС необходимо руководствоваться следующими основными принципами :

* выделение групп свойств должно производиться по четко определенным признакам;
* свойства, входящие в одну группу, должны, как правило, взаимно исключать друг друга и быть независимыми. Если свойства зависят друг от друга, то в методиках определения значении показателей качества должны быть даны четкие указания по исключению многократного (неоднократного) влияния одного и того же свойства на обобщенную оценку качества ПС;
* всякая исходная номенклатура показателей должна быть открытой, т. е. должна допускать возможность внесения мне исключения из нее отдельных элементов. Это требование обусловлено, с одной стороны, недостаточным опытом оценки качества программной продукции, а с другой,—большим разнообразием ПС и условий их применения;
* для каждого из выделенных свойств должна существовать возможность выражения их в шкалах “лучше — хуже”, “больше — меньше”;
* в группу следует включать свойства, необходимые и достаточные для определения соответствующего сложного свойства;
* формулировка свойств должна быть однозначной;
* совокупность свойств, характеризующих качество оцениваемого ПС, должна быть упорядочена по определенному правилу в виде многоуровневой иерархической структуры — дерева свойств;
* дерево свойств должно отражать все основные особенности использования н функционирования ПС;
* выбранные показатели качества должны быть скоррелнрованны с соответствующими свойствами ПС. Это значит, что между каждым из выделенных свойств и характеризующими его показателями должно быть установлено однозначное соответствие. Установление такого соответствия позволяет вместо дерева свойств использовать дерево показателей качества программной продукции;
* показатели качества, характеризующие свойства ПС, должны способствовать обеспечению соответствия качества ПС требованиям со стороны их пользователей и учитывать современные достижения науки и техники. Для выполнения этого принципа часто необходимо проводить специальные исследования, так как в общем случае между показателями качества могут возникать серьезные противоречия, а улучшение одного показателя может привести к ухудшению другого.

Для проверки работоспособности выбранной системы показателей качества необходимо устанавливать степень корреляции каждого рассматриваемого показателя с качеством ПС, полезность показателя, возможность количественного представление и автоматической оценки показателя. В частности, оценку полезности каждого из выбранных показателей для конкретных ПС рекомендуется производить по следующей шкале:

*5—*крайне важно, чтобы данный показатель имел высокое значение;

*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*—при низких значениях данного показателя ощутимых потерь нет,

Около 50 % частных показателей можно определить автоматически с помощью ЭВМ, 25 % —с помощью компаратора. Таким образом, оценка около 75 % показателей может быть формализована. Оценка 20 % показателей может быть произведена только квалифицированным специалистом. Большинство показателей устанавливают путем статического анализа программ и лишь около 5 % — в процессе динамических испытаний (Данные соответствуют положению в этой области в 80-е годы).

Следует иметь в виду, что оценка качества, а следовательно, к выбор показателей качества сложных многофункциональных программных комплексов типа операционных систем, систем управления базами данных, пакетов прикладных программ и так далее имеет свои особенности. Каждая функция таких ПС реализуется программным путем, задающим определенный технологический процесс преобразования входных данных в выходные. Известны цель этого процесса и потребность в нем, Для того чтобы удовлетворить эту потребность, ПС должна обладать определенными свойствами. Причем свойства ПС, удовлетворяющие потребности в одной функции, могут существенно отличаться от свойств ПС, необходимых для реализации другой функции. Поэтому степень удовлетворения потребности в выполнении каждой из функций ПС в общем случае характеризуется своими показателями или, по крайней мере, параметрами весомости показателей. Возникает необходимость выбора показателей и определения их весомости для оценки качества (эффективности) реализации каждой из основных функций ПС. Попытка выбора единой номенклатуры показателей качества оказывается, как правило, безрезультатной. В этом можно легко убедиться на примере оценки качества операционных систем (ОС) ЭВМ. На ОС ЭВМ возлагаются следующие функции: управление данными, заданиями, вводом-выводом; обслуживание библиотек пользователей;

трансляция и редактирование программ; протоколирование состояний и событий; перезапись и сортировка информации и др. Очевидно, что требования, например, к трансляторам существенно отличаются от требований к ПС протоколирования событий как по своему перечню, так и по весомости каждого из показателей. В свою очередь различие требований обусловливает необходимость использования различных показателей качества, характеризующих потребительские свойства программ, реализующих эти функции.

ГРУППЫ ПОКАЗАТЕЛЕЙ КАЧЕСТВА

Любому классу продукции присущи определенные свойства, характерные для данного класса. В свою очередь каждый подкласс, группа, вид этой продукции имеет частные свойства, отличающие изделия одной классификационной группировки от другой. Рассмотрим формирование номенклатуры показателей качества, характеризующей общие свойства класса программной продукции. Эта номенклатура может быть использована в качестве исходной при выборе рабочей номенклатуры показателей качества любого конкретного ПС.

Номенклатуры показателей качества всегда имеют иерархическую структуру. Их формирование начинается с выделения групп верхнего уровня иерархии, а затем номенклатуры детализируются вплоть до получения единичных показателей.

Выделение групп показателей качества является важной и сложной задачей формирования номенклатуры показателей качества. Неудачное комплектование групп может привести к усложнению взаимосвязей между группами и отдельными показателями, а также сделать номенклатуру показателей качества малоконструктивной.

Известно, что для оценки качества промышленной продукции используют следующие группы показателей: назначения; экономичного использования сырья, материалов, топлива, энергии; надежности; эргономичности; эстетичности; технологичности; патентно-правовые; унификации и стандартизации; экологичности; безопасности.

Все эти показатели можно использовать и при оценке качества ПП. По в силу особенностей ПП некоторые группы показателей при оценке ее качества применять нецелесообразно (неконструктивно). К таким показателям относят показатели экологичности, безопасности.

***Экологические показатели и показатели безопасности*** нехарактерны для ПП, так как программные изделия непосредственно не могут оказывать вредных воздействий ни на окружающую среду, ни на здоровье человека. В принципе, такие воздействия возможны в тех случаях, когда ПИ используют в качестве элементов управляющих объектов, например в АСУ. В этом случае вырабатываемые ЭВМ по определенному алгоритму управляющие воздействия могут вызвать и неблагоприятные экологические последствия, и быть опасными для человека. Но это уже косвенное воздействие через управляющие органы и исполнительные механизмы автоматизированных технологических комплексов (АТК). Они учитываются как соответствующие показатели AT К.

***Патентно-правовые показатели*** программной продукции не могут быть использованы до тех пор, пока вопросы патентно-правовой защиты этой продукции не будут решены в законодательном (юридическом) плане.

Относительно ***надежности*** программной продукции существует много противоречивых мнений. Вместе с тем большинство специалистов единодушны в мнении о том, что природа надежности программных и технических средств различна. Для программной продукции малопродуктивными являются такие показатели надежности, как долговечность, сохраняемость, ремонтопригодность. Источниками низкой надежности ПС в основном являются ошибки в программах, внесенные на стадии проектирования и невыявленные при отладке и испытаниях. Заслуживает внимания мнение американского специалиста Фокса Д., который считает, что использование термина “надежность программного обеспечения” наносит вред, так как способствует неправильному пониманию природы программного обеспечения . Вместе с тем следует учитывать тот факт, что при анализе некоторых свойств ПП, проявляющихся при ее функционировании, приходится пользоваться категориями надежности (работоспособность, отказ, сбой, восстановление и др.). Поэтому в номенклатуре показателей качества ПП признано целесообразным выделять в отдельную группу показатели, характеризующие свойства ПП, близкие по своим внешним проявлениям показателям надежности аппаратуры. Эта группа названа показателями надежности функционирования.

Таким образом, в базовой номенклатуре показателей качества ПП на верхнем уровне выделяем следующие показатели: назначения, надежности функционирования, эргономичности, технологичности, унификации и стандартизации. Качество ПП в основном формируется в процессе создания продукции и в значительной мере зависит от эффективности структурных (конструктивных) решений. Поэтому на этом же уровне в отдельную группу выделим структурные показатели.

Показатели назначения, надежности функционирования, эргономичности и технологичности характеризуют свойства ПП, проявляющиеся в процессе ее использования (эксплуатации). По этому признаку их можно считать ***эксплуатационными****.* Структурные показатели и показатели унификации и стандартизации характеризуют свойства структуры (конструкции) ПС, их можно объединить в одну группу ***конструктивных показателей****.* По отношению к группе эксплуатационных показателей эта группа носит вспомогательный характер. Достижение определенного уровня значений этих показателей не может служить самоцелью, это лишь средство (путь) обеспечения требуемых значений одного или нескольких показателей, относящихся к основной группе — группе эксплуатационных показателей.