ИНТУИТ
НАЦИОНАЛЬНЫЙ ОТКРЫТЫЙ УНИВЕРСИТЕТ

Национальный Открытый Университет "ИНТУИТ": www.intuit.ru
Екатерина Лаврищева, Владимир Петрухин
Лекция 8. Методы проверки и тестирования программ и систем

В фундаментальную концепцию проектирования ПС входят базовые положения, стратегии, методы, которые применяются на процессах ЖЦ и обеспечивают тестирование (верификацию) на множестве тестовых наборов данных. К методам проектирования ПС относятся структурные, объектно-ориентированные и др. Их основу составляют теоретические, инструментальные и прикладные средства, которые влияют на процесс тестирования.

Теоретические средства определяют процесс программирования и тестирования программного продукта. К ним относятся методы верификации и доказательства правильности *спецификации программ* (см. "Формальные спецификации, доказательство и верификация программ"), метрики измерения (Холстеда, цикломатичная сложность Маккейба и др.) в качестве отдельных характеристик как формализованных элементов теории определения правильности и гарантии свойств конечного *ПО*. Например, концепция " *чистая комната* " базируется на формализмах доказательства и изучения свойств процессов кодирования и тестирования программ. Что касается тестирования как процесса, то это проверка правильности работы программы *по* заданным описаниям тестов и покрытия данными соответствующих критериев программы [7.1–7.5].

Инструментальные средства – это способы поддержки кодирования и тестирования (компиляторы, генераторы программ, отладчики и др.), а также организационные средства планирования и *отбора тестов* для программ, которые обеспечивают обработку текста на ЯП и подготовку для них соответствующих тестов.

При проверке правильности программ и систем рассматриваются процессы верификации, валидации и тестирования ПС, которые регламентированы в стандарте ISO/IEC 12207 [7.7] жизненного *цикла ПО*. В некоторой зарубежной литературе процессы верификации и тестирования отождествляются. С теоретической точки зрения *верификация* была рассмотрена в "Формальные спецификации, доказательство и верификация программ" , здесь же будут определены задачи и действия, соответствующих процессов ЖЦ.

Тестирование – это процесс обнаружения ошибок в *ПО* путем исполнения выходного кода ПС на тестовых данных, сбора рабочих характеристик в динамике выполнения в конкретной операционной среде, выявления различных ошибок, дефектов, отказов и изъянов, вызванных нерегулярными и аномальными ситуациями или аварийным прекращением работы *ПО*.Важное *место* в проведении верификации и тестирования занимают организационные аспекты – *деятельность* группы специалистов, осуществляющих планирование этих процессов, подготовку тестовых данных и наблюдение за тестированием.

7.1. Процессы ЖЦ верификация и валидация программ

Верификация и *валидация*, как методы, обеспечивают соответственно проверку и *анализ* правильности выполнения заданных функций и соответствия *ПО* требованиям заказчика, а также заданным спецификациям. Они представлены в стандартах [7.7–7.8] как самостоятельные процессы ЖЦ и используются, начиная от этапа анализа требований и кончая проверкой правильности функционирования программного кода на заключительном этапе, а именно, тестировании.

Для этих процессов определены цели, задачи и действия *по* проверке правильности создаваемого продукта (рабочие, промежуточные продукты) на этапах ЖЦ. Рассмотрим их трактовку в стандартном представлении.

Процесс верификации. Цель процесса – убедиться, что каждый *программный продукт* (и/или сервис) проекта отражает согласованные требования к их реализации. Этот процесс основывается:

- на стратегии и критериях верификации применительно ко всем рабочим программным продуктам;
- на выполнении действий стандарта по верификации;
- на *устранении недостатков*, обнаруженных в программных (рабочих и промежуточных) продуктах;
- на согласовании результатов верификации с заказчиком.

Процесс верификации может проводиться исполнителем программы или другим сотрудником той же организации, или сотрудником другой организации, например заказчиком. Этот процесс включает в себя действия *по* его внедрению и выполнению.

Внедрение процесса заключается в определении критических элементов (процессов и программных продуктов), которые должны подвергаться верификации, в выборе исполнителя верификации, инструментальных средств поддержки процесса верификации, в составлении плана верификации и его утверждении. В процессе верификации выполняются задачи проверки условий: контракта, процесса, требований, интеграции, проекта, кода и документации.При верификации согласно плану и требований заказчика проверяется правильность выполнения функций системы, интерфейсов и взаимосвязей компонентов, а также доступа к данным и к средствам защиты.

Процесс валидации. Цель процесса – убедиться, что специфические требования для программного продукта выполнены, и осуществляется это с помощью:

разработанной стратегии и критериев валидации для всех *рабочих продуктов*;

оговоренных действий по проведению валидации;

демонстрации соответствия разработанных программных продуктов требованиям заказчика и правилам их использования;

согласования с заказчиком полученных результатов валидации.

Процесс валидации может проводиться самим исполнителем или другим лицом, например, заказчиком, осуществляющим действия *по* внедрению и проведению этого процесса *по* плану, в котором отражены элементы и задачи проверки. При этом используются методы, инструментальные средства и процедуры выполнения задач процесса для установления соответствия тестовых требований и особенностей использования программных продуктов проекта.

На других процессах ЖЦ выполняются дополнительные действия:

проверка и контроль проектных решений с помощью методик и процедур просмотра хода разработки;

обращение к *CASE-системам* [7.10], которые содержат процедуры проверки требований к продукту;

просмотры и инспекции промежуточных результатов на соответствие их требованиям для подтверждения того, что ПО имеет корректную реализацию требований и удовлетворяет условиям выполнения системы.

Таким образом, основные задачи процессов верификации и валидации состоят в том, чтобы *проверить и подтвердить*, что конечный *программный продукт* отвечает назначению и удовлетворяет требованиям заказчика. Эти процессы взаимосвязаны и определяются, как правило, одним общим термином "*верификация* и *валидация*" или "*Verification and Validation*" (V&V) [7.7].

V&V основаны на планировании их как процессов, так и проверки для наиболее критичных элементов проекта: *компонент*, интерфейсов (программных, технических и информационных), взаимодействий объектов (протоколов и сообщений), передач данных между компонентами и их защиты, а также оставленных тестов и *тестовых процедур*.

После проверки отдельных компонентов системы проводятся их *интеграция* и повторная *верификация* и *валидация* интегрированной системы, создается комплект документации, отображающий правильность проверки формирования требований, результатов инспекций и тестирования.

7.2. Тестирование программ

Тестирование можно рассматривать, как процесс семантической отладки (проверки) программы, заключающийся в исполнении последовательности различных наборов контрольных тестов, для которых заранее известен результат. Т.е. тестирование предполагает выполнение программы и получение конкретных результатов выполнения тестов [7.1-7.5, 7.11, 7.12].

Тесты подбираются так, чтобы они охватывали как можно больше типов ситуаций алгоритма программы. Менее жесткое требование - выполнение хотя бы один раз каждой ветви программы.

Исторически первым видом тестирования была *отладка*.

Отладка - это проверка описания программного объекта на ЯП с целью обнаружения в нем ошибок и последующее их устранение. Ошибки обнаруживаются компиляторами при их синтаксическом контроле. После этого проводится *верификация по* проверке правильности кода и *валидация по* проверке соответствия продукта заданным требованиям.

Целью тестирования - проверка работы реализованных функций в соответствии с их спецификацией. На основе внешних спецификаций функций и проектной информации на процессах ЖЦ создаются функциональные тесты, с помощью которых проводится тестирование с учетом требований, сформулированных на этапе анализа *предметной области*. Методы *функционального тестирования* подразделяются на статические и динамические.

7.2.1. Статические методы тестирования

Статические методы используются при проведении инспекций и рассмотрении спецификаций компонентов без их выполнения.Техника статического анализа заключается в методическом просмотре (или обзоре) и анализе структуры программ, а также в доказательстве их правильности. Статический анализ направлен на анализ документов, разработанных на всех этапах ЖЦ и заключается в инспекции исходного кода и сквозного контроля программы.

Инспекция ПО - это статическая проверка соответствия программы заданным спецификациями, проводится путем анализа различных представлений результатов проектирования (документации, требований, спецификаций, схем или исходного кода программ) на процессах ЖЦ. Просмотры и инспекции результатов проектирования и соответствия их требованиям заказчика обеспечивают более высокое качество создаваемых ПС.

При инспекции программ рассматриваются документы рабочего проектирования на этапах ЖЦ совместно с независимыми экспертами и участниками разработки ПС. На начальном этапе проектирования инспекция

предполагает проверку полноты, целостности, однозначности, непротиворечивости и совместимости документов с исходными требованиями к программной системе. На этапе реализации системы под *инспекцией* понимается анализ текстов программ на соблюдение требований стандартов и принятых руководящих документов технологии программирования.

Эффективность такой проверки заключается в том, что привлекаемые эксперты пытаются взглянуть на проблему "со стороны" и подвергают ее всестороннему критическому анализу.

Эти приемы позволяют на более ранних этапах проектирования обнаружить ошибки или дефекты путем многократного просмотра исходных кодов. Символьное тестирование применяется для проверки отдельных участков программы на входных символьных значениях.

Кроме того, разрабатывается множество новых способов автоматизации символьного выполнения программ. Например, автоматизированное средство статического контроля для языков ориентированной разработки, инструменты автоматизации *доказательства корректности* и автоматизированный аппарат сетей Петри.

7.2.2. Динамические методы тестирования

Динамические методы тестирования используются в процессе выполнения программ. Они базируются на графе, связывающем причины ошибок с ожидаемыми реакциями на эти ошибки. В процессе тестирования накапливается информация об ошибках, которая используется при оценке надежности и качества ПС.

Динамическое тестирование ориентировано на проверку корректности ПС на множестве тестов, прогоняемых по ПС, в целях проверки и сбора данных на этапах ЖЦ и проведения измерения отдельных показателей (число отказов, сбоев) тестирования для оценки характеристик качества, указанных в требованиях, посредством выполнения системы на ЭВМ. Тестирование основывается на систематических, статистических, (вероятностных) и имитационных методах.

Дадим краткую их характеристику.

Систематические методы тестирования делятся на методы, в которых программы рассматриваются как "черный ящик" (используется информация о решаемой задаче), и методы, в которых программа рассматривается как "белый ящик" (используется структура программы). Этот вид называют тестированием с управлением по данным или управлением по входувыходу. Цель – выяснение обстоятельств, при которых поведение программы не соответствует ее спецификации. При этом количество обнаруженных ошибок в программе является критерием качества входного тестирования.

Цель динамического тестирования программ по принципу "черного ящика" – выявление одним тестом максимального числа ошибок с использованием небольшого подмножества возможных входных данных.

Методы "черного ящика" обеспечивают:

- эквивалентное разбиение;
- анализ граничных значений;
- применение функциональных диаграмм, которые в соединении с реверсивным анализом дают достаточно полную информацию о функционировании тестируемой программы.

Эквивалентное разбиение состоит в разбиении входной области данных программы на конечное число классов эквивалентности так, чтобы каждый тест, являющийся представителем некоторого класса, был эквивалентен любому другому тесту этого класса.

Классы эквивалентности выделяются путем перебора входных условий и разбиения их на две или более групп. При этом различают два типа классов эквивалентности: правильные, задающие входные данные для программы, и неправильные, основанные на задании ошибочных входных значений.Разработка тестов методом эквивалентного разбиения осуществляется в два этапа: выделение классов эквивалентности и построение тестов. При построении тестов, основанных на выборе входных данных, проводится символическое выполнение программы.

Итак, методы тестирования по принципу "черного ящика" используются для тестирования функций, реализованных в программе, путем проверки несоответствия между реальным поведением функций и ожидаемым поведением с учетом спецификаций требований. Во время подготовки к этому тестированию строятся таблицы условий, причинно-следственные графы и области разбивки. Кроме того, подготавливаются тестовые наборы, учитывающие параметры и условия среды, которые влияют на поведение функций. Для каждого условия определяется множество значений и ограничений предикатов, которые тестируются.

Метод "белого ящика" позволяет исследовать внутреннюю структуру программы, причем обнаружение всех ошибок в программе является критерием исчерпывающего тестирования маршрутов потоков (графа) передач управления, среди которых рассматриваются:

(а) критерий покрытия операторов – набор тестов в совокупности должен обеспечить прохождение каждого оператора не менееодного раза;

(б) критерий тестирования ветвей (известный как покрытие решений или покрытие переходов) – набор тестов в совокупности должен обеспечить прохождение каждой ветви и выхода, по крайней мере, один раз.

Критерий (б) соответствует простому структурному тесту и наиболее распространен на практике. Для удовлетворения этого критерия необходимо построить систему путей, содержащую все ветви программы. Нахождение такого оптимального покрытия в некоторых случаях осуществляется просто, а в других является более сложной задачей.

Тестирование по принципу "белого ящика" ориентировано на проверку прохождения всех путей программ посредством применения путевого и имитационного тестирования.

Путевое тестирование применяется на уровне модулей и графовой *модели программы* путем выбора тестовых ситуаций, подготовки данных и включает тестирование следующих элементов:

операторов, которые должны быть выполнены хотя бы один раз, без учета ошибок, которые могут остаться в программе иззабольшого количества логических путей и необходимости прохождения подмножеств этих путей;

путей по заданному графу потоков управления для выявления разных маршрутов передачи управления с помощью путевых предикатов, для вычисления которого создается набор тестовых данных, гарантирующих прохождение всех путей. Однако все пути протестировать бывает невозможно, поэтому остаются не выявленные ошибки, которые могут проявиться в процессе эксплуатации;

блоков, разделяющих программы на отдельные частиблоки, которые выполняются один раз или многократно при нахождении путей в программе, включающих совокупность блоков реализации одной функции либо нахождения входного множества данных, которое будет использоваться для выполнения указанного пути.

"Белый ящик" базируется на структуре программы, в случае "черного ящика", о структуре программы ничего неизвестно. Для выполнения тестирования с помощью этих "ящиков" известными считаются выполняемые функции, входы (входные данные) и выходы (выходные данные), а также логика обработки, представленные в документации.

7.2.3. Функциональное тестирование

Цель *функционального тестирования* – обнаружение несоответствий между реальным поведением реализованных функций и ожидаемым поведением в соответствии со спецификацией и исходными требованиями. Функциональные тесты должны охватывать все реализованные функции с учетом наиболее вероятных типов ошибок. Тестовые сценарии, объединяющие отдельные тесты, ориентированы на проверку качества решения функциональных задач.

Функциональные тесты создаются по внешним спецификациям функций, проектной информации и по тексту на ЯП, относятся к функциональным его характеристикам и применяются на этапе комплексного тестирования и испытаний для определения полноты реализации функциональных задач и их соответствия исходным требованиям.

В задачи *функционального тестирования* входят:

идентификация множества функциональных требований;

идентификация внешних функций и построение последовательностей функций в соответствии с их использованием в ПС;- идентификация множества входных данных каждой функции и определение областей их изменения;

построение тестовых наборов и сценариев тестирования функций;

выявление и представление всех функциональных требований с помощью тестовых наборов и проведение тестирования ошибок в программе и при взаимодействии со средой.

Тесты, создаваемые по проектной информации, связаны со структурами данных, алгоритмами, интерфейсами между отдельными компонентами и применяются для тестирования компонентов и их интерфейсов. Основная цель – обеспечение полноты и согласованности реализованных функций и интерфейсов между ними.

Комбинированный метод "черного ящика" и "прозрачного ящика" основан на разбиении входной области функции на подобласти обнаружения ошибок. Подобласть содержит однородные элементы, которые все обрабатываются корректно либо некорректно. Для тестирования подобласти производится выполнение программы на одном из элементов этой области.

Предпосылки *функционального тестирования*:

корректное оформление требований и ограничений к качеству ПО;

корректное описание модели функционирования ПО в среде эксплуатации у заказчика;

адекватность модели ПО заданному классу.

7.3. Инфраструктура процесса тестирования ПС

Под инфраструктурой процесса тестирования понимается:

- выделение объектов тестирования;
- проведение классификации ошибок для рассматриваемого класса тестируемых программ;
- подготовка тестов, их выполнение и поиск разного рода ошибок и отказов в компонентах и в системе в целом;
- служба проведения и управление процессом тестирования;
- анализ результатов тестирования.

Объекты тестирования – компоненты, группы компонентов, подсистемы и система. Для каждого из них формируется стратегия проведения тестирования. Если *объект* тестирования относится к "белому ящику" или "черному ящику", состав компонентов которого неизвестный, то тестирование проводится посредством ввода внего входных тестовых данных для получения выходных данных. Стратегическая цель тестирования состоит в том, чтобы убедиться, что каждый рассматриваемый *входной* набор данных соответствует ожидаемым выходным выходных данным. При таком подходе к тестированию не требуется знания внутренней структуры и логики объекта тестирования.

Проектировщик тестов должен заглянуть внутрь "черного ящика" и исследовать детали процессов обработки данных, вопросы обеспечения защиты и восстановления данных, а также интерфейсы с другими программами и системами. Это способствует подготовке тестовых данных для проведения тестирования.

Для некоторых типов объектов *группа* тестирования не может сгенерировать представительное множество тестовых наборов, которые демонстрировали бы функциональную правильность работы компоненты при всех возможных наборах тестов.

Поэтому предпочтительным является метод "белого ящика", при котором можно использовать структуру объекта для организации тестирования *по* различным ветвям. Например, можно выполнить тестовые наборы, которые проходят через все *операторы* или все контрольные точки компонента для того, чтобы убедиться в правильности их работы.

7.3.1. Методы поиска ошибок в программах

Международный стандарт ANSI/IEEE-729-83 разделяет все ошибки в разработке программ на следующие типы.

Ошибка (error) – состояние программы, при котором выдаются неправильные результаты, причиной которых являются изъяны (*flaw*) в операторах программы или в технологическом процессе ее разработки, что приводит к неправильной интерпретации исходной информации, следовательно, и к неверному решению.

Дефект (fault) в программе – следствие ошибок разработчика на любом из этапов разработки, которая может содержаться в исходных или проектных спецификациях, текстах кодов программ, эксплуатационной документация и т.п. В процессе выполнения программы может быть обнаружен дефект или сбой.

Отказ (failure) – это отклонение программы от функционирования или невозможность программы выполнять функции, определенные требованиями и ограничениями, что рассматривается как событие, способствующее переходу программы в неработоспособное состояние из-за ошибок, скрытых в ней дефектов или сбоев в среде функционирования [7.6, 7.11]. Отказ может быть результатом следующих причин:

- ошибочная спецификация или пропущенное требование, означающее, что спецификация точно не отражает того, что предполагал пользователь;
- спецификация может содержать требование, которое невозможно выполнить на данной аппаратуре и программном обеспечении;
- проект программы может содержать ошибки (например, база данных спроектирована без средств защиты от несанкционированного доступа пользователя, а требуется защита);
- программа может быть неправильной, т.е. она выполняет несвойственный алгоритм или он реализован не полностью.

Таким образом, отказы, как правило, являются результатами одной или более ошибок в программе, а также наличия разного рода дефектов.

Ошибки на этапах процесса тестирования.Приведенные типы ошибок распределяются по этапам ЖЦ и им соответствуют такие источники их возникновения [7.12]:

- непреднамеренное отклонение разработчиков от рабочих стандартов или планов реализации;
- спецификации функциональных и интерфейсных требований выполнены без соблюдения стандартов разработки, что приводит к нарушению функционирования программ;
- организации процесса разработки – несовершенная или недостаточное управление руководителем проекта ресурсами (человеческими, техническими, программными и т.д.) и вопросами тестирования и интеграции элементов проекта.

Рассмотрим процесс тестирования, исходя из рекомендаций стандарта ISO/IEC 12207, и приведем типы ошибок, которые обнаруживаются на каждом процессе ЖЦ.

Процесс разработки требований. При определении исходной концепции системы и исходных требований к системе возникают ошибки аналитиков при спецификации верхнего уровня системы и построении концептуальной модели предметной области.

Характерными ошибками этого процесса являются:

- неадекватность спецификации требований конечным пользователям;- некорректность спецификации взаимодействия ПО со средой функционирования или с пользователями;
- несоответствие требований заказчика к отдельным и общим свойствам ПО;
- некорректность описания функциональных характеристик;
- необеспеченность инструментальными средствами всех аспектов реализации требований заказчика и др.

Процесс проектирования.Ошибки при проектировании компонентов могут возникать при описании алгоритмов, логики управления, структур данных, интерфейсов, логики моделирования потоков данных, форматов ввода-вывода и др. В основе этих ошибок лежат дефекты спецификаций аналитиков и недоработки проектировщиков. К ним относятся ошибки, связанные:

- с определением интерфейса пользователя со средой;
- с описанием функций (неадекватность целей и задач компонентов, которые обнаруживаются при проверке комплекса компонентов);
- с определением процесса обработки информации и взаимодействия между процессами (результат некорректного определения взаимосвязей компонентов и процессов);
- с некорректным заданием данных и их структур при описании отдельных компонентов и ПС в целом;
- с некорректным описанием алгоритмов модулей;
- с определением условий возникновения возможных ошибок в программе;
- с нарушением принятых для проекта стандартов и технологий.

Этап кодирования.На данном этапе возникают ошибки, которые являются результатом дефектов проектирования, ошибок программистов и менеджеров в процессе разработки и отладки системы. Причиной ошибок являются:

- бесконтрольность значений входных параметров, индексов массивов, параметров циклов, выходных результатов, деления на 0 и др.;
- неправильная обработка нерегулярных ситуаций при анализе кодов возврата от вызываемых подпрограмм, функций и др.;
- нарушение стандартов кодирования (плохие комментарии, нерациональное *выделение модулей* и компонент и др.);
- использование одного имени для обозначения разных объектов или разных имен одного объекта, плохая мнемоника имен;- несогласованное внесение изменений в программу разными разработчиками и др.

Процесс тестирования.На этом процессе ошибки допускаются программистами и тестировщиками при выполнении технологии сборки и тестирования, выбора тестовых наборов и сценариев тестирования и др. Отказы в программном обеспечении, вызванные такого рода ошибками, должны выявляться, устраняться и не отражаться на статистике ошибок компонент и программного обеспечения в целом.

Процесс сопровождения.На процессе сопровождения обнаруживаются ошибки, причиной которых являются недоработки и дефекты эксплуатационной документации, недостаточные показатели модифицируемости и удобочитаемости, а также некомпетентность лиц, ответственных за сопровождение и/или усовершенствование ПО. В зависимости от сущности вносимых изменений на этом этапе могут возникать практически любые ошибки, аналогичные ранее перечисленным ошибкам на предыдущих этапах.

Все ошибки, которые возникают в программах, принято подразделять на следующие классы [7.12]:

- логические и функциональные ошибки;
- ошибки вычислений и времени выполнения;
- ошибки вводавывода и манипулирования данными;
- ошибки интерфейсов;
- ошибки объема данных и др.

Логические ошибки являются причиной нарушения логики алгоритма, внутренней несогласованности переменных и операторов, а также правил программирования. Функциональные ошибки – следствие неправильно определенных функций, нарушения порядка их применения или отсутствия полноты их реализации и т.д.

Ошибки вычислений возникают по причине неточности исходных данных и реализованных формул, погрешностей методов, неправильного применения операций вычислений или операндов. Ошибки времени выполнения связаны с необеспечением требуемой скорости обработки запросов или времени восстановления программы.

Ошибки ввода-вывода и манипулирования данными являются следствием некачественной подготовки данных для выполнения программы, сбоев при занесении их в базы данных или при выборке из нее.

Ошибки интерфейса относятся к ошибкам взаимосвязи отдельных элементов друг с другом, что проявляется при передаче данных между ними, а также при взаимодействии со средой функционирования.

Ошибки объема относятся к данным и являются следствием того, что реализованные методы доступа и размеры баз данных не удовлетворяют реальным объемам информации системы или интенсивности их обработки.

Приведенные основные классы ошибок свойственны разным типам компонентов ПО и проявляются они в программах по разному. Так, при работе с БД возникают ошибки представления и манипулирования данными, *логические ошибки* в задании прикладных процедур обработки данных и др. В программах вычислительного характера преобладают ошибки вычислений, а в программах управления и обработки - логические и функциональные ошибки. В ПО, которое состоит из множества разноплановых программ, реализующих разные функции, могут содержаться ошибки разных типов. Ошибки интерфейсов и нарушение объема характерны для любого типа систем.

Анализ типов ошибок в программах является необходимым условием создания планов тестирования и методов тестирования для обеспечения правильности ПО.

На современном этапе развития средств поддержки разработки ПО (*CASE-технологии*, объектно-ориентированные методы и средства проектирования моделей и программ) проводится такое проектирование, при котором ПО защищается от наиболее типичных ошибок и тем самым предотвращается появление программных дефектов.

Связь ошибки с отказом.Наличие ошибки в программе, как правило, приводит к отказу ПО при его функционировании. Для анализа причинно-следственных связей "ошибкаотказ" выполняются следующие действия:

- идентификация изъянов в технологиях проектирования и программирования;
- взаимосвязь изъянов процесса проектирования и допускаемых человеком ошибок;
- классификация отказов, изъянов и возможных ошибок, а также дефектов на каждом этапе разработки;- сопоставление ошибок человека, допускаемых на определенном процессе разработки, и дефектов в объекте, как следствий ошибок спецификации проекта, моделей программ;
- проверка и защита от ошибок на всех этапах ЖЦ, а также обнаружение дефектов на каждом этапе разработки;
- сопоставление дефектов и отказов в ПО для разработки системы взаимосвязей и методики локализации, сбора и анализа информации об отказах и дефектах;
- разработка подходов к процессам документирования и испытания ПО.

Конечная цель причинно-следственных связей "ошибкаотказ" заключается в определении методов и средств тестирования и обнаружения ошибок определенных классов, а также критериев завершения тестирования на множестве наборов данных; в определении путей совершенствования организации процесса разработки, тестирования и сопровождения ПО.

Приведем следующую классификацию типов отказов:

- аппаратный, при котором общесистемное ПО не работоспособно;
- информационный, вызванный ошибками во входных данных и передаче данных по каналам связи, а также при сбое устройств ввода (следствие аппаратных отказов);
- эргономический, вызванный ошибками оператора при его взаимодействии с машиной (этот отказ - вторичный отказ, может привести к информационному или функциональному отказам);
- программный, при наличии ошибок в компонентах и др.

Некоторые ошибки могут быть следствием недоработок при определении требований, проекта, генерации выходного кода или документации. С другой стороны, они порождаются в процессе разработки программы или при разработке интерфейсов отдельных элементов программы (нарушение порядка параметров, меньше или больше параметров и т.п.).

Источники ошибок.Ошибки могут быть порождены в процессе разработки проекта, компонентов, кода и документации. Как правило, они обнаруживаются при выполнении или сопровождении программного обеспечения в самых неожиданных и разных ее точках.

Некоторые ошибки в программе могут быть следствием недоработок при определении требований, проекта, генерации кода или документации. С другой стороны, ошибки порождаются в процессе разработки программы или интерфейсов ее элементов (например, при нарушении порядка задания параметров связи – меньше или больше, чем требуется и т.п.).

Причиной появления ошибок – непонимание требований заказчика; неточная спецификация требований в документах проекта и др. Это приводит к тому, что реализуются некоторые функции системы, которые будут работать не так, как предлагает заказчик. В связи с этим проводится совместное обсуждение заказчиком и разработчиком некоторых деталей требований для их уточнения.

Команда разработчиков системы может также изменить синтаксис и семантику описания системы. Однако некоторые ошибки могут быть не обнаружены (например, неправильно заданы индексы или значения переменных этих операторов).

7.3.2. Классификация ошибок и тестов

Каждая организация, разрабатывающая ПО общесистемного назначения, сталкивается с проблемами нахождения ошибок. Поэтому приходится классифицировать типы обнаруживаемых ошибок и определять свое отношение к устранению этих ошибок.

На основе многолетней деятельности в области создания ПО разные фирмы создали свою классификацию ошибок, основанную на выявлении причин их появления в процессе разработки, в функциях и в области функциональной деятельности ПО.

Известно много различных подходов к классификации ошибок, рассмотрим некоторые из них.

Фирма IBM разработала подход к классификации ошибок, называемый ортогональной классификацией дефектов [7.4]. Подход предусматривает разбиение ошибок по категориям с соответствующей ответственностью разработчиков за них.

Схема классификации не зависит от продукта, организации разработки и может применяться ко всем стадиям разработки ПО разного назначения. табл. 7.1 дает список ошибок согласно данной классификации. Используя эту таблицу, разработчик имеет возможность идентифицировать не только типы ошибок, но и места, где пропущены или совершены ошибки. Предусмотрены ситуации, когда найдена неинициализированная переменная или инициализированной переменной присвоено неправильное значение.

Ортогональность схемы классификации заключается в том, что любой ее термин принадлежит только одной категории.

Таблица 7.1. Ортогональная классификация дефектов IBM

Контекст ошибки	Классификация дефектов
Функция	Ошибки интерфейсов конечных пользователей ПО, вызванные аппаратурой или связаны с глобальными структурами данных
Интерфейс	Ошибки во взаимодействии с другими компонентами, в вызовах, макросах, *управляющих блоках* или в списке параметров
Логика	Ошибки в программной логике, неохваченной валидацией, а также в использовании значений переменных
Присваивание	Ошибки в структуре данных или в инициализации переменных отдельных частей программы
Зацикливание	Ошибки, вызванные ресурсом времени, реальным временем или разделением времени
Среда	Ошибки в репозитории, в управлении изменениями или в контролируемых версиях проекта
Алгоритм	Ошибки, связанные с обеспечением эффективности, корректности алгоритмов или структур данных системы
Документация	Ошибки в записях документов сопровождения или в публикациях

Другими словами, прослеживаемая ошибка в системе должна находиться в одном из классов, что дает возможность разным разработчикам классифицировать ошибки одинаковым способом.

Фирма Hewlett-Packard использовала классификацию Буча, установив процентное соотношение ошибок, обнаруживаемых в ПО на разных стадиях разработки (рис. 7.2) [7.12].

Это соотношение – типичное для многих фирм, производящих ПО, имеет некоторые отклонения.

Исследования фирм IBM показали, чем позже обнаруживается ошибка в программе, тем дороже обходится ее исправление, эта зависимость близка к экспоненциальной. Так военновоздушные силы США оценили стоимость разработки одной инструкции в 75 долларов, а ее стоимость сопровождения составляет около 4000 долларов.

Рис. 7.2. Процентное соотношение ошибок при разработке ПО

Согласно данным [7.6, 7.11] стоимость анализа и формирования требований, внесения в них изменений составляет примерно 10%, аналогично оценивается стоимость спецификации продукта. Стоимость кодирования оценивается более чем 20%, а стоимость тестирования продукта составляет более 45% от его общей стоимости. Значительную часть стоимости составляет сопровождение готового продукта и исправление обнаруженных в нем ошибок.

Определение теста. Для проверки правильности программ специально разрабатываются тесты и тестовые данные. Под *тестом* понимается некоторая программа, предназначенная для проверки работоспособности другой программы и обнаружения в ней ошибочных ситуаций. Тестовую проверку можно провести также путем введения в проверяемую программу *отладочных операторов*, которые будут сигнализировать о ходе ее выполнения и получения результатов.

Тестовые данные служат для проверки работы системы и составляются разными способами: генератором тестовых данных, проектной группой на основе внемашинных документов или имеющихся файлов, пользователем по спецификациям требований и др. Очень часто разрабатываются специальные формы входных документов, в которых отображается процесс выполнения программы с помощью тестовых данных [7.11].

Создаются тесты, проверяющие:

- полноту функций;
- согласованность интерфейсов;
- корректность выполнения функций и правильность функционирования системы в заданных условиях;- надежность выполнения системы;
- защиту от сбоев аппаратуры и не выявленных ошибок и др.

Тестовые данные готовятся как для проверки отдельных программных элементов, так и для групп программ или комплексов на разных стадиях процесса разработки. На рис. 7.3. приведена классификация тестов проверки по объектам тестирования на основных этапах разработки.

увеличить изображение

Рис. 7.3. Классификация тестов проверки

Многие типы тестов готовятся заказчиком для проверки работы программной системы. Структура и содержание тестов зависят от вида тестируемого элемента, которым может быть модуль, компонент, группа компонентов, подсистема или система. Некоторые тесты зависят от цели и необходимости знать: работает ли система в соответствии с ее проектом, удовлетворены ли требования и участвует ли заказчик в проверке работы тестов и т.п.

В зависимости от задач, которые ставятся перед тестированием программ, составляются тесты проверки промежуточных результатов проектирования элементов на этапах ЖЦ, а также создаются тесты испытаний окончательного кода системы.

Тесты интегрированной системы. Тесты для проверки отдельных элементов системы и тесты интегрированной системы имеют общие и отличительные черты. Так, на рис. 7.4 в качестве примера приведена схема интеграции готовых оттестированных элементов. В ней заданы связи между разными уровнями тестирования интегрируемой ПС.

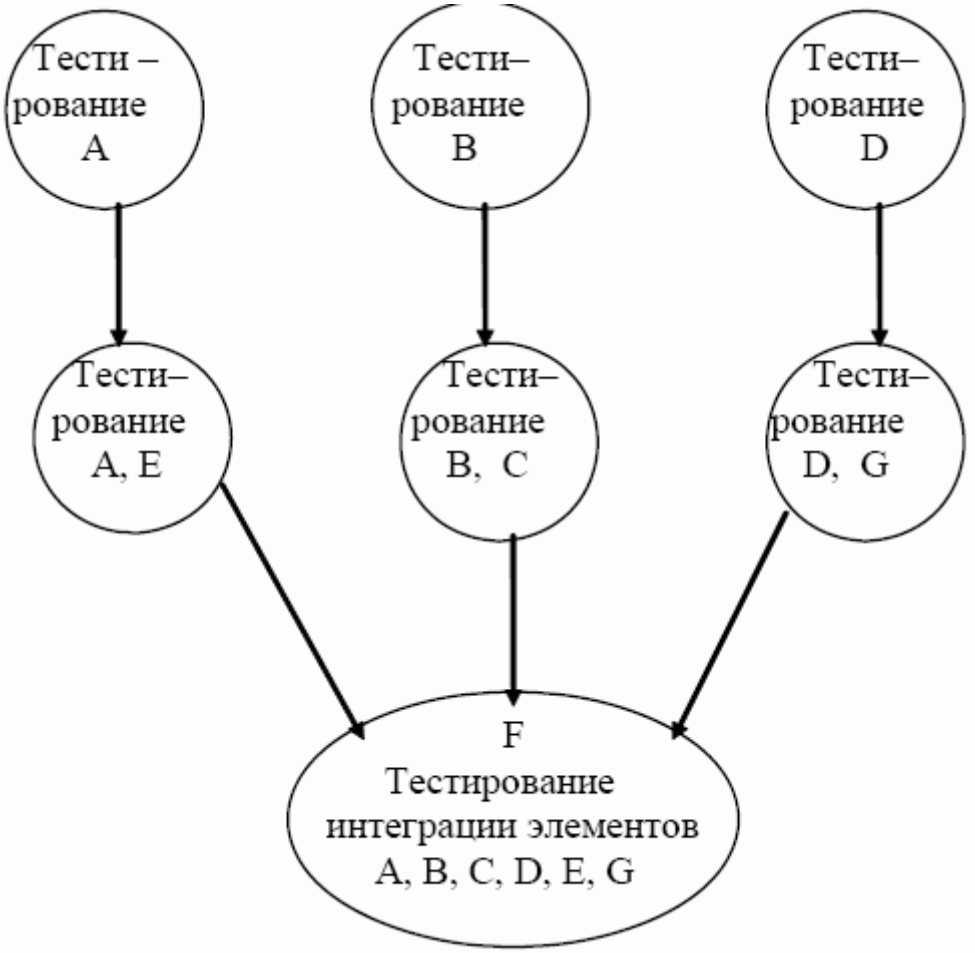

Рис. 7.4. Интегрированное тестирование компонент

Рассмотрим этот процесс более подробно. Каждый компонент этой схемы тестируется отдельно от других компонентов с помощью тестов, включающих наборы данных и сценарии, составленные в соответствии с их типами и функциями, специфицированные в проекте системы. Тестирование проводится в контрольной операционной среде на предопределенном множестве тестовых данных и операциях, производимых над ними.

Тесты обеспечивают проверку внутренней структуры, логики и граничных условий выполнения для каждого компонента.

Согласно приведенной схеме сначала тестируются компоненты A, B, D независимо друг от друга и каждый с отдельным тестом. После их проверки выполняется проверка интерфейсов для последующей их интеграции, суть которой заключается в анализе выполнения операторов вызова A -> E, B -> C, D -> G, на нижних уровнях графа: компоненты E, C, G. При этом предполагается, что указанные вызываемые компоненты, так же должны быть отлажены отдельно. Аналогично проверяются все обращения к компоненту F, являющемуся связывающим звеном с вышележащими элементами.

При этом могут возникать ошибки, в случае неправильного задания параметров в операторах вызова или при вычислениях процедур или функций. Возникающие ошибки в связях устраняются, а затем повторно проверяется связь с компонентом F в виде троек: компонентинтерфейскомпонент.Следующим шагом тестирования комплексной системы является проверка функционирования системы с помощью тестов проверки функций и требований к ним. После проверки системы на функциональных тестах происходит проверка на исполнительных и испытательных тестах, подготовленных согласно требованиям к ПО, аппаратуре и выполняемым функциям. Испытательному тесту предшествует верификация и валидация ПО.

Тест испытаний системы в соответствии с требованиями заказчика проверяется в реальной среде, в которой система будет в дальнейшем функционировать.

7.3.1. Служба тестирования ПС

За функциональные и исполнительные тесты несут ответственность разработчики заказчик, он больше влияет на составление тестов испытаний и инсталляции системы [7.6].

Для этих целей, как правило, создается служба проверяющих ПС – команда тестировщиков, которая не зависит от штата разработчиков ПС. Некоторые члены этой команды – опытные или даже профессионалы в этой области. К ним относятся аналитики, программисты, инженерытестировщики, которые посвящают в *проблемы тестирования* систем с начала разработки. Они имеют дело не только со спецификациями, но и с методами и средствами тестирования, организуют создание и выполнение тестов. С самого начала создания

проекта тестировщики составляют планы тестирования, тестовые данные и сценарии, а также графики выполнения тестов.

Профессиональные тестировщики работают совместно с группой управления конфигурацией, чтобы обеспечить их документацией и другими механизмами для связи между собой тестов с требованиями проекта, конфигурацией и кодом. Они разрабатывают методы и процедуры тестирования. В эту команду включаются дополнительные специалисты, которые знакомы с требованиями системы или с подходами к их разработке. Аналитиков включают в команду, так как они понимают проблемы определения спецификаций заказчиков.

Многие специалисты сравнивают тестирование системы с созданием новой системы, в которой аналитики отражают потребности и цели заказчика, работая совместно с проектировщиками и добиваясь реализации идей и принципов работы системы.Проектировщики системы сообщают команде тестировщиков проектные цели, чтобы они знали декомпозицию системы на подсистемы и ее функции, а также принципы их работы. После проектирования тестов и тестовых покрытий, команда тестировщиков проводит анализ возможностей системы.

Так как тесты и тестовые сценарии являются прямым отражением требований и проекта в целом, перспективы управления конфигурацией системы определяются именно этой командой. Обнаруживаемые в программе ошибки и изменения в системе отражаются в документации, требованиях, проекте, а также в описаниях входных и выходных данных или в других разрабатываемых артефактах. Вносимые изменения в процессе разработки приводят к модификации тестовых сценариев или в большей части к изменению планов тестирования. Специалисты по управлению конфигурацией учитывают эти изменения и координируют составление тестов.

В команду тестировщиков входят также пользователи. Они оценивают получаемые результаты, удобство использования, а также высказывают свое мнение о принципах работы системы.

Уполномоченные заказчика планируют работы для тех пор, пока используется и сопровождается система. При этом они могут привнести некоторые изменения в проект из-за неполноты заданных требований и сформулировать системные требования для проведения верификации системы и принятия решений о ее готовности и полезности.

План тестирования.Для проведения тестирования создается план (*Test Plan*), в котором описываются стратегии, ресурсы и график тестирования отдельных компонентов и системы в целом. В плане отмечаются работы для разных членов команды, которые выполняют определенные роли в этом процессе. План включает также определение роли тестов в каждом процессе, степень покрытия программы тестами и процент тестов, которые выполняются со специальными данными.

Тестовые инженеры создают множество тестовых сценариев (*Test Cases*), каждый из которых проверяет результат взаимодействия между актором и системой на основе пред- и постусловий использования таких сценариев. Сценарии в основном относятся к тестированию по типу белого "ящика" и ориентированы на проверку структуры и операций интеграции компонентов системы.

Для проведения тестирования тестовые инженеры предлагают процедуры тестирования (*Test Procedures*), включающие валидациюобъектов и верификацию тестовых сценариев в соответствии с планом графикам. Оценка тестов (Test Evaluation) заключается в оценке результатов тестирования, степени покрытия программ сценариями и статуса полученных ошибок.

На рис. 7.5. приведен круг обязанностей инженератестировщика.

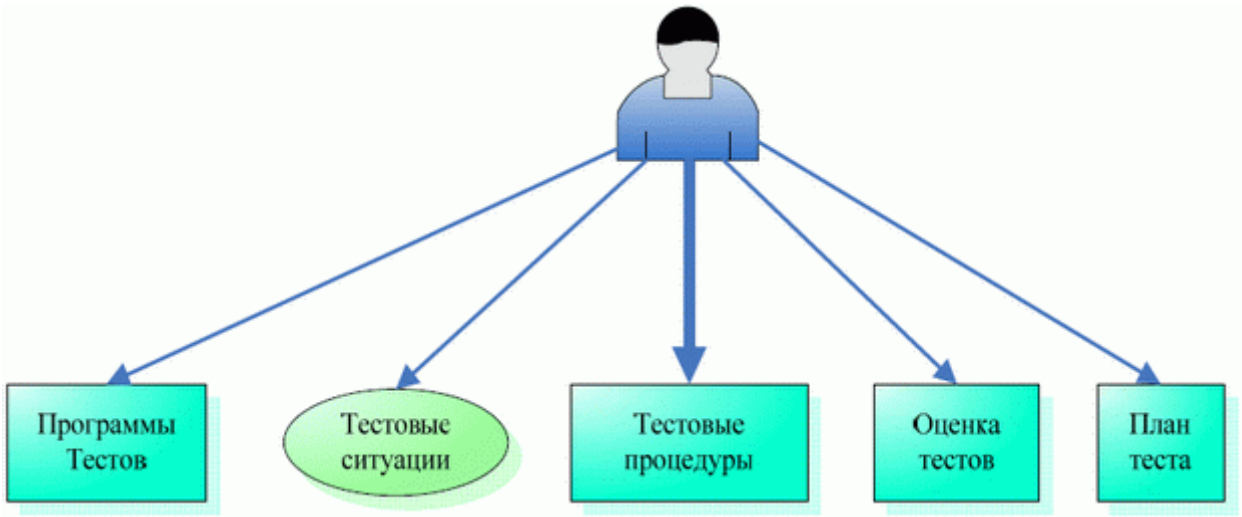

Рис. 7.5. Ответственности инженера-тестировщика

Тестировщик интегрированной системы проводит тестирование интерфейсов и дает оценку результатов выполнения соответствующих тестов с помощью создаваемых им системных тестов, выполняет анализ результатов тестирования, проведенного с отдельными элементами системы. При выполнении системных тестов, как правило, находятся дефекты, как результат глубоко скрытых погрешностей в программах, обнаруживаемых при длительной прогонке системы на тестовых данных и сценариях.

7.3.2. Управление процессом тестирования

Все способы тестирования ПС объединяются базой данных, где помещаются результаты тестирования системы. В ней содержатся все компоненты, тестовые контрольные данные, результаты тестирования и информация о документировании процесса тестирования.

База данных проекта поддерживается специальными инструментальными средствами типа CASE, которые обеспечивают ведение анализа ПрО, сборку данных об их объектах, потоках данных и тому подобное. База данных проекта хранит также начальные и эталонные данные, которые используются для сопоставления данных,накопленных в базе, с данными, которые получены в процессе тестирования системы.

При тестировании выполняются разные *виды расчетов* характеристик этого процесса и способы планирования и управления.

1. Расчет продолжительности выполнения функций путем сбора средних показателей скорости выполнения операторов без выполнения программы на машине. Выявляются компоненты, которые требуют большого времени выполнения в реальной среде.
2. Управление выполнением тестирования путем подбора тестов проверки, их выполнения, селекции результатов тестирования и проведения сопоставления с эталонными значениями. Результаты данного процесса отображаются на дисплее, например, в графической форме (пути прохождения по графу программы), в виде диаграмм UML, данных об отказах и ошибках или конкретных значений исходных параметров программы. Эти данные анализируются разработчиками для формулирования выводов о направлениях дальнейшей проверки правильности программы или их завершении.
3. Планирование тестирования предназначено для распределения сроков работ по тестированию, распределения тестировщиков по отдельным видам работ и составления ими тестов проверки системы. Определяются стратегия и пути тестирования. В диалоге запрашиваются данные о реальных значениях процесса выполнения системы, структуре ветвления вершин графа и параметрах циклов. Проверенные циклы, как правило, изымаются из путей выполнения программы. При планировании путей выполнения создаются соответствующие тесты, критерии и входные значения.

Документирование результатов тестирования в соответствии с действующим стандартом ANSI/IEEE 829 включает:

описание задач, назначение и содержание ПС, а также перечень функций в соответствии с требованиями заказчика;

технологии разработки системы;

планов тестирования различных объектов, необходимых ресурсов, соответствующих специалистов для проведения тестирования и технологических способов;

тестов, контрольных примеров, критериев и ограничений, оценки результатов программного продукта, а также процесса тестирования;

учета процесса тестирования, составление отчетов об аномальных событиях, отказах и дефектах в итоговом документе системы.

Таким образом, были рассмотрены современные методы и процессы верификации и тестирования ПС, основанные на понятии программы – "белый ящик" и "черный ящик", а также на анализе реализованных в ПС функций. Определены критерии тестирования, типы ошибок, обнаруживаемых в программах, а также отказы и ошибки на этапах ЖЦ процесса тестирования. Сформулированы цели и задачи группы тестировщиков.

Контрольные вопросы и задания

1. Назовите формальные методы проверки правильности программ.
2. Какие процессы проверки зафиксированы в стандарте?
3. Какие функции у процесса *верификации программ*?
4. Назовите основные задачи процесса валидации программ.
5. Сравните задачи процессов верификации и валидации программ.
6. В чем отличие верификации и валидации?
7. Определите процесс тестирования.
8. Назовите методы тестирования.
9. Объясните значения терминов "черный ящик", "белый ящик".

10. Назовите объекты тестирования и подходы к их тестированию.
11. Какая существует классификация типов ошибок в программах?
12. Определите основные процессы ЖЦ тестирования ПО.
13. Наведите классификацию тестов для проверки ПО.
14. Какие задачи выполняет группа текстовиков?
15. Какая организация работ в проведении тестирования?

Внимание! Если Вы увидите ошибку на нашем сайте, выделите её и нажмите Ctrl+Enter.