

Программная инженерия

Тестирование программного обеспечения (Software Testing)

Глава базируется на IEEE Guide to the Software Engineering Body of Knowledge - SWEBOK®, 2004. Содержит перевод описания области знаний SWEBOK® "Software Testing", с комментариями и замечаниями.

"Основы программной инженерии" разработаны на базе IEEE Guide to SWEBOK® 2004 в соответствии с IEEE SWEBOK 2004 Copyright and Reprint Permissions: "This document may be copied, in whole or in part, in any form or by any means, as is, or with alterations provided that (1) alterations are clearly marked as alterations and (2) this copyright notice is included unmodified in any copy."

Русский перевод SWEBOK 2004 с замечаниями и комментариями подготовлены [Сергеем Орликом](#) при участии [Юрия Булуя](#). Дополнительные главы написаны [Сергеем Орликом](#). Текст расширений SWEBOK отмечен цветом, отличным от перевода оригинального текста.

"Основы программной инженерии" Copyright © 2004-2010 [Сергей Орлик](#). Все права защищены.
[SWEBOK](#) Copyright © 2004 by The Institute of Electrical and Electronics Engineers, Inc. All rights reserved.

Официальный сайт "Основ программной инженерии" (по SWEBOK) - <http://swebok.sorlik.ru>

Программная инженерия

Тестирование программного обеспечения (Software Testing)

Программная инженерия	2
Тестирование программного обеспечения (Software Testing).....	2
1. Основы тестирования (Software Testing Fundamentals).....	4
1.1 Терминология тестирования (Testing-Related Terminology)	4
1.2 Ключевые вопросы (Key Issues).....	4
1.3 Связь тестирования с другой деятельностью (Relationships of testing with other activities)	5
2. Уровни тестирования (Test Levels).....	5
2.1 Над чем производятся тесты (The target of the test)	5
2.2 Цели тестирования (Objectives of Testing)	6
3. Техники тестирования (Test Techniques).....	8
3.1 Техники, базирующиеся на интуиции и опыте инженера (Based on the software engineer's intuition and experience)	8
3.2 Техники, базирующиеся на спецификации (Specification-based techniques).....	9
3.3 Техники, ориентированные на код (Code-based techniques)	9
3.4 Тестирование, ориентированное на дефекты (Fault-based techniques).....	10
3.5 Техники, базирующиеся на условиях использования (Usage-based techniques).....	10
3.6 Техники, базирующиеся на природе приложения (Techniques based on the nature of the application)	11
3.7 Выбор и комбинация различных техник (Selecting and combining techniques).....	11
4. Измерение результатов тестирования (Test-related measures).....	11
4.1 Оценка программ в процессе тестирования (Evaluation of the program under test, IEEE 982.1-98)	11
4.2 Оценка выполненных тестов (Evaluation of the tests performed).....	12
5. Процесс тестирования (Test Process)	13
5.1 Практические соображения (Practical considerations).....	13
5.2 Тестовые работы (Test Activities)	15

Тестирование (software testing) – деятельность, выполняемая для оценки и улучшения качества программного обеспечения. Эта деятельность, в общем случае, базируется на обнаружении дефектов и проблем в программных системах.

Тестирование программных систем состоит из *динамической* верификации поведения программ на *конечном (ограниченном)* наборе тестов (set of test cases), *выбранных* соответствующим образом из обычно выполняемых действий прикладной области и обеспечивающих проверку соответствия *ожидаемому* поведению системы.

В данном определении тестирования выделены слова, определяющие основные вопросы, которым адресуется данная область знаний:

- *Динамичность (dynamic)*: этот термин подразумевает тестирование всегда предполагает выполнение тестируемой программы с заданными входными данными. При этом, величины, задаваемые на вход тестируемому программному обеспечению, не всегда достаточны для определения теста. Сложность и недетерминированность систем приводит к тому, что система может по-разному реагировать на одни и те же входные параметры, в зависимости от состояния системы. В данной области знаний термин “вход” (input) будет использоваться в рамках соглашения о том, что вход может также специфицировать состояние системы, в тех случаях, когда это необходимо. Кроме динамических техник проверки качества, то есть тестирования, существуют также и статические техники, рассматриваемые в области знаний “Software Quality”.
- *Конечность (ограниченность, finite)*: даже для простых программ теоретически возможно столь большое количество тестовых сценариев, что исчерпывающее тестирование может занять многие месяцы и даже годы. Именно поэтому, с практической точки зрения, всестороннее тестирование считается бесконечным. Тестирование всегда предполагает

компромисс между ограниченными ресурсами и заданными сроками, с одной стороны, и практически неограниченными требованиями по тестированию, с другой. То есть мы снова говорим об определении характеристик “приемлемого” качества, на основе которых планируем необходимы объем тестирования.

- *Выбор (selection)*: многие предлагаемые техники тестирования отличаются друг от друга в том, как выбираются сценарии тестирования. Инженеры по программному обеспечению должны обладать представлением о том, что различные критерии выбора тестов могут давать разные результаты, с точки зрения эффективности тестирования. Определение подходящего набора тестов для заданных условий является очень сложной проблемой. Обычно, для выбора соответствующих тестов совместно применяют техники анализа рисков, анализ требований и соответствующую экспертизу в области тестирования и заданной прикладной области.
- *Ожидаемое поведение (expected behaviour)*: Хотя это не всегда легко, все же необходимо решить, какое наблюдаемое поведение программы будет приемлемо, а какое – нет. В противном случае, усилия по тестированию – бесполезны. Наблюдаемое поведение может рассматриваться в контексте пользовательских ожиданий (подразумевая “тестирования для проверки” - testing for validation), спецификации (“тестирование для аттестации” - testing for verification) или, наконец, в контексте предсказанного поведения на основе неявных требований или обоснованных ожиданий. См. тему SWEBOK 6.4 “Приемочные тесты” области знаний “Software Requirements”.

Общий взгляд на тестирование программного обеспечения последние годы активно эволюционировал, становясь все более конструктивным, прагматичным и приближенным к реалиям современных проектов разработки программных систем. Тестирование более не рассматривается как деятельность, начинающаяся только после завершения фазы конструирования. Сегодня тестирование рассматривается как деятельность, которую необходимо проводить на протяжении всего процесса разработки и сопровождения и является важной частью конструирования программных продуктов. Действительно, планирование тестирования должно начинаться на ранних стадиях работы с требованиями, необходимо систематически и постоянно развивать и уточнять планы тестов и соответствующие процедуры тестирования. Даже сами по себе сценарии тестирования оказываются очень полезными для тех, кто занимается проектированием, позволяя выделять те аспекты требований, которые могут неоднозначно интерпретироваться или даже быть противоречивыми.

Не секрет, что легче предотвратить проблему, чем бороться с ее последствиями. Тестирование, наравне с управлением рисками, является тем инструментом, который позволяет действовать именно в таком ключе. Причем действовать достаточно эффективно. С другой стороны, необходимо осознавать, что даже если приемочные тесты показали положительные результаты, это совсем не означает, что полученный продукт не содержит ошибок. Этим вопросам, в частности, адресована область знаний “Сопровождение программного обеспечения” (Software Maintenance). Однако, адекватное внимание вопросам тестирования качественно снижает риск возникновения ошибок на этапе эксплуатации, обеспечивая более высокую удовлетворенность пользователей, что и является, по существу, целью любого проекта.

В области знаний “Качество программного обеспечения” (Software Quality) техники управления качеством четко разделены на *статические* (без выполнения кода) и *динамические* (с выполнением кода). Обе эти категории важны. Данная область знаний - “Тестирование” – касается динамических техник.

Как уже отмечалось ранее, тестирование тесно связано с областью знаний “Конструирование” (Software Construction). Более того, модульное (unit-) и интеграционное тестирование все чаще рассматривают как неотъемлемый элемент деятельности по конструированию.

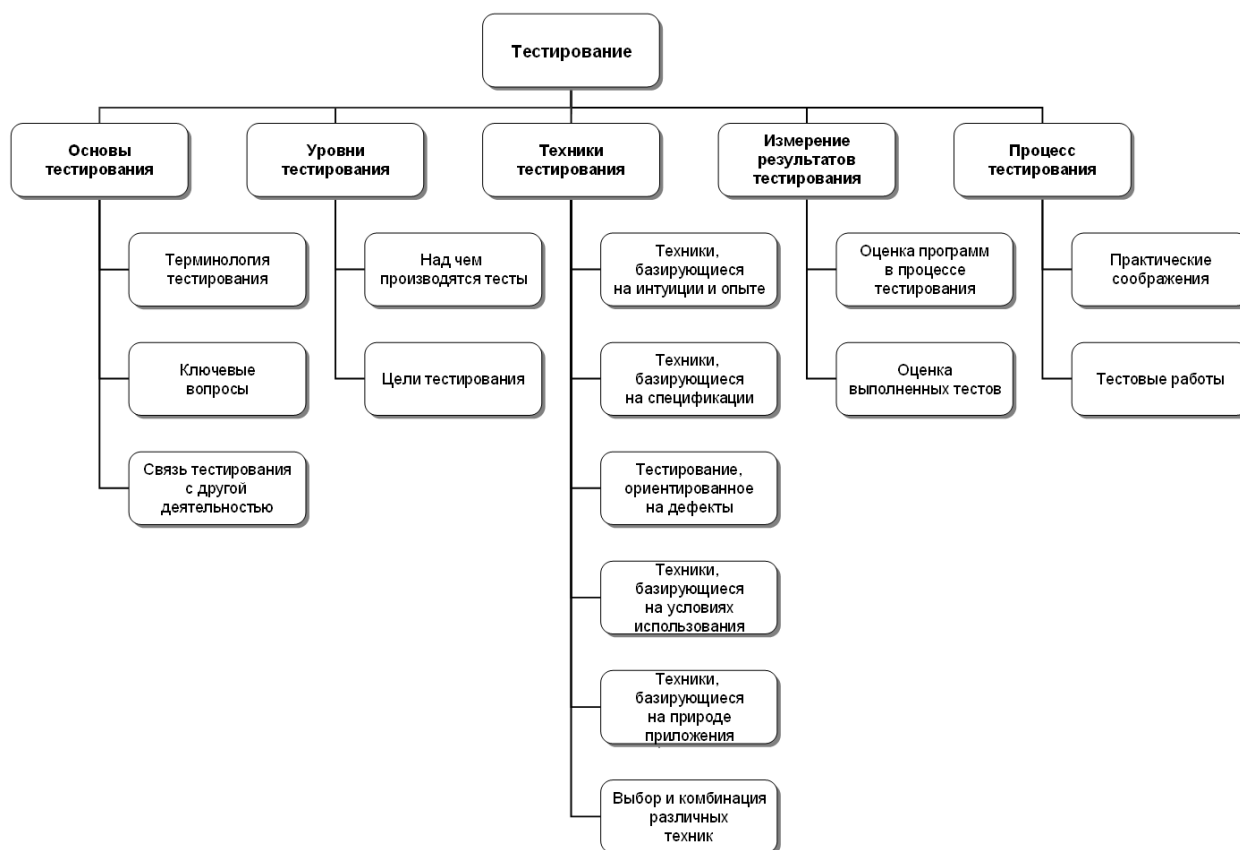


Рисунок 5. Область знаний “Тестирование программного обеспечения” [SWEBOK, 2004, с.5-2, рис. 1]

1. Основы тестирования (Software Testing Fundamentals)

Охватывает основные понятия в области тестирования, базовые термины, ключевые проблемы и их связь другими областями деятельности и знаний.

1.1 Терминология тестирования (Testing-Related Terminology)

1.1.1 Определение тестирования и связанной терминологии достаточно полно даётся в “Глоссарии терминов по программной инженерии” – IEEE Standard 610-90 (Standard Glossary of Software Engineering Terminology).

1.1.2 Недостатки и сбои (Faults vs. Failures)

В литературе, посвященной программной инженерии, встречается множество терминов, описывающих нарушение функционирования программных систем – недостатки (faults), дефекты (defects), сбои (failures), ошибки (errors) и др. Соответствующая терминология, как было указано выше в 1.1.1, описана в IEEE Std. 610-90 также обсуждается в области знаний SWEBOK “Качество программного обеспечения” (Software Quality). Важно чётко разделять *причину* нарушения работы прикладных систем, обычно описываемую терминами *недостаток* или *дефект*, и наблюдаемый нежелательный эффект, вызываемый этими причинами – *сбой*. Термин *ошибка*, в зависимости от контекста, может описывать и как причину сбоя, и сам сбой. Тестирование позволяет обнаружить дефекты, приводящие к сбоям.

Необходимо понимать, что причина сбоя не всегда может быть однозначно определена. Не существует теоретических критериев, позволяющих гарантированно определить какой именно дефект приводит к наблюдаемому сбою.

1.2 Ключевые вопросы (Key Issues)

1.2.1 Критерии отбора тестов/критерии адекватности тестов, правила прекращения тестирования (Test selection criteria/test adequacy criteria, or stopping rules)

Критерии отбора тестов могут использоваться как для создания набора тестов, так и для проверки, насколько выбранные тесты адекватны решаемым задачам (тестирования). При этом, обсуждаемые критерии помогают определить, когда можно или необходимо прекратить тестирование.

1.2.2 Эффективность тестирования/Цели тестирования (Test effectiveness/Objectives for testing)

Тестирование – это наблюдение за выполнением программы, запущенной в целях тестирования с заданными параметрами, по заданному сценарию или с другими заданными начальными условиями или целями тестирования. Эффективность теста может быть определена только в контексте заданных условий.

1.2.3 Тестирование для идентификации дефектов (Testing for defect identification)

Данный случай тестирования подразумевает успешность процедуры тестирования, если дефект найден. Это отличается от подхода в тестировании, когда тесты запускаются для демонстрации того, что программное обеспечение удовлетворяет предъявляемым требованиям и, соответственно, тест считается успешным, если не найдено дефектов.

1.2.4 Проблема оракула (The oracle problem)

“Оракул”, в данном контексте, любой агент (человек или программа), оценивающий поведение программы, формулируя вердикт - тест пройден (“pass”) или нет (“fail”).

1.2.5 Теоретические и практические ограничения тестирования (Theoretical and practical limitation of testing)

Теория тестирования выступает против необоснованного уровня доверия к серии успешно пройденных тестов. К сожалению, большинство установленных результатов теории тестирования – негативны, означая, по словам Дейкстры (Dijkstra), то, что “тестирование программы может использоваться для демонстрации наличия дефектов, но никогда не покажет их отсутствие”. Основная причина этого в том, что полное (всеобъемлющее) тестирование недостижимо для реального программного обеспечения.

1.2.6 Проблема неосуществимых путей (The problem of infeasible paths)

Эта сложнейшая проблема автоматизированного тестирования связана с тем, что путь, по которому выполняются потоки работ тестируемой программной системы, не могут быть заданы входными параметрами.

1.2.7 Тестируемость (Testability)

Это понятие может подразумевать две различных идеи. Первая описывает степень легкости описания критериев покрытия тестами для заданной программной системы. Вторая определяет возможность вероятности, возможность статистического измерения того, что при тестировании проявится сбой программной системы. Обе интерпретации этого понятия одинаково важны для тестирования.

1.3 Связь тестирования с другой деятельностью (Relationships of testing with other activities)

Тестирование программного обеспечения отличается от статических техник управления качеством, проверки корректности, отладки и программирования, но связано со всеми этими работами. Полезно рассматривать тестирование с точки зрения аналитиков и специалистов по сертификации качества.

2. Уровни тестирования (Test Levels)

2.1 Над чем производятся тесты (The target of the test)

Тестирование обычно производится на протяжении всей разработки и сопровождения на разных *уровнях*. Уровень тестирования определяет “над чем” производятся тесты: над отдельным модулем, группой модулей или системой, в целом. При этом ни один из уровней тестирования не может считаться приоритетным. Важны все уровни тестирования, вне зависимости от используемых моделей и методологий.

2.1.1 Модульное тестирование (Unit testing)

Этот уровень тестирования позволяет проверить функционирование отдельно взятого элемента системы. Что считать элементом – *модулем* системы определяется контекстом. Наиболее полно данный вид тестов описан в стандарте IEEE 1008-87 “Standard for Software Unit Testing”, задающем интегрированную концепцию систематического и документированного подхода к модульному тестированию.

2.1.2 Интеграционное тестирование (Integration testing)

Данный уровень тестирования является процессом проверки взаимодействия между программными компонентами/модулями.

Классические стратегии интеграционного тестирования – “сверху-вниз” и “снизу-вверх” – используются для традиционных, иерархически структурированных систем и их сложно применять, например, к тестированию слабосвязанных систем, построенных в сервисно-ориентированных архитектурах (SOA).

Современные стратегии в большей степени зависят от архитектуры тестируемой системы и строятся на основе идентификации функциональных “поток” (например, потоков операций и данных).

Интеграционное тестирование – постоянно проводимая деятельность, предполагающая работу на достаточно высоком уровне абстракции. Наиболее успешная практика интеграционного тестирования базируется на инкрементальном подходе, позволяющем избежать проблем проведения разовых тестов, связанных с тестированием результатов очередного длительного этапа работ, когда количество выявленных дефектов приводит к серьезной переработке кода (традиционно, негативный опыт выпуска и тестирования только крупных релизов называют “big bang”).

2.1.3 Системное тестирование (System testing)

Системное тестирование охватывает целиком всю систему. Большинство функциональных сбоев должно быть идентифицировано еще на уровне модульных и интеграционных тестов. В свою очередь, системное тестирование, обычно фокусируется на нефункциональных требованиях – безопасности, производительности, точности, надежности т.п.

На этом уровне также тестируются интерфейсы к внешним приложениям, аппаратному обеспечению, операционной среде и т.д.

2.2 Цели тестирования (Objectives of Testing)

Тестирование проводится в соответствии с определенными целями (могут быть заданы явно или неявно) и различным уровнем точности. Определение цели точным образом, выражаемым количественно, позволяет обеспечить контроль результатов тестирования.

Тестовые сценарии могут разрабатываться как для проверки функциональных требований (известны как функциональные тесты), так и для оценки нефункциональных требований. При этом, существуют такие тесты, когда количественные параметры и результаты тестов могут лишь опосредованно говорить об удовлетворении целям тестирования (например, “usability” – легкость, простота использования, в большинстве случаев, не может быть явно описана количественными характеристиками).

Можно выделить следующие, наиболее распространенные и обоснованные цели (а, соответственно, виды) тестирования:

2.2.1 Приёмочное тестирование (Acceptance/qualification testing)

Проверяет поведение системы на предмет удовлетворения требований заказчика. Это возможно в том случае, если заказчик берет на себя ответственность, связанную с проведением таких работ, как сторона “принимающая” программную систему, или специфицированы типовые задачи, успешная проверка (тестирование) которых позволяет говорить об удовлетворении требований заказчика.

Такие тесты могут проводиться как с привлечением разработчиков системы, так и без них.

2.2.2 Установочное тестирование (Installation testing)

Из названия следует, что данные тесты проводятся с целью проверки процедуры инсталляции системы в целевом окружении.

2.2.3 Альфа- и бета-тестирование (Alpha and beta testing)

Перед тем, как выпускается программное обеспечение, как минимум, оно должно проходить стадии альфа (внутреннее пробное использование) и бета (пробное использование с привлечением отобранных внешних пользователей) версий. Отчеты об ошибках, поступающие от пользователей этих версий продукта, обрабатываются в соответствии с определенными процедурами, включающими подтверждающие тесты (любого уровня), проводимые специалистами группы разработки.

Данный вид тестирования не может быть заранее спланирован.

2.2.4 Функциональные тесты/тесты соответствия (Conformance testing/Functional testing/Correctness testing)

Эти тесты могут называться по-разному, однако, их суть проста – проверка соответствия системы, предъявляемым к ней требованиям, описанным на уровне спецификации поведенческих характеристик.

2.2.5 Достижение и оценка надежности (Reliability achievement and evaluation)

Помогая идентифицировать причины сбоев, тестирование подразумевает и повышение надежности программных систем. Случайно генерируемые сценарии тестирования могут применяться для статистической оценки надежности. Обе цели – повышение и оценка надежности – могут достигаться при использовании моделей повышения надежности. Эти вопросы затрагиваются и в тематическом фрагменте 4.1.4 “Life test, reliability evaluation”.

2.2.6 Регрессионное тестирование (Regression testing)

Определение успешности регрессионных тестов (IEEE 610-90 “Standard Glossary of Software Engineering Terminology”) гласит: “повторное выборочное тестирование системы или компонент для проверки сделанных модификаций не должно приводить к непредусмотренным эффектам”. На практике это означает, что если система успешно проходила тесты до внесения модификаций, она должна их проходить и после внесения таковых. Основная проблема регрессионного тестирования заключается в поиске компромисса между имеющимися ресурсами и необходимостью проведения таких тестов по мере внесения каждого изменения. В определенной степени, задача состоит в том, чтобы определить критерии “масштабов” изменений, с достижением которых необходимо проводить регрессионные тесты.

2.2.7 Тестирование производительности (Performance testing)

Специализированные тесты проверки удовлетворения специфических требований, предъявляемых к параметрам производительности. Существует особый подвид таких тестов, когда делается попытка достижения количественных пределов, обусловленных характеристиками самой системы и ее операционного окружения.

2.2.8 Нагрузочное тестирование (Stress testing)

Необходимо понимать отличия между рассмотренным выше тестированием производительности с целью достижения ее реальных (достижимых) *возможностей* производительности и выполнением программной системы с повышением нагрузки, вплоть до достижения *запланированных характеристик* и далее, с отслеживанием поведения на всем протяжении повышения загрузки системы.

2.2.9 Сравнительное тестирование (Back-to-back testing)

Единый набор тестов, позволяющих сравнить две версии системы.

2.2.10 Восстановительные тесты (Recovery testing)

Цель – проверка возможностей рестарта системы в случае непредусмотренной катастрофы (disaster), влияющей на функционирование операционной среды, в которой выполняется система.

2.2.11 Конфигурационное тестирование (Configuration testing)

В случаях, если программное обеспечение создается для использования различными пользователями (в терминах “ролей”), данный вид тестирования направлен на проверку поведения и работоспособности системы в различных конфигурациях.

2.2.12 Тестирование удобства и простоты использования (Usability testing)

Цель – проверить, насколько легко конечный пользователь системы может ее освоить, включая не только функциональную составляющую – саму систему, но и ее документацию; насколько эффективно пользователь может выполнять задачи, автоматизация которых осуществляется с использованием данной системы; наконец, насколько хорошо система застрахована (с точки зрения потенциальных сбоев) от ошибок пользователя.

2.2.13 Разработка, управляемая тестированием (Test-driven development)

По-сути, это не столько техника тестирования, сколько стиль организации процесса разработки, жизненного цикла, когда тесты являются неотъемлемой частью требований (и соответствующих спецификаций) вместо того, чтобы рассматриваться независимой деятельностью по проверке удовлетворения требований программной системой.

Иногда говорят о таком стиле разработки как о самостоятельной методологии – TDD. Насколько это верно, зависит от того, что именно понимать под методологией разработки. Скорее, с точки зрения автора, это техника, практика или стиль организации работы, чем самостоятельная методология.

В меньшей степени это относится к FDD – Feature-Driven Development (разработка на основе функциональных возможностей). TDD может естественно рассматриваться как составная часть XP или, как минимум Agile-методов. В свою очередь, FDD может рассматриваться как один из методов гибкой разработки.

В чем отличие столь близких, на первый взгляд, подходов (и, кстати, соответствующих аббревиатур)? Причина – проста. *Тесты – инструмент* достижения характеристик системы, удовлетворяющей заданным требованиям, то есть потребностям пользователей, а “*возможности*” (features) – практически сами (чаще – функциональные) *требования*, воплощенные (в идеальном случае) в код.

3. Техники тестирования (Test Techniques)

3.1 *Техники, базирующиеся на интуиции и опыте инженера (Based on the software engineer's intuition and experience)*

3.1.1 Специализированное тестирование (Ad hoc testing)

Возможно, наиболее широко практикуемая техника. Тесты основываются на опыте, интуиции и знаниях инженера, рассматривающего проблему с точки зрения имевшихся ранее аналогий. Данный вид тестирования может быть полезен для идентификации тех тестов, которые не охватываются рассматривавшимися выше более формализованными техниками.

3.1.2 Исследовательское тестирование (Exploratory testing)

Такое тестирование определяется как одновременное обучение, проектирование теста и его исполнение. Данный вид тестирования заранее не определяется в плане тестирования и такие тесты создаются, выполняются и модифицируются динамически, по мере необходимости. Эффективность исследовательских тестов напрямую зависит от знаний инженера, формируемых на основе поведения тестируемого продукта в процессе проведения тестирования, степени знакомства с приложением, платформой, типами возможных сбоев и дефектов, рисками, ассоциированными с конкретным продуктом и т.п.

3.2 Техники, базирующиеся на спецификации (*Specification-based techniques*)

3.2.1 Эквивалентное разделение <приложения> (Equivalence partitioning)

Рассматриваемая область приложения разделяется на коллекцию наборов или эквивалентных классов, которые считаются эквивалентными с точки зрения рассматриваемых связей и характеристик <спецификации>. Репрезентативный набор тестов (иногда – только один тест) формируется из тестов эквивалентных классов (или наборов классов).

3.2.2 Анализ граничных значений (Boundary-value analysis)

Тесты строятся с ориентацией на использование тех величин, которые определяют предельные характеристики тестируемой системы. Расширением этой техники являются *тесты оценки живучести* (*robustness testing*) системы, проводимые с величинами, выходящими за рамки специфицированных пределов значений.

3.2.3 Таблицы принятия решений (Decision table)

Такие таблицы представляют логические связи между условиями (могут рассматриваться в качестве “входов”) и действиями (могут рассматриваться как “выходы”). Набор тестов строится последовательным рассмотрением всех возможных кросс-связей в такой таблице.

3.2.4 Тесты на основе конечного автомата (Finite-state machine-based)

Строятся как комбинация тестов для всех состояний и переходов между состояниями, представленных в соответствующей модели (переходов и состояний приложения).

3.2.5 Тестирование на основе формальной спецификации (Testing from formal specification)

Для спецификации, определенных с использованием формального языка, возможно автоматически создавать и тесты для функциональных требований. В ряде случаев могут строиться на основе модели, являющейся частью спецификации, не использующей формального языка описания.

3.2.6 Случайное тестирование (Random testing)

В отличие от статистического тестирования (будет рассматриваться в 3.5.1 “Operational profile”), сами тесты генерируются случайным образом по списку заданного набора специфицированных характеристик.

3.3 Техники, ориентированные на код (*Code-based techniques*)

3.3.1 Тесты, базирующиеся на блок-схеме (Control-flow-based criteria)

Набор тестов строится исходя из покрытия всех условий и решений блок-схемы. В какой-то степени напоминает тесты на основе конечного автомата. Отличие – в источнике набора тестов.

Максимальная отдача от тестов на основе блок-схемы получается когда тесты покрывают различные пути блок-схемы – по-сути, сценарии потоков работ (поведения) тестируемой системы. Адекватность таких тестов оценивается как процент покрытия всех возможных путей блок-схемы.

3.3.2 Тесты на основе потоков данных (Data-flow-based criteria)

В данных тестах отслеживается полный жизненный цикл величин (переменных) – с момента рождения (определения), на всем протяжении использования, вплоть до уничтожения (неопределенности). В реальной практике используются нестрогое тестирование такого вида, ориентированное, например, только на проверку задания начальных значений всех переменных или всех вхождений переменных в код, с точки зрения их использования.

3.3.3 Ссылочные модели для тестирования, ориентированного на код (Reference models for code-based testing – flowgraph, call graph)

Является не столько техникой тестирования, сколько контролем структуры программы, представленной в виде дерева вызовов (например, sequence-диаграммы, определенной в нотации UML и построенной на основе анализа кода).

3.4 Тестирование, ориентированное на дефекты (Fault-based techniques)

Как это ни странно звучит на уровне названия таких техник тестирования, они, действительно, ориентированы на ошибки. Точнее – на специфические категории ошибок.

3.4.1 Предположение ошибок (Error guessing)

Направлены на обнаружение наиболее вероятных ошибок, предсказываемых, например, в результате анализа рисков.

3.4.2 Тестирование мутаций (Mutation testing)

Мутация – небольшое изменение тестируемой программы, произошедшее за счет частных синтаксических изменений кода (в частности, рефакторинга). Соответствующие тесты запускаются для оригинального и всех “мутировавших” вариантов тестируемой программы.

SWEBOK фокусируется на возможности, с помощью тестов, определять отличия между мутантами и исходным вариантом кода. Если такое отличие установлено, мутанта “убивают”, а тест считается успешным. Обычно, данный подход фокусируется на синтаксических ошибках, на практике отслеживаемых современными средами разработки и, конечно, компиляторами.

3.5 Техники, базирующиеся на условиях использования (Usage-based techniques)

3.5.1 Операционный профиль (Operational profile)

Базируется на условиях использования системы.

Тестирование для оценки надёжности системы должно проводиться в таком тестовом окружении, которое максимально приближено к реальным условиям работы системы. Результаты таких тестов позволяют оценить поведение системы в реальных условиях. Входные параметры тестов задаются на основе вероятностного распределения соответствующих параметров или их наборов при эксплуатации (входные данные могут прогнозироваться исходя из частоты возможных сценариев работы пользователей).

3.5.2 Тестирование, базирующееся на надёжности инженерного процесса (Software Reliability Engineered Testing)

Базируется на условиях разработки системы.

Соответствующие тесты (обозначаемые также аббревиатурой SRET) проектируются в контексте используемого процесса разработки и методик тестирования.

3.6 Техники, базирующиеся на природе приложения (Techniques based on the nature of the application)

Описанные выше техники могут применяться к любым типам программных систем. В то же время, в зависимости от технологической или архитектурной природы приложений, могут также применяться специфические техники, важные именно для заданного типа приложения. Среди таких техник:

- Объектно-ориентированное тестирование
- Компонентно-ориентированное тестирование
- Web-ориентированное тестирование
- Тестирование на соответствие протоколам
- Тестирование систем реального времени

3.7 Выбор и комбинация различных техник (Selecting and combining techniques)

3.7.1 Функциональное и структурное (Functional and structural)

Техники тестирования, строящиеся на основе спецификаций или кода часто называют функциональными или структурными, соответственно. Оба подхода не должны противопоставляться, но дополнять друг друга.

3.7.1 Определенное или случайное (Deterministic vs. random)

Обычно тесты можно распределить по данным группам на основе используемой политики выбора или определения входных параметров тестов.

4. Измерение результатов тестирования (Test-related measures)

Часто техники тестирования путают с целями тестирования. Степень покрытия тестами - не то же самое, что высокое качество тестируемой системы. Однако, эти вопросы связаны. Чем выше степень покрытия, тем больше вероятность обнаружения скрытых дефектов. Когда мы говорим о результатах тестирования, мы должны подходить к их оценке, как оценке самой тестируемой системы. Именно количественные оценки результатов тестирования (но не самих тестов, например, покрытия ими возможных сценариев работы системы) освещаются ниже. В свою очередь, метрики самих тестов или процесса тестирования, как такового – вопросы, рассматриваемые в областях знаний “Процессы программной инженерии” (Software Engineering Process) и “Управление инженерной деятельностью” (Software Engineering Management).

Измерения являются инструментом анализа качества. Измерение результатов тестирования касается оценки качества получаемого продукта – программной системы. История измерений демонстрирует прогресс достижения приемлемого качества. Такая история является инструментом менеджмента качества.

4.1 Оценка программ в процессе тестирования (Evaluation of the program under test, IEEE 982.1-98)

4.1.1 Измерения программ как часть планирования и разработки тестов (Program measurements to aid in planning and design testing)

Данные измерения могут базироваться на размере программ (например, в терминах количества строк кода или функциональных точек) или их структуре (например, с точки зрения оценки ее сложности в тех или иных архитектурных терминах). Структурные измерения могут также включать частоту обращений одних модулей программы к другим.

4.1.2 Типы дефектов, их классификация и статистика возникновения (Fault types, classification and statistics)

В литературе по тестированию встречается большое количество различных классификаций дефектов. Эффективность тестирования может быть достигнута в том случае, если мы понимаем какие типы дефектов могут быть найдены в процессе тестирования программной системы и как изменяется их частота во времени (подразумевая историческую перспективу развития системы, а не

её сбоев в процессе работы). Эта информация позволяет прогнозировать качество системы и помогает совершенствовать процесс разработки, в целом.

Стандарт IEEE 1044-93 классифицирует возможные программные “аномалии”.

4.1.3 Плотность дефектов (Fault density)

Тестируемая программа может оцениваться на основе подсчета и классификации найденных дефектов. Для каждого класса дефектов можно определить отношение между количеством соответствующих дефектов и размером программы (в терминах выбранных метрик оценки размера).

4.1.4 Жизненный цикл тестов, оценка надежности (Life test, reliability evaluation)

Статистические ожидания в отношении надежности программной системы (см. выше 2.2.5 “Достижение и оценка надежности”) могут использоваться для принятия решения о продолжении или прекращении (окончании) тестирования, исходя из заданных параметров приемлемого качества (например, плотности дефектов заданного класса).

4.1.5 Модели роста надежности (Reliability growth models)

Данные модели обеспечивают возможности прогнозирования надежности системы, базируясь на обнаруженных сбоях (см. выше 2.2.5). Модели такого рода разбиваются на две группы – по количеству сбоев (*failure-count*) и времени между сбоями (*time-between-failure*).

4.2 Оценка выполненных тестов (Evaluation of the tests performed)

4.2.1 Метрики покрытия/глубины тестирования (Coverage/thoroughness measures)

Критерии “адекватности” тестирования, в ряде случаев, требуют систематического выполнения тестов для определенных набора элементов программы, задаваемых ее архитектурой или спецификацией. Соответствующие метрики позволяют оценить степень охвата характеристик системы (например, процент различных тестируемых параметров производительности) и глубину их детализации (например, случайное тестирование параметров производительности или с учетом граничных значений и т.п.). Такие метрики помогают прогнозировать вероятностное достижение заданных параметров качества системы.

4.2.2 Введение искусственных дефектов (Fault seeding)

“Своими руками?! Никогда! ...” – такова, обычно, первая реакция на идею искусственного внесения дефектов, например, в программный код. На практике, этот подход помогает классифицировать возможные ошибки и следующие за ними сбои, применяя в дальнейшем полученные результаты для моделирования (пусть, часто, и интуитивного) возможных причин реальных сбоев, обнаруженных в процессе тестирования.

Безусловно, данная техника должна использоваться с максимальной осторожностью опытными специалистами, хорошо представляющими общую архитектуру тестируемой программной системы и разбирающимися во её внутренних связях.

4.2.3 Оценка мутаций (Mutation score)

Получаемое в процессе тестирования мутаций (см. выше 3.4.2) отношение “убитых” к общему числу сгенерированных мутантов помогает измерить эффективность выполняемых тестов. В силу специфики такой техники тестирования, количественные оценки мутаций имеют практическое значение только для определенных типов систем.

4.2.4 Сравнение и относительная эффективность различных техник тестирования (Comparison and relative effectiveness of different techniques)

Различные исследования в области тестирования связаны с попытками сравнения (с точки зрения достигаемого качества продукта) разных подходов к тестированию. Когда мы говорим об

“эффективности” тестирования надо чётко договориться, что именно мы подразумеваем под эффективностью, желательно, в количественном выражении. Возможные варианты интерпретации этого понятия – число тестов (данной техники), необходимых для обнаружения первого дефекта; отношение количества всех обнаруженных дефектов к дефектам, найденным с применением заданного подхода и т.п. Только обладая такого рода данными можно говорить о корректности сравнения и оценки эффективности.

5. Процесс тестирования (Test Process)

Концепции, стратегии, техники и измерения тестирования должны быть объединены в единый процесс тестирования как деятельности по обеспечению качества. Процесс тестирования поддерживает работы по тестированию и определяет “правила игры” для членов команды тестирования – от планирования тестов до оценки их результатов. Хотя, в большинстве современных методов разработки, в частности, гибких (agile) подходов, тестирование выходит на передний план и является одной из базовых практик, многостороннее тестирование и, тем более, прогнозирование на основе полученных результатов, часто подменяется отдельными работами в этой области, не позволяющими добиться необходимых параметров качества (что, кстати, ясно показывают уже упоминавшиеся результаты исследований Standish Group [Chaos, 2004]). Только в том случае, если тестирование рассматривать как один из важных процессов всей деятельности по созданию и поддержке программного обеспечения, можно добиться оценки стоимости соответствующих работ и, в конце концов, соблюсти те ограничения, которые определены для проекта.

5.1 Практические соображения (Practical considerations)

5.1.1 Программирование без персонаний (Attitudes/Egoless programming)

Очень важным компонентом успешного тестирования является *совместное стремление участников проекта обеспечить необходимое качество продукта*. Менеджеры играют ключевую роль в организации этой деятельности и на стадии разработки и в процессе сопровождения программных систем.

5.1.2 Руководства по тестированию (Test guides)

Работы по тестированию могут руководствоваться различными соображениями и критериями – от управления рисками до специфицированных сценариев работы программных систем. В любом случае, желательно, исходя из ресурсов, количественных оценок и других характеристик, обеспечить использование различных техник тестирования для многосторонней оценки и улучшения качества получаемого продукта.

5.1.3 Управление процессом тестирования (Test process management)

Работы по тестированию, ведущиеся на разных уровнях (см. выше 2. “Уровни тестирования”), должны быть организованы в единый (однозначно интерпретируемый) процесс, на основе учета 4 элементов и связанных с ними факторов: людей (в том числе, в контексте организационной структуры и культуры), инструментов, регламентов и количественных оценок (измерений). Стандарт жизненного цикла IEEE, ISO/IEC, ГОСТ Р 12207 не выделяет деятельность по тестированию в качестве самостоятельного процесса, однако, рассматривает соответствующие принципы работ по тестированию как неотъемлемую часть процессов жизненного цикла и сопровождения программных систем. В другом распространенном стандарте IEEE 1074 деятельность по тестированию также объединена с другими оценочными работами как интегральная часть полного жизненного цикла.

5.1.4 Документирование тестов и рабочего продукта (Test documentation and work products)

Документация – составная часть формализации процесса тестирования. Существует стандарт IEEE 829-98 “Standard for Software Test Documentation”, предоставляющий прекрасное описание тестовых документов, их связей между собой и с процессом тестирования. Среди таких документов могут быть:

- План тестирования
- Спецификация процедуры тестирования

- Спецификация тестов
- Лог тестов
- и др.

Документирование тестов, в случае его формального ведения, должно быть актуальным. В противном случае, как и любые другие документы, документация по тестированию ляжет “мертвым грузом”. В то же время, деятельность по тестированию, в случае отсутствия соответствующих регламентов и результатов (в том числе, исторических, для разных проектов), сложно поддается оценке для прогнозирования и, тем более, улучшению - в общем контексте улучшения процессов. Если компания-разработчик не ведет соответствующей документации по тестированию, говорить о сертификации или оценке по тем или иным моделям или стандартам (CMMI, ISO, SixSigma и т.п.) – просто не представляется возможным. А это уже вопрос доверия заказчиков, не имевших опыта работы с конкретной компанией-разработчиком.

5.1.5 Внутренние и независимые команды тестирования (Internal vs. independent test team)

Формализация процесса тестирования может включать и организационную формализацию команд(ы) тестирования. В нее могут входить как члены проектной команды, в частности, разрабатывающие код, так и внешние лица и группы. В идеале – желательно иметь как внутреннюю команду тестирования, так и внешнюю группу тестирования (обеспечения качества). Соответствующие организационные решения принимаются на основе стоимостных характеристик проекта, доступных ресурсов, анализа стоимости тестирования, как такового, организационной культуры и т.п.

5.1.6 Оценка стоимости и усилий, а также другие измерения процесса (Cost/effort estimation and other process measures)

Ряд метрик, связанных с оценкой ресурсов, необходимых для тестирования, как и оценка эффективности тестирования на разных этапах и уровнях, основывается на точке зрения и практиках менеджмента проекта (подразделения, компании...) и используется для оценки и улучшения (оптимизации) процесса тестирования. Разные техники, концепции и модели тестирования требуют разных затрат – по времени и необходимым ресурсам. Результат – стоимость тестирования, как затратная составляющая проекта. Понимание соответствия между стоимостью/усилиями, необходимыми для той или иной формы тестирования является обязательной частью современного управления проектами разработки программного обеспечения.

5.1.7 Окончание тестирования (Termination)

Очень важным аспектом тестирования является решение о том, в каком объеме тестирование достаточно и когда необходимо завершить процесс тестирования. Тщательные измерения, такие как достигнутое покрытие кода тестами или охват функциональности, безусловно, очень полезны. Однако, сами по себе они не могут определить критериев достаточности тестирования. Принятие решения об окончании тестирования также включает рассмотрение стоимости и рисков, связанных с потенциальными сбоями и нарушениями надёжности функционирования тестируемой программной системы. В то же время, стоимость самого тестирования также является одним из ограничений, на основе которых принимается решение о продолжении тех или иных связанных с проектом работ (с частности, тестирования) или об их прекращении. См. также 1.2.1 “Критерии отбора тестов/критерии адекватности тестов, правила прекращения тестирования”.

5.1.8 Повторное использование и шаблоны тестов (Test reuse and test patterns)

Доведение тестов до конца и обеспечение сопровождения программной системы необходимо каждый фрагмент системы тестировать систематическим образом, повторно используя наработанные тесты. Общий репозиторий тестовых активов должен находиться под контролем системы конфигурационного управления, с тем, чтобы любые изменения в требованиях или дизайне могли быть отражены в используемых наборах тестов, в том числе, с точки зрения их расширения новыми тестами, если этого требуют соответствующие изменения.

Шаблоны тестов конструируются на основе тестовых решений, наработанных для проверки определенных ситуаций или типовых фрагментов программных систем. Такие шаблоны должны

быть документированы с учетом повторного использования, включая прозрачные возможности их адаптации под специфику программных решений, к которым такие шаблоны применяются.

5.2 Тестовые работы (Test Activities)

Данная тема дает краткий обзор работ по тестированию. При этом подразумевается, что успешное управление тестовыми работами сильно зависит от процессов конфигурационного управления (Software Configuration Management), рассматриваемых позднее как самостоятельная область знаний.

5.2.1 Планирование (Planning)

Также как и другие аспекты управления проектами, работы по тестированию должно планироваться заранее. Как минимум, на уровне организации соответствующего процесса. Ключевые аспекты планирования тестовой деятельности включают:

- координацию персонала
- управление оборудованием и другими средствами, необходимыми для организации тестирования
- планирование обработки нежелательных результатов (т.е. является управлением определенными видами рисков)

В случае одновременной поддержки и сопровождения нескольких версий программной системы или нескольких систем, необходимо уделять особое внимание планированию времени, усилий и ресурсов, связанных с проведением работ по тестированию. Данная позиция перекликается с вопросами управления портфелями проектов с точки зрения общего управления проектами.

5.2.2 Генерация сценариев тестирования (Test-case generation)

Создание тестовых сценариев основывается на уровне и конкретных техниках тестирования. Тесты должны находиться под управлением системы конфигурационного управления и описывать ожидаемые результаты тестирования.

5.2.3 Разработка тестового окружения (Test environment development)

Используемое для тестирования окружение должно быть совместимо с инструментами программной инженерии (будут рассматриваться позднее как тема самостоятельной области знаний). Это окружение должно обеспечивать разработку и контроль тестовых сценариев, ведение журнала тестирования, и возможности восстановления ожидаемых и отслеживаемых результатов тестирования, самих сценариев, а также других активов тестирования.

5.2.4 Выполнение тестов (Execution)

Выполнение тестов должно содержать основные принципы ведения научного эксперимента:

- должны фиксироваться все работы и результаты процесса тестирования
- форма журналирования таких работ и их результатов должна быть такой, чтобы соответствующее содержание было понятно, однозначно интерпретируемой и повторяемо другими лицами (не теми, кто первоначально проводил тестирование)
- тестирование должно проводиться в соответствии с заданными и документированными процедурами
- тестирование должно производиться над однозначно идентифицируемой версией и конфигурацией программной системы

Ряд вопросов выполнения тестов и других работ по тестированию освещен в стандарте IEEE 1008-87.

5.2.5 Анализ результатов тестирования (Test results evaluation)

Для определения успешности тестов их результаты должны оцениваться, анализироваться. В большинстве случаев, “успешность” тестирования подразумевает, что тестируемое программное обеспечение функционирует так, как ожидалось и в процессе работы не приводит к

непредусмотренным последствиям. Не все такие последствия обязательно являются сбоями, они могут восприниматься как “помехи”. Однако, любое непредусмотренное поведение может стать источником сбоев при изменении конфигурации или условий функционирования системы, поэтому требуют внимания, как минимум, с точки зрения идентификации причин таких помех. Перед устранением обнаруженного сбоя, необходимо определить и зафиксировать те усилия, которые необходимы для анализа проблемы, отладки и устранения. Это позволит в дальнейшем обеспечить большую глубину измерений, а, соответственно, в перспективе, иметь возможность улучшения самого процесса тестирования. В тех случаях, когда результаты тестирования особенно важны, например, в силу критичности обнаруженного сбоя, может быть сформирована специальная группа анализа (review board).

5.2.6 Отчёты о проблемах/журнал тестирования (Problem reporting/Test log)

Во многих случаях, в процессе тестовой деятельности ведётся журнал тестирования, фиксирующий информацию о соответствующих работах: когда проводится тест, какой тест, кем проводится, для какой конфигурации программной системы (в терминах параметров и в терминах идентифицируемой версии контекста конфигурационного управления) и т.п. Неожиданные или некорректные результаты тестов могут записываться в специальной подсистеме ведения отчетности по сбоям (problem-reporting system, обеспечивая формирование базы данных, используемой для отладки, устранения проблем и дальнейшего тестирования. Кроме того, аномалии (помехи), которые нельзя идентифицировать как сбои, также могут фиксироваться в журнале и/или системе ведения отчетности по сбоям. В любом случае, документирование таких аномалий снижает риски процесса тестирования и помогает решать вопросы повышения надежности самой тестируемой системы. Отчёты по тестам могут являться входом для процесса управления изменениями и генерации запросов на изменения (change request) в рамках процессов конфигурационного управления (см. далее соответствующую область знаний “Software Configuration Management”).

5.2.7 Отслеживание дефектов (Defect tracking)

Сбои, обнаруженные в процессе тестирования, чаще всего порождаются дефектами и ошибками, присутствующими в тестируемой программной системе (также они могут быть следствием поведения операционного и/или тестового окружения). Такие дефекты могут (и, чаще всего, должны) анализироваться для определения момента и места первого появления данного дефекта в системе, какие типы ошибок стали причиной этих дефектов (например, плохо сформулированные требования, некорректный дизайн, утечки памяти и т.д.) и когда они могли бы быть обнаружены впервые. Вся эта информация используется для определения того, как может быть улучшен сам процесс тестирования и насколько критична необходимость таких улучшений.