Глава 4: Тестирование программного обеспечения

Из SWEBOK

Содержание

АКРОНИМЫ

API	Интерфейс прикладной программы
TDD	Разработка через тестирование
TTCN3	Нотация тестирования и управления тестированием, версия 3
XP	Экстремальное программирование

ВВЕДЕНИЕ

Тестирование программного обеспечения состоит из *динамической* проверки того, что программа обеспечивает *ожидаемое* поведение на *конечном* наборе тестовых случаев, соответствующим образом *выбранных* из обычно бесконечной области выполнения. В приведенном выше определении слова, выделенные курсивом, соответствуют ключевым вопросам описания области знаний по тестированию программного обеспечения (KA):

- *Динамический* : этот термин означает, что тестирование всегда подразумевает выполнение программы на выбранных входных данных. Чтобы быть точным, одного входного значения не

всегда достаточно для определения теста, поскольку сложная недетерминированная система может реагировать на одни и те же входные данные по-разному, в зависимости от состояния системы. Однако в этом КА термин «вход» будет сохранен с подразумеваемым соглашением о том, что его значение также включает указанное состояние ввода в тех случаях, для которых это важно. Статические методы отличаются от динамического тестирования и дополняют его. Статические методы описаны в КА качества программного обеспечения. Стоит отметить, что терминология неодинакова среди разных сообществ, и некоторые используют термин «тестирование» также в отношении статических методов.

- *Конечное* : даже в простых программах теоретически возможно так много тестовых случаев, что для выполнения исчерпывающего тестирования могут потребоваться месяцы или годы. Вот почему на практике полный набор тестов вообще можно считать бесконечным, а тестирование проводится на подмножестве всех возможных тестов, которое определяется риском и критериями приоритизации. Тестирование всегда подразумевает компромисс между ограниченными ресурсами и расписанием, с одной стороны, и изначально неограниченными требованиями к тестированию, с другой.
- *Выбрано* : многие предлагаемые методы тестирования существенно различаются тем, как выбирается набор тестов, и разработчики программного обеспечения должны знать, что разные критерии выбора могут давать совершенно разные степени эффективности. Как определить наиболее подходящий критерий отбора в данных условиях — сложная проблема; на практике применяются методы анализа рисков и опыт разработки программного обеспечения.
- *Ожидаемый* : Должна быть возможность, хотя и не всегда легко, решить, являются ли наблюдаемые результаты тестирования программы приемлемыми или нет; в противном случае усилия по тестированию бесполезны. Наблюдаемое поведение может быть проверено на соответствие потребностям пользователя (обычно это называется тестированием для проверки), соответствием спецификации (тестированием для проверки) или, возможно, ожидаемым поведением, исходя из неявных требований или ожиданий (см. Приемочные тесты в Требованиях к программному обеспечению КА).).

В последние годы взгляд на тестирование программного обеспечения превратился в конструктивный. Тестирование больше не рассматривается как действие, которое начинается только после завершения фазы кодирования с ограниченной целью обнаружения сбоев. Тестирование программного обеспечения является или должно распространяться на протяжении всего жизненного цикла разработки и сопровождения. Действительно, планирование тестирования программного обеспечения должно начинаться с самых ранних стадий процесса разработки требований к программному обеспечению, а планы и процедуры тестирования должны систематически и непрерывно разрабатываться и, возможно, уточняться по ходу разработки программного обеспечения. Эти мероприятия по планированию тестирования и разработке тестов обеспечивают полезную информацию для разработчиков программного обеспечения и помогают выявить потенциальные недостатки, такие как упущения/ противоречия в дизайне или упущения/неясности в документации.

Для многих организаций подход к качеству программного обеспечения заключается в предотвращении: очевидно, что гораздо лучше предотвращать проблемы, чем исправлять их. Таким образом, тестирование можно рассматривать как средство предоставления информации о функциональных возможностях и атрибутах качества программного обеспечения, а также для выявления ошибок в тех случаях, когда предотвращение ошибок не было эффективным. Возможно, это очевидно, но стоит признать, что программное обеспечение все еще может содержать ошибки, даже после завершения обширного тестирования. Сбои программного обеспечения, возникшие после доставки, устраняются корректирующим обслуживанием. Темы сопровождения программного обеспечения рассматриваются в документе КА по сопровождению программного обеспечения.

В КА качества программного обеспечения (см. Методы управления качеством программного обеспечения) методы управления качеством программного обеспечения подразделяются на статические методы (отсутствие выполнения кода) и динамические методы (исполнение кода). Обе категории полезны. Этот КА фокусируется на динамических техниках.

Тестирование программного обеспечения также связано с созданием программного обеспечения (см. Тестирование построения в КА построения программного обеспечения). В частности, модульное и интеграционное тестирование тесно связано с созданием программного обеспечения, если не является его частью.

РАЗБИВКА ТЕМ ДЛЯ ТЕСТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Разбивка тем КА по тестированию программного обеспечения показана на рис. 4.1. Более подробная разбивка представлена в Матрице тем и справочных материалов в конце этого КА.

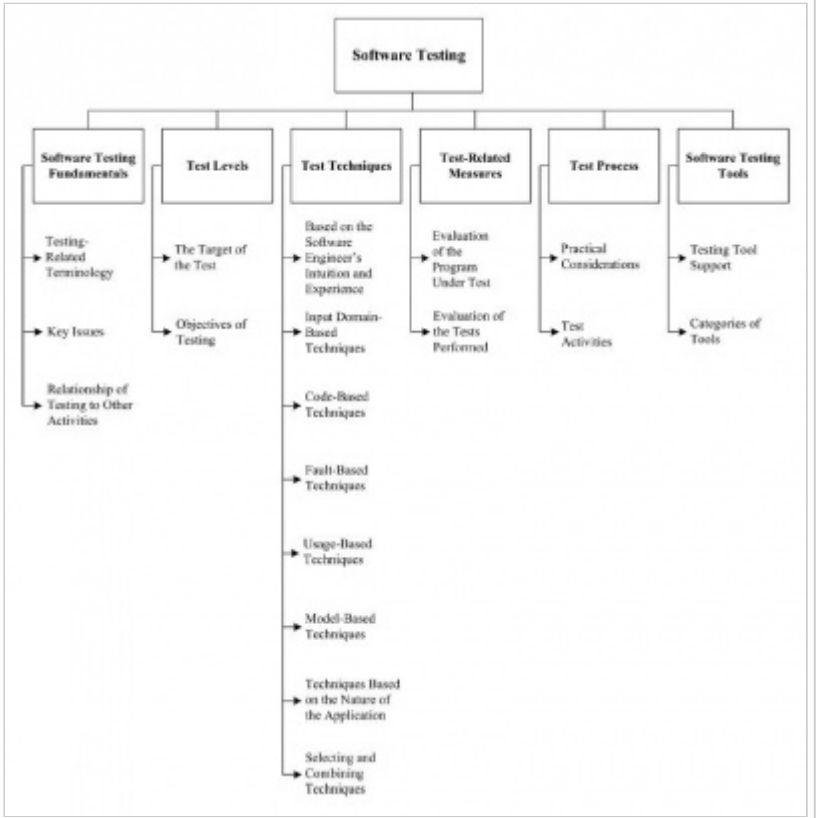

Рисунок 4.1: Разбивка тем для КА по тестированию программного обеспечения

Первая тема описывает основы тестирования программного обеспечения. Он охватывает основные определения в области тестирования программного обеспечения, основную терминологию и ключевые вопросы, а также взаимосвязь тестирования программного обеспечения с другими видами деятельности.

Вторая тема, «Уровни тестирования», состоит из двух (ортогональных) подтем: в первой подтеме перечислены уровни, на которые традиционно подразделяется тестирование больших программ, а во второй подтеме рассматривается тестирование на конкретные условия или свойства и упоминается как «Цели тестирования». Тестирование. Не все типы тестирования применимы к каждому программному продукту, и не перечислены все возможные типы.

Цель теста и цель теста вместе определяют, как идентифицируется набор тестов, как в отношении его согласованности — сколько тестов достаточно для достижения заявленной цели, так и в отношении его состава — какие тестовые наборы должны быть выбраны для достижения заявленной цели (хотя обычно «для достижения заявленной цели» остается имплицитным и ставится только первая часть из двух вышеприведенных вопросов, выделенных курсивом). Критерии ответа на первый вопрос называются критериями адекватности теста, а критерии ответа на второй вопрос — критериями выбора теста.

За последние несколько десятилетий было разработано несколько методов тестирования, и новые все еще предлагаются. Общепринятые методы рассматриваются в третьем разделе.

Меры, связанные с тестированием, рассматриваются в четвертом разделе, а вопросы, связанные с процессом тестирования, — в пятом. Наконец, инструменты тестирования программного обеспечения представлены в разделе шесть.

1 Основы тестирования программного обеспечения

1.1 Терминология, связанная с тестированием

1.1.1 Определения тестирования и соответствующей терминологии

[1 , c1, c2] [2 , c8]

Определения тестирования и связанной с тестированием терминологии приведены в цитируемых источниках и резюмированы следующим образом.

1.1.2 Неисправности и отказы

[1 , c1c5] [2 , c1]

В литературе по разработке программного обеспечения для описания неисправности используется множество терминов: в частности , *неисправность* , *сбой* и *ошибка* ., среди прочих. Эта терминология точно определена в [3, c2]. Важно четко различать причину неисправности (для которой здесь будет использоваться термин «неисправность») и нежелательный эффект, наблюдаемый в предоставляемых услугах системы (который будет называться отказом). Действительно, в программном обеспечении вполне могут быть сбои, которые никогда не проявляются как сбои (см. Теоретические и практические ограничения тестирования в разделе 1.2, Ключевые вопросы). Таким образом, тестирование может выявить сбои, но именно недостатки могут и должны быть устранены [3]. Более общий термин «дефект» может использоваться для обозначения неисправности или сбоя, когда различие не важно [3].

Однако следует признать, что причину отказа не всегда можно однозначно определить. Не существует теоретических критериев для окончательного определения неисправности, вызвавшей наблюдаемый отказ. Можно сказать, что это была ошибка, которую нужно было изменить, чтобы устранить ошибку, но другие модификации могли бы работать так же хорошо. Чтобы избежать двусмысленности, можно было бы ссылаться на входные данные, вызывающие отказ, вместо ошибок, то есть на те наборы входных данных, которые вызывают появление отказа.

1.2 Ключевые вопросы

1.2.1 Критерии выбора тестов/критерии адекватности тестов (правила остановки)

[1 , c1c14, c6c6, c12c7]

Критерий выбора тестов — это средство выбора тестовых наборов или определения того, что набор тестовых наборов достаточен для определенной цели. Критерии адекватности тестирования можно использовать для принятия решения о том, когда будет проведено или уже проведено достаточное тестирование [4] (см. Прекращение в разделе 5.1, Практические соображения).

1.2.2 Эффективность тестирования / цели тестирования

[1 , c11c4, c13c11]

Эффективность тестирования определяется путем анализа набора запусков программы. Выбор тестов для выполнения может определяться различными целями: только в свете преследуемой цели можно оценить эффективность набора тестов.

1.2.3 Тестирование для обнаружения дефектов

[1 , c1c14]

При тестировании обнаружения дефектов успешным считается тест, который приводит к сбою системы. Это сильно отличается от тестирования, чтобы продемонстрировать, что программное обеспечение соответствует своим спецификациям или другим желаемым свойствам, и в этом случае тестирование считается успешным, если в реалистичных тестовых примерах и тестовых средах не наблюдается сбоев.

1.2.4 Проблема оракула

[1 , c1c9, c9c7]

Оракул — это любой человек или механический агент, который решает, правильно ли вела себя программа в заданном тесте, и, соответственно, выносит вердикт «пройдено» или «не пройдено». Существует много различных видов оракулов; например, однозначные спецификации требований, поведенческие модели и аннотации кода. Автоматизация механизированных оракулов может быть сложной и дорогостоящей.

1.2.5 Теоретические и практические ограничения тестирования

[1 , c2c7]

Теория тестирования предостерегает от приписывания неоправданного уровня уверенности серии успешных тестов. К сожалению, большинство признанных результатов теории тестирования являются отрицательными, поскольку они утверждают, что тестирование никогда не может достичь, а не то, что на самом деле достигается. Наиболее известной цитатой в этом отношении является афоризм Дейкстры о том, что «тестирование программы может быть использовано для того, чтобы показать наличие ошибок,

но никогда не для того, чтобы показать их отсутствие» [5]. Очевидной причиной этого является то, что полное тестирование невозможно в реалистичном программном обеспечении. Из-за этого тестирование должно проводиться на основе риска [6, часть 1] и может рассматриваться как стратегия управления рисками.

1.2.6 Проблема недопустимых путей

[1 , c4c7]

Недопустимые пути — это пути потока управления, которые не могут быть реализованы никакими входными данными. Они представляют собой серьезную проблему при тестировании на основе путей, особенно при автоматическом получении тестовых входных данных для выполнения путей управления потоками.

1.2.7 Тестируемость

[1 , c17s2]

Термин «тестируемость программного обеспечения» имеет два связанных, но разных значения: с одной стороны, он относится к легкости, с которой может быть удовлетворен заданный критерий тестового покрытия; с другой стороны, он определяется как вероятность, возможно измеренная статистически, того, что набор тестовых случаев обнаружит отказ, если программное обеспечение неисправно. Оба значения важны.

1.3 Связь тестирования с другими видами деятельности

Тестирование программного обеспечения связано со статическими методами управления качеством программного обеспечения, доказательствами правильности, отладкой и построением программ, но отличается от них. Однако полезно рассмотреть тестирование с точки зрения аналитиков качества программного обеспечения и органов по сертификации.

- Тестирование и статические методы управления качеством программного обеспечения (см. Методы управления качеством программного обеспечения в KA качества программного обеспечения [1*, c12]).
- Тестирование в сравнении с доказательствами правильности и формальной проверкой (см. Модели и методы разработки программного обеспечения KA [1*, c17s2]).
- Тестирование и отладка (см. раздел «Тестирование конструкции» в KA «Конструкция программного обеспечения» и «Инструменты и методы отладки» в KA «Основы вычислений» [1*, c3s6]).
- Тестирование и построение программы (см. Тестирование построения в KA построения программного обеспечения [1*, c3s2]).

2 уровня тестирования

Тестирование программного обеспечения обычно выполняется на разных уровнях процессов разработки и сопровождения. Уровни можно различать по объекту тестирования, называемому целевым, или по цели, называемой задачей (уровня тестирования).

2.1 Цель теста

[1 , c1s13] [2 , c8s1]

Цель теста может быть разной: отдельный модуль, группа таких модулей (связанных по назначению, использованию, поведению или структуре) или целая система. Можно выделить три этапа тестирования: блок, интеграция и система. Эти три этапа тестирования не предполагают какой-либо модели процесса, и ни один из них не считается более важным, чем два других.

2.1.1 Модульное тестирование

[1 , c3] [2 , c8]

Модульное тестирование проверяет функционирование отдельных программных элементов, которые можно тестировать по отдельности. В зависимости от контекста это могут быть отдельные подпрограммы или более крупный компонент, состоящий из взаимосвязанных единиц. Обычно модульное тестирование

происходит с доступом к тестируемому коду и с поддержкой средств отладки. Программисты, написавшие код, обычно, но не всегда, проводят модульное тестирование.

2.1.2 Интеграционное тестирование

[1 , c7] [2 , c8]

Интеграционное тестирование — это процесс проверки взаимодействия между программными компонентами. Классические стратегии интеграционного тестирования, такие как «сверху вниз» и «снизу вверх», часто используются с иерархически структурированным программным обеспечением.

Современные стратегии систематической интеграции обычно ориентированы на архитектуру, которая включает в себя постепенную интеграцию программных компонентов или подсистем на основе определенных функциональных потоков. Интеграционное тестирование часто представляет собой непрерывную деятельность на каждом этапе разработки, во время которой инженеры-программисты абстрагируются от точек зрения более низкого уровня и концентрируются на перспективах уровня, на котором они интегрируются. Для любого, кроме небольшого, простого программного обеспечения, стратегии поэтапного интеграционного тестирования обычно предпочтительнее объединения всех компонентов одновременно, что часто называют тестированием «большого взрыва».

2.1.3 Тестирование системы

[1 , c8] [2 , c8]

Системное тестирование связано с тестированием поведения всей системы. Эффективное модульное и интеграционное тестирование позволит выявить многие дефекты программного обеспечения. Системное тестирование обычно считается подходящим для оценки нефункциональных системных требований, таких как безопасность, скорость, точность и надежность (см. «Функциональные и нефункциональные требования» в «Требованиях к программному обеспечению КА» и «Требования к качеству программного обеспечения» в «Качестве программного обеспечения КА»). Внешние интерфейсы к другим приложениям, утилитам, аппаратным устройствам или операционным средам также обычно оцениваются на этом уровне.

2.2 Цели тестирования

[1 , c1c7]

Тестирование проводится с учетом конкретных целей, которые формулируются более или менее явно и с разной степенью точности. Формулировка целей тестирования в точном количественном выражении способствует измерению и контролю процесса тестирования.

Тестирование может быть направлено на проверку различных свойств. Тестовые примеры могут быть разработаны для проверки правильности реализации функциональных спецификаций, что по-разному упоминается в литературе как тестирование на соответствие, тестирование правильности или функциональное тестирование. Однако могут быть протестированы и некоторые другие нефункциональные свойства, в том числе производительность, надежность и удобство использования, среди многих других (см. Модели и характеристики качества в КА качества программного обеспечения).

Другие важные цели тестирования включают, помимо прочего, измерение надежности, выявление уязвимостей в системе безопасности, оценку удобства использования и приемку программного обеспечения, для чего могут применяться различные подходы. Обратите внимание, что, как правило, цели теста варьируются в зависимости от цели теста; разные цели решаются на разных уровнях тестирования.

Перечисленные ниже подтемы наиболее часто упоминаются в литературе. Обратите внимание, что некоторые виды тестирования больше подходят для заказных программных пакетов — например, установочное тестирование, — а другие — для потребительских продуктов, таких как бета-тестирование.

2.2.1 Приемочные/квалификационные испытания

[1 , c1c7] [2 , c8c4]

Приемочное/квалификационное тестирование определяет, удовлетворяет ли система своим критериям приемки, обычно путем проверки желаемого поведения системы на соответствие требованиям заказчика. Таким образом, потребитель или представитель потребителя указывает или непосредственно предпринимает действия для проверки того, что их требования были выполнены, или, в случае потребительского продукта, что организация удовлетворила установленным требованиям для целевого рынка. Эта деятельность по тестированию может включать или не включать разработчиков системы.

2.2.2 Тестирование установки

[1 , c12c2]

Часто после завершения системного и приемочного тестирования программное обеспечение проверяется при установке в целевой среде. Инсталляционное тестирование можно рассматривать как системное тестирование, проводимое в операционной среде аппаратных конфигураций и других операционных ограничений. Процедуры установки также могут быть проверены.

2.2.3 Альфа- и бета-тестирование

[1 , c13s7, c16s6] [2 , c8s4]

Перед выпуском программного обеспечения оно иногда предоставляется небольшой группе потенциальных пользователей для пробного использования (альфа-тестирование) и/или более широкому кругу репрезентативных пользователей (бета-тестирование). Эти пользователи сообщают о проблемах с продуктом. Альфа- и бета-тестирование часто неконтролируемо и не всегда упоминается в плане тестирования.

2.2.4 Достижение и оценка надежности

[1 , c15] [2 , c15c2]

Тестирование повышает надежность за счет выявления и исправления ошибок. Кроме того, статистические показатели надежности могут быть получены путем случайной генерации тестовых случаев в соответствии с рабочим профилем программного обеспечения (см. Рабочий профиль в разделе 3.5, Методы, основанные на использовании). Последний подход называется операционным тестированием. Используя модели повышения надежности, обе цели могут быть достигнуты вместе [3] (см. «Жизненный тест», «Оценка надежности» в разделе 4.1 «Оценка тестируемой программы»).

2.2.5 Регрессионное тестирование

[1 , c8c11, c13c3]

Согласно [7], регрессионное тестирование — это «выборочное повторное тестирование системы или компонента для проверки того, что модификации не привели к непредвиденным последствиям и что система или компонент по-прежнему соответствуют заданным требованиям». На практике подход заключается в том, чтобы показать, что программное обеспечение по-прежнему проходит ранее пройденные тесты в наборе тестов (на самом деле это также иногда называют нерегрессионным тестированием). Для добавочной разработки цель регрессионного тестирования состоит в том, чтобы показать, что поведение программного обеспечения не изменяется при добавочных изменениях в программном обеспечении, за исключением случаев, когда это необходимо. В некоторых случаях необходимо найти компромисс между уверенностью, даваемой регрессионным тестированием каждый раз, когда вносятся изменения, и ресурсами, необходимыми для выполнения регрессионного тестирования, что может занять довольно много времени из-за большого количества тестов, которые могут быть выполнены. Регрессионное тестирование включает в себя выбор, минимизацию и/или определение приоритетов подмножества тестовых случаев в существующем наборе тестов [8]. Регрессионное тестирование может проводиться на каждом из уровней тестирования, описанных в разделе 2.1 «Цель теста», и может применяться к функциональному и нефункциональному тестированию.

2.2.6 Тестирование производительности

[1 , c8c6]

Тестирование производительности проверяет, соответствует ли программное обеспечение заданным требованиям к производительности, и оценивает характеристики производительности, например емкость и время отклика.

2.2.7 Тестирование безопасности

[1 , c8c3] [2 , c11c4]

Тестирование безопасности направлено на проверку того, что программное обеспечение защищено от внешних атак. В частности, тестирование безопасности проверяет конфиденциальность, целостность и доступность систем и их данных. Обычно тестирование безопасности включает в себя проверку на предмет неправомерного использования и злоупотребления программным обеспечением или системой (отрицательное тестирование).

2.2.8 Стресс-тестирование

[1 , c8c8]

Стресс-тестирование осуществляет программное обеспечение при максимальной расчетной нагрузке, а также за ее пределами, с целью определения поведенческих ограничений и проверки защитных механизмов в критических системах.

2.2.9 Параллельное тестирование

{{RightSection|[7] Стандарт IEEE/ISO/IEC 24765 определяет параллельное тестирование как «тестирование, при котором два или более вариантов программы выполняются с одними и теми же входными данными, выходные данные сравниваются, а ошибки анализируются в случае расхождения».

2.2.10 Тестирование восстановления

[1 , c14c2]

Тестирование восстановления направлено на проверку возможностей перезапуска программного обеспечения после системного сбоя или другой «катастрофы».

2.2.11 Тестирование интерфейса

[2 , c8s1.3] [9 , c4s4.5]

Дефекты интерфейса распространены в сложных системах. Тестирование интерфейса направлено на проверку того, правильно ли интерфейс компонентов обеспечивает правильный обмен данными и управляющей информацией. Обычно тестовые примеры генерируются из спецификации интерфейса. Конкретной целью тестирования интерфейса является имитация использования API приложениями конечного пользователя. Это включает в себя генерацию параметров вызовов API, настройку условий внешней среды и определение внутренних данных, влияющих на API.

2.2.12 Тестирование конфигурации

[1 , c8c5]

В тех случаях, когда программное обеспечение создано для обслуживания разных пользователей, тестирование конфигурации проверяет программное обеспечение в различных заданных конфигурациях.

2.2.13 Тестирование удобства использования и взаимодействия человека с компьютером

[10 , c6]

Основная задача тестирования удобства использования и взаимодействия человека с компьютером состоит в том, чтобы оценить, насколько легко конечным пользователям изучать и использовать программное обеспечение. В общем, это может включать в себя тестирование программных функций, которые поддерживают задачи пользователя, документацию, которая помогает пользователям, и способность системы восстанавливаться после пользовательских ошибок (см. Проектирование пользовательского интерфейса в Проекте программного обеспечения KA).

3 метода тестирования

Одна из целей тестирования — выявить как можно больше отказов. Для этого было разработано множество методов [6, часть 4]. Эти методы пытаются «сломать» программу, максимально систематически определяя входные данные, которые будут давать репрезентативное поведение программы; например, путем рассмотрения подклассов входной области, сценариев, состояний и потоков данных.

Представленная здесь классификация методов тестирования основана на том, как генерируются тесты: на основе интуиции и опыта инженера-программиста, спецификаций, структуры кода, реальных или воображаемых ошибок, которые необходимо обнаружить, прогнозируемого использования, моделей или характера приложения. . Одна категория связана с комбинированным использованием двух или более методов.

Иногда эти методы классифицируются как методы «белого ящика» (также называемые «стеклянным ящиком»), если тесты основаны на информации о том, как программное обеспечение было разработано или закодировано, или как «черный ящик», если тестовые примеры полагаются только на ввод/вывод. поведение программного обеспечения. Следующий список включает те методы тестирования, которые обычно используются, но некоторые специалисты-практики полагаются на одни методы больше, чем на другие.

3.1 На основе интуиции и опыта инженера-программиста

3.1.1 Специальный

Возможно, наиболее широко практикуемым методом является специальное тестирование: тесты создаются с опорой на навыки, интуицию и опыт инженера-программиста в работе с аналогичными программами. Специальное тестирование может быть полезно для выявления тестовых случаев, которые нелегко создать с помощью более формализованных методов.

3.1.2 Исследовательское тестирование

Исследовательское тестирование определяется как одновременное обучение, проектирование и выполнение тестов [6, часть 1]; то есть тесты не определяются заранее в установленном плане тестирования, а динамически разрабатываются, выполняются и модифицируются. Эффективность исследовательского тестирования зависит от знаний инженера-программиста, которые могут быть получены из различных источников: наблюдаемое поведение продукта во время тестирования, знакомство с приложением, платформой, процесс отказа, тип возможных ошибок и сбоев, риск, связанный с конкретный товар и так далее.

3.2 Технологии входных доменов

3.2.1 Эквивалентное разбиение

[1 , c9c4]

Разделение эквивалентности включает в себя разделение входной области на набор подмножеств (или эквивалентных классов) на основе указанного критерия или отношения. Этим критерием или отношением могут быть различные результаты вычислений, отношение, основанное на потоке управления или потоке данных, или различие между действительными входными данными, которые принимаются и обрабатываются системой, и недопустимыми входными данными, такими как значения вне диапазона, которые не принимаются. и должен генерировать сообщение об ошибке или инициировать обработку ошибки. Репрезентативный набор тестов (иногда только один) обычно берется из каждого класса эквивалентности.

3.2.2 Парное тестирование

[1 , c9c3]

Тестовые примеры получаются путем объединения интересных значений для каждой пары набора входных переменных вместо рассмотрения всех возможных комбинаций. Парное тестирование относится к комбинаторному тестированию, которое, как правило, также включает комбинации более

высокого уровня, чем пары: эти методы называются t-мудрыми, при этом рассматривается каждая возможная комбинация t входных переменных.

3.2.3 Анализ граничных значений

[1 , c9c5]

Тестовые случаи выбираются на границах входной области переменных или рядом с ними, исходя из того, что многие ошибки имеют тенденцию концентрироваться вблизи экстремальных значений входных данных. Расширением этого метода является тестирование надежности, при котором тестовые примеры также выбираются за пределами входной области переменных для проверки надежности программы при обработке неожиданных или ошибочных входных данных.

3.2.4 Выборочное тестирование

[1 , c9c7]

Тесты генерируются чисто случайным образом (не путать со статистическим тестированием из операционного профиля, как описано в операционном профиле в разделе 3.5). Эта форма тестирования относится к тестированию входной области, поскольку входная область должна быть известна, чтобы иметь возможность выбирать в ней случайные точки. Случайное тестирование обеспечивает относительно простой подход к автоматизации тестирования; недавно были предложены расширенные формы случайного тестирования, в которых случайная выборка исходных данных управляется другими критериями отбора входных данных [11]. Нечеткое тестирование или фаззинг — это особая форма случайного тестирования, направленная на взлом программного обеспечения; он чаще всего используется для тестирования безопасности.

3.3 Методы на основе кода

3.3.1 Критерии управления потоком

[1 , c4]

Критерии покрытия на основе потока управления нацелены на охват всех операторов, блоков операторов или определенных комбинаций операторов в программе. Самым сильным критерием, основанным на потоке управления, является тестирование путей, целью которого является выполнение всех путей потока управления от входа до выхода в графе потока управления программы. Поскольку исчерпывающее тестирование пути, как правило, невозможно из-за циклов, другие менее строгие критерии сосредоточены на покрытии путей, которые ограничивают количество итераций цикла, таких как покрытие операторов, покрытие ветвей и проверка условий/решений. Адекватность таких тестов измеряется в процентах; например, когда все ветки были выполнены тестами хотя бы один раз, было достигнуто 100% покрытие веток.

3.3.2 Критерии, основанные на потоке данных

[1 , c5]

При тестировании на основе потока данных граф потока управления аннотируется информацией о том, как программные переменные определяются, используются и уничтожаются (не определены). Самый сильный критерий, все пути использования определения, требует, чтобы для каждой переменной выполнялся каждый сегмент пути потока управления от определения этой переменной до использования этого определения. Чтобы уменьшить количество требуемых путей, используются более слабые стратегии, такие как все определения и все виды использования.

3.3.3 Эталонные модели для тестирования на основе кода

[1 , c4]

Хотя это и не метод сам по себе, управляющая структура программы может быть графически представлена с помощью блок-схемы для визуализации методов тестирования на основе кода. Потоковым графом называется ориентированный граф, узлы и дуги которого соответствуют элементам программы (см. Графы и деревья в книге «Математические основы КА»). Например, узлы могут представлять операторы или непрерывные последовательности операторов, а дуги могут представлять передачу управления между узлами.

3.4 Методы, основанные на неисправностях

[1 , c1c14]

С разной степенью формализации методы тестирования на основе ошибок разрабатывают тестовые примеры, специально предназначенные для выявления категорий вероятных или заранее определенных ошибок. Чтобы лучше сфокусировать генерацию или выбор тестового примера, можно ввести модель сбоя, которая классифицирует различные типы сбоев.

3.4.1 Предположение об ошибке

[1 , c9c8]

При угадывании ошибок тестовые примеры специально разрабатываются разработчиками программного обеспечения, которые пытаются предвидеть наиболее вероятные ошибки в данной программе. Хорошим источником информации является история ошибок, обнаруженных в более ранних проектах, а также опыт инженера-программиста.

3.4.2 Тестирование мутаций

[1 , c3s5]

Мутант — это слегка измененная версия тестируемой программы, отличающаяся от нее небольшим синтаксическим изменением. В каждом тестовом примере используются как исходная программа, так и все сгенерированные мутанты: если тестовый пример успешно идентифицирует разницу между программой и мутантом, последний считается «убитым». Первоначально задуманное как метод оценки тестовых наборов (см. раздел 4.2. Оценка выполненных тестов), мутационное тестирование также само по себе является критерием тестирования: либо тесты генерируются случайным образом до тех пор, пока не будет уничтожено достаточное количество мутантов, либо тесты специально разработаны для уничтожения выжившие мутанты. В последнем случае мутационное тестирование также может быть классифицировано как метод, основанный на коде. Лежащее в основе тестирования мутаций предположение, эффект сцепления, состоит в том, что поиск простых синтаксических ошибок, будут найдены более сложные, но реальные неисправности. Чтобы метод был эффективным, большое количество мутантов должно быть автоматически сгенерировано и выполнено систематическим образом [12].

3.5 Методы, основанные на использовании

3.5.1 Рабочий профиль

[1 , c15c5]

При тестировании для оценки надежности (также называемом эксплуатационным тестированием) тестовая среда максимально точно воспроизводит рабочую среду программного обеспечения или рабочий профиль. Цель состоит в том, чтобы на основе наблюдаемых результатов испытаний сделать вывод о будущей надежности программного обеспечения при фактическом использовании. Для этого входным данным присваиваются вероятности или профили в соответствии с их частотой появления в реальной работе. Рабочие профили можно использовать во время тестирования системы, чтобы направлять создание тестовых примеров, которые будут оценивать достижение целей надежности и проверять относительное использование и критичность различных функций, аналогичных тем, которые будут встречаться в операционной среде [3].

3.5.2 Эвристика наблюдения пользователя

[10 , c5, c7]

Принципы юзабилити могут служить руководством для обнаружения проблем в дизайне пользовательского интерфейса [10*, c1s4] (см. «Проектирование пользовательского интерфейса» в «Проектировании программного обеспечения»). Специализированные эвристики, также называемые методами проверки удобства использования, применяются для систематического наблюдения за использованием системы в контролируемых условиях, чтобы определить, насколько хорошо люди могут использовать систему и ее интерфейсы. Эвристика юзабилити включает в себя когнитивные пошаговые инструкции, анализ утверждений, полевые наблюдения, размышления вслух и даже непрямые подходы, такие как пользовательские анкеты и интервью.

3.6 Методы тестирования на основе моделей

Модель в этом контексте является абстрактным (формальным) представлением тестируемого программного обеспечения или его требований к программному обеспечению (см. Моделирование в Модели и методы разработки программного обеспечения KA). Тестирование на основе моделей используется для проверки требований, проверки их согласованности и создания тестовых сценариев, ориентированных на поведенческие аспекты программного обеспечения. Ключевыми компонентами тестирования на основе моделей являются [13]: нотация, используемая для представления модели программного обеспечения или его требований; модели рабочего процесса или аналогичные модели; стратегия тестирования или алгоритм, используемый для генерации тестового примера; вспомогательная инфраструктура для выполнения теста; и оценка результатов испытаний по сравнению с ожидаемыми результатами. Из-за сложности методов подходы к тестированию на основе моделей часто используются в сочетании со средствами автоматизации тестирования.

3.6.1 Таблицы решений

[1 , c9c6]

Таблицы решений представляют собой логические отношения между условиями (грубо говоря, входными данными) и действиями (грубо говоря, выходными данными). Тестовые случаи систематически выводятся путем рассмотрения всех возможных комбинаций условий и соответствующих им результирующих действий. Родственным методом является построение графика причинно-следственных связей [1*, c13s6].

3.6.2 Конечные автоматы

[1 , c10]

При моделировании программы как конечного автомата можно выбрать тесты, чтобы охватить состояния и переходы.

3.6.3 Формальные спецификации

[1 , c10c11] [2 , c15]

Формулировка спецификаций на формальном языке (см. Формальные методы в моделях разработки программного обеспечения и методах KA) позволяет автоматически выводить функциональные тестовые примеры и в то же время обеспечивает оракул для проверки результатов тестирования.

TTCN3 (Testing and Test Control Notation, версия 3) — это язык, разработанный для написания тестовых случаев. Нотация была задумана для конкретных нужд тестирования телекоммуникационных систем, поэтому она особенно подходит для тестирования сложных протоколов связи.

3.6.4 Модели рабочего процесса

[2 , c8c3.2, c19c3.1]

Модели рабочего процесса определяют последовательность действий, выполняемых людьми и/или программными приложениями, обычно представляемыми с помощью графических обозначений. Каждая последовательность действий составляет один рабочий процесс (также называемый сценарием). Должны быть протестированы как типичные, так и альтернативные рабочие процессы [6, часть 4]. Особое внимание уделяется ролям в спецификации рабочего процесса при тестировании бизнес-процессов.

3.7 Методы, основанные на характере приложения

Вышеуказанные методы применимы ко всем видам программного обеспечения. Дополнительные методы создания и выполнения тестов основаны на характере тестируемого программного обеспечения; Например,

- объектно-ориентированное программное обеспечение
- компонентное программное обеспечение
- веб-программное обеспечение
- параллельные программы

- программное обеспечение на основе протокола
- системы реального времени
- критически важные для безопасности системы
- сервис-ориентированное программное обеспечение
- программное обеспечение с открытым исходным кодом
- встроенное программное обеспечение

3.8 Выбор и комбинирование методов

3.8.1 Сочетание функционального и структурного

[1 , c9]

Методы тестирования на основе моделей и на основе кода часто противопоставляются функциональному и структурному тестированию. Эти два подхода к выбору тестов следует рассматривать не как альтернативу, а как дополнение; на самом деле они используют разные источники информации и, как было показано, выявляют разные виды проблем. Их можно использовать в комбинации, в зависимости от бюджетных соображений.

3.8.2 Детерминированный и случайный

[1 , c9c6]

Тестовые случаи могут быть выбраны детерминированным образом, в соответствии с одним из многих методов, или случайным образом взяты из некоторого распределения входных данных, как это обычно делается при тестировании надежности. Было проведено несколько аналитических и эмпирических сравнений для анализа условий, которые делают один подход более эффективным, чем другой.

4 Меры, связанные с тестированием

Иногда методы тестирования путают с целями тестирования. Методы тестирования можно рассматривать как средства, помогающие обеспечить достижение целей тестирования [6, часть 4]. Например, покрытие ветвей является популярным методом тестирования. Достижение определенного показателя охвата ветвей (например, 95% покрытия ветвей) не должно быть целью тестирования как такового: это способ повысить шансы обнаружения сбоев путем попытки систематически проверять каждую ветвь программы в каждой точке принятия решения. Во избежание подобных недоразумений следует проводить четкое различие между показателями, связанными с тестированием, которые обеспечивают оценку тестируемой программы на основе наблюдаемых выходных данных теста, и показателями, которые оценивают полноту набора тестов.

Измерение обычно считается фундаментальным для анализа качества. Измерение также может быть использовано для оптимизации планирования и выполнения тестов. Управление тестированием может использовать несколько различных показателей процесса для отслеживания прогресса. (См. раздел 5.1 «Практические соображения» для обсуждения показателей процесса тестирования, полезных для целей управления.)

4.1 Оценка тестируемой программы

4.1.1 Программные измерения, помогающие в планировании и разработке тестов

[9 , c11]

Меры, основанные на размере программного обеспечения (например, исходные строки кода или функциональный размер; см. Измерение требований в требованиях к программному обеспечению KA) или на структуре программы, могут использоваться для руководства тестированием. К структурным показателям также относятся измерения, определяющие частоту, с которой модули вызывают друг друга.

4.1.2 Типы неисправностей, классификация и статистика

[9 , c4]

Литература по тестированию богата классификациями и таксономиями неисправностей. Чтобы сделать тестирование более эффективным, важно знать, какие типы ошибок могут быть обнаружены в тестируемом программном обеспечении, а также относительную частоту, с которой эти ошибки возникали в прошлом. Эта информация может быть полезна для прогнозирования качества, а также для улучшения процесса (см. Характеристика дефектов в KA качества программного обеспечения).

4.1.3 Плотность неисправностей

[1 , c13c4] [9 , c4]

Тестируемую программу можно оценить, подсчитав обнаруженные ошибки как отношение между количеством найденных ошибок и размером программы.

4.1.4 Срок службы, оценка надежности

[1 , c15] [9 , c3]

Статистическая оценка надежности программного обеспечения, которую можно получить путем наблюдения за достигнутой надежностью, может использоваться для оценки программного продукта и принятия решения о том, можно ли прекратить тестирование (см. раздел 2.2, Достижение и оценка надежности).

4.1.5 Модели роста надежности

[1 , c15] [9 , c8]

Модели роста надежности позволяют прогнозировать надежность на основе сбоев. В целом они исходят из того, что, когда ошибки, вызвавшие наблюдаемые отказы, были устранены (хотя некоторые модели также допускают несовершенные исправления), оцениваемая надежность продукта в среднем демонстрирует тенденцию к увеличению. Существует множество опубликованных моделей роста надежности. Примечательно, что эти модели делятся на модели подсчета отказов и модели времени между отказами.

4.2 Оценка проведенных испытаний

4.2.1 Меры охвата/обстоятельности

[9 , c11]

Некоторые критерии адекватности тестирования требуют, чтобы тестовые примеры систематически использовали набор элементов, определенных в программе или спецификациях (см. раздел 3, Методы тестирования). Чтобы оценить тщательность выполненных тестов, инженеры-программисты могут отслеживать охваченные элементы, чтобы они могли динамически измерять соотношение между охваченными элементами и общим количеством. Например, можно измерить процент ветвей, охваченных графом потока программы, или процент выполняемых функциональных требований среди перечисленных в документе спецификаций. Критерии адекватности на основе кода требуют соответствующего оснащения тестируемой программы.

4.2.2 Подача ошибок

[1 , c2c5] [9 , c6]

При посеве ошибок некоторые ошибки искусственно вводятся в программу перед тестированием. Когда тесты будут выполнены, будут выявлены некоторые из этих засеянных ошибок, а также, возможно, некоторые ошибки, которые уже были. Теоретически, в зависимости от того, какие искусственные неисправности и в каком количестве они обнаружены, можно оценить эффективность тестирования и оценить оставшееся количество подлинных неисправностей. На практике статистики ставят под сомнение распределение и репрезентативность засеянных ошибок по сравнению с истинными ошибками и небольшой размер выборки, на котором основаны любые экстраполяции. Некоторые также утверждают, что этот метод следует использовать с большой осторожностью, поскольку внесение ошибок в программное обеспечение сопряжено с очевидным риском оставить их там.

4.2.3 Оценка мутации

[1 , c3s5]

При тестировании мутаций (см. Тестирование мутаций в разделе 3.4, Методы, основанные на ошибках) отношение убитых мутантов к общему количеству сгенерированных мутантов может быть мерой эффективности выполненного набора тестов.

4.2.4 Сравнение и относительная эффективность различных методов

Было проведено несколько исследований для сравнения относительной эффективности различных методов тестирования. Важно быть точным в отношении свойства, по которому оцениваются методы; какое, например, точное значение придается термину «эффективность»? Возможные интерпретации включают количество тестов, необходимых для обнаружения первого отказа, отношение количества ошибок, обнаруженных в ходе тестирования, ко всем ошибкам, обнаруженным во время и после тестирования, а также степень повышения надежности. Аналитические и эмпирические сравнения между различными методами были проведены в соответствии с каждым из указанных выше понятий эффективности.

5 Процесс тестирования

Концепции, стратегии, методы и меры тестирования необходимо интегрировать в определенный и контролируемый процесс. Процесс тестирования поддерживает действия по тестированию и предоставляет тестировщикам и группам тестирования руководство, от планирования тестирования до оценки результатов тестирования, таким образом, чтобы обеспечить уверенность в том, что цели тестирования будут достигнуты экономически эффективным способом.

5.1 Практические соображения

5.1.1 Отношения/программирование без эго

[1 , c16] [9 , c15]

Важным элементом успешного тестирования является совместное отношение к тестированию и деятельности по обеспечению качества. Менеджеры играют ключевую роль в обеспечении в целом положительного отношения к обнаружению и исправлению сбоев во время разработки и обслуживания программного обеспечения; например, путем преодоления мышления программистов об индивидуальном владении кодом и путем создания среды для совместной работы с командной ответственностью за аномалии в коде.

5.1.2 Руководства по тестированию

[1 , c12s1] [9 , c15s1]

Этапы тестирования могут руководствоваться различными целями — например, тестирование на основе рисков использует риски продукта для определения приоритетов и фокусировки стратегии тестирования, а тестирование на основе сценариев определяет тестовые наборы на основе заданных программных сценариев.

5.1.3 Управление процессом тестирования

[1 , c12] [9 , c15]

Действия по тестированию, проводимые на разных уровнях (см. раздел 2, Уровни тестирования), должны быть организованы — вместе с людьми, инструментами, политиками и мерами — в четко определенный процесс, являющийся неотъемлемой частью жизненного цикла.

5.1.4 Тестовая документация и рабочие продукты

[1 , c8s12] [9 , c4s5]

Документация является неотъемлемой частью формализации процесса тестирования [6, часть 3]. Тестовые документы могут включать, среди прочего, план тестирования, спецификацию дизайна тестирования, спецификацию процедуры тестирования, спецификацию тестового примера, журнал тестирования и отчет об инциденте тестирования. Тестируемое программное обеспечение документируется как элемент тестирования. Тестовая документация должна производиться и постоянно обновляться до того же уровня качества, что и другие типы документации в области разработки программного обеспечения. Тестовая документация также должна находиться под контролем управления

конфигурацией программного обеспечения (см. КА «Управление конфигурацией программного обеспечения»). Более того, тестовая документация включает рабочие продукты, которые могут предоставить материал для руководств пользователя и обучения пользователей.

5.1.5 Разработка через тестирование

[1 , c1c16]

Разработка через тестирование (TDD) возникла как одна из основных практик XP (экстремальное программирование) и состоит из написания модульных тестов перед написанием тестируемого кода (см. Agile Methods in the Software Engineering Models и Method KA). Таким образом, TDD разрабатывает тестовые примеры как заменитель документа спецификации требований к программному обеспечению, а не как независимую проверку того, что программное обеспечение правильно реализовало требования. TDD — это не стратегия тестирования, а практика, которая требует, чтобы разработчики программного обеспечения определяли и поддерживали модульные тесты; таким образом, это также может оказать положительное влияние на разработку спецификаций потребностей пользователей и требований к программному обеспечению.

5.1.6 Внутренняя и независимая группа тестирования

[1 , c16]

Формализация процесса тестирования может также включать формализацию организации группы тестирования. Группа тестирования может состоять из внутренних членов (то есть из проектной группы, вовлеченных или не вовлеченных в создание программного обеспечения), из внешних членов (в надежде внести беспристрастную, независимую точку зрения) или из внутренних и внешних членов. Соображения стоимости, графика, уровней зрелости вовлеченных организаций и критичности приложения могут определять решение.

5.1.7 Оценка затрат/усилий и процесс тестирования

[1 , c18s3] [9 , c5s7]

Несколько показателей, связанных с ресурсами, затрачиваемыми на тестирование, а также с относительной эффективностью выявления ошибок на различных этапах тестирования, используются менеджерами для контроля и улучшения процесса тестирования. Эти меры тестирования могут охватывать, среди прочего, такие аспекты, как количество заданных тестовых случаев, количество выполненных тестовых случаев, количество пройденных тестовых случаев и количество неудачных тестовых случаев.

Оценка отчетов фазы тестирования может быть объединена с анализом первопричин для оценки эффективности процесса тестирования в обнаружении ошибок как можно раньше. Такая оценка может быть связана с анализом рисков. Более того, ресурсы, которые стоит потратить на тестирование, должны быть соизмеримы с использованием/критичностью приложения: разные методики имеют разную стоимость и дают разный уровень уверенности в надежности продукта.

5.1.8 Прекращение

[9 , c10c14]

Необходимо принять решение о том, сколько тестов достаточно и когда этап тестирования может быть прекращен. Меры тщательности, такие как достигнутое покрытие кода или функциональное покрытие, а также оценки плотности ошибок или эксплуатационной надежности, обеспечивают полезную поддержку, но сами по себе недостаточны. Решение также включает соображения о затратах и рисках, связанных с возможными оставшимися отказами, в отличие от затрат, связанных с продолжением тестирования (см. Критерии выбора теста / Критерии адекватности теста в разделе 1.2, Ключевые вопросы).

5.1.9 Тестовое повторное использование и тестовые шаблоны

[9 , c2s5]

Для организованного и рентабельного проведения тестирования или обслуживания средства, используемые для тестирования каждой части программного обеспечения, должны систематически повторно использоваться. Репозиторий тестовых материалов должен находиться под контролем

управления конфигурацией программного обеспечения, чтобы изменения в требованиях или дизайне программного обеспечения могли отражаться в изменениях проводимых тестов.

Тестовые решения, принятые для тестирования некоторых типов приложений при определенных обстоятельствах, с мотивами принятых решений, образуют шаблон тестирования, который сам по себе может быть задокументирован для последующего повторного использования в аналогичных проектах.

5.2 Тестовые действия

Как показано в следующем описании, успешное управление действиями по тестированию сильно зависит от процесса управления конфигурацией программного обеспечения (см. KA управления конфигурацией программного обеспечения).

5.2.1 Планирование

[1 , cc12s1, c12s8]

Как и все другие аспекты управления проектом, деятельность по тестированию необходимо планировать. Ключевые аспекты планирования тестирования включают координацию персонала, наличие средств и оборудования для тестирования, создание и ведение всей документации, связанной с тестированием, а также планирование возможных нежелательных результатов. Если поддерживается более чем одна базовая версия программного обеспечения, то основное внимание при планировании уделяется времени и усилиям, необходимым для обеспечения правильной конфигурации тестовой среды.

5.2.2 Генерация тестового примера

[1 , c1c12c1, c12c3]

Генерация тестовых случаев основана на уровне тестирования, которое необходимо выполнить, и конкретных методах тестирования. Тестовые случаи должны находиться под контролем управления конфигурацией программного обеспечения и включать ожидаемые результаты для каждого теста.

5.2.3 Разработка тестовой среды

[1 , c12c6]

Среда, используемая для тестирования, должна быть совместима с другими принятыми инструментами разработки программного обеспечения. Это должно облегчить разработку и контроль тестовых случаев, а также регистрацию и восстановление ожидаемых результатов, сценариев и других материалов для тестирования.

5.2.4 Исполнение

[1 , c12c7]

Выполнение тестов должно отражать основной принцип научных экспериментов: все, что делается во время тестирования, должно быть выполнено и задокументировано достаточно четко, чтобы другой человек мог воспроизвести результаты. Следовательно, тестирование должно выполняться в соответствии с документированными процедурами с использованием четко определенной версии тестируемого программного обеспечения.

5.2.5 Оценка результатов испытаний

[9 , c15]

Результаты тестирования должны быть оценены, чтобы определить, было ли тестирование успешным. В большинстве случаев «успешно» означает, что программное обеспечение работало так, как ожидалось, и не имело серьезных непредвиденных результатов. Не все неожиданные результаты обязательно являются ошибками, но иногда они определяются как просто шум. Прежде чем ошибку можно будет устранить, необходимо провести анализ и отладку, чтобы изолировать, идентифицировать и описать ее. Когда результаты испытаний особенно важны, для их оценки может быть созвана официальная наблюдательная комиссия.

5.2.6 Отчет о проблемах / журнал испытаний

[1 , c12c9]

[1 , c13c9]

Действия по тестированию можно занести в журнал тестирования, чтобы определить, когда проводилось тестирование, кто его выполнял, какая конфигурация программного обеспечения использовалась, а также другую соответствующую идентификационную информацию. Неожиданные или неверные результаты тестирования могут быть зарегистрированы в системе отчетов о проблемах, данные для которой составляют основу для последующей отладки и исправления проблем, которые были замечены как сбои во время тестирования. Кроме того, аномалии, не классифицируемые как неисправности, могут быть задокументированы на случай, если позже они окажутся более серьезными, чем предполагалось на первый взгляд. Отчеты о тестировании также являются входными данными для процесса запроса управления изменениями (см. Управление конфигурацией программного обеспечения в КА управления конфигурацией программного обеспечения).

5.2.7 Отслеживание дефектов

[9 , c9]

Дефекты можно отслеживать и анализировать, чтобы определить, когда они были введены в программное обеспечение, почему они были созданы (например, плохо определенные требования, неправильное объявление переменных, утечка памяти, синтаксическая ошибка программирования) и когда они могли быть впервые обнаружены в программном обеспечении. программного обеспечения. Информация об отслеживании дефектов используется для определения того, какие аспекты тестирования программного обеспечения и других процессов нуждаются в улучшении и насколько эффективными были предыдущие подходы.

6 инструментов тестирования программного обеспечения

6.1 Поддержка инструментов тестирования

[1 , c12c11] [9 , c5]

Тестирование требует выполнения многих трудоемких задач, выполнения многочисленных программ и обработки большого объема информации. Соответствующие инструменты могут облегчить бремя канцелярских, утомительных операций и сделать их менее подверженными ошибкам. Сложные инструменты могут поддерживать разработку тестов и создание тестовых сценариев, что делает их более эффективными.

6.1.1 Выбор инструментов

[1 , c12c11]

Руководство для менеджеров и тестировщиков по выбору инструментов тестирования, которые будут наиболее полезными для их организации и процессов, является очень важной темой, поскольку выбор инструментов сильно влияет на эффективность и результативность тестирования. Выбор инструмента зависит от различных данных, таких как варианты разработки, цели оценки, средства выполнения и т. д. В общем, не может быть уникального инструмента, который удовлетворит определенные потребности, поэтому набор инструментов может быть подходящим выбором.

6.2 Категории инструментов

Мы классифицируем доступные инструменты в соответствии с их функциональностью:

- *Тестовые наборы* (драйверы, заглушки) [1*, c3s9] обеспечивают контролируемую среду, в которой можно запускать тесты и регистрировать результаты тестов. Для выполнения частей программы предоставляются драйверы и заглушки для имитации вызывающих и вызываемых модулей соответственно.
- *Генераторы тестов* [1*, c12s11] помогают в тестовых примерах генерации. Генерация может быть случайной, на основе пути, на основе модели или их сочетания.
- *Инструменты захвата/воспроизведения* [1*, c12s11] автоматически повторно выполняют или воспроизводят ранее выполненные тесты, которые имеют записанные входные и выходные данные (например, экраны).
- *Oracle/компараторы файлов/инструменты проверки утверждений* [1*, c9s7] помогают определить, является ли результат теста успешным или нет.

- *Анализаторы покрытия и инструменты* [1*, c4] работают вместе. Анализаторы покрытия оценивают, какие и сколько объектов потокового графа программы были проверены среди всех объектов, требуемых выбранным критерием покрытия тестами. Анализ может быть выполнен благодаря программным инструментам, которые вставляют в код записывающие зонды.
- *Трассировщики* [1*, c1s7] записывают историю путей выполнения программы.
- *Инструменты регрессионного тестирования* [1*, c12s16] поддерживают повторное выполнение набора тестов после изменения части программного обеспечения. Они также могут помочь выбрать подмножество тестов в соответствии с внесенными изменениями.
- *Инструменты оценки надежности* [9*, c8] поддерживают анализ результатов испытаний и графическую визуализацию для оценки мер, связанных с надежностью, в соответствии с выбранными моделями.

ИСПОЛЬЗОВАННАЯ ЛИТЕРАТУРА

[1] С. Найк и П. Трипати, *Тестирование программного обеспечения и обеспечение качества: теория и практика* , Wiley-Spektrum, 2008.

[2] И. Соммервиль, *Разработка программного обеспечения* , 9-е изд., Addison-Wesley, 2011.

[3] М. Р. Лю, изд., *Справочник по проектированию надежности программного обеспечения* , McGraw-Hill и IEEE Computer Society Press, 1996.

[4] Х. Чжу, П.А.В. Холл и Дж.Х.Р. Мэй, «Покрытие и адекватность модульных тестов программного обеспечения», *ACM Computing Surveys* , vol. 29, нет. 4, декабрь 1997 г., стр. 366-427.

[5] EW Dijkstra, «Заметки о структурном программировании», *TH-Report* , 70-WSE-03, Технологический университет, Эйндховен, 1970, http://www.cs.utexas.edu/users/EWD/ewd02xx/EWD249. PDF .

[6] *Проект стандарта ISO/IEC/IEEE P29119-1/DIS для разработки программного обеспечения и систем — Тестирование программного обеспечения — Часть 1: Понятия и определения* , ISO/IEC/IEEE, 2012 г.

[7] *ISO/IEC/IEEE 24765:2010 Системная и программная инженерия. Словарь* , ISO/IEC/IEEE, 2010.

[8] С. Ю и М. Харман, «Минимизация регрессионного тестирования, выбор и расстановка приоритетов: обзор», *Проверка и надежность тестирования программного обеспечения* , том. 22, нет. 2, март 2012 г., стр. 67–120.

[9] С. Х. Кан, *Метрики и модели в разработке качества программного обеспечения* , 2-е изд., Addison-Wesley, 2002.

[10] Дж. Нильсен, *Юзабилити-инжиниринг* , Морган Кауфманн, 1993.

[11] TY Chen и др., «Адаптивное случайное тестирование: искусство разнообразия тестовых случаев», *Journal of Systems and Software* , vol. 83, нет. 1, январь 2010 г., стр. 60-66.

[12] Ю. Цзя и М. Харман, "Анализ и обзор развития тестирования мутаций", *IEEE Trans. Программная инженерия* , том. 37, нет. 5, сен-окт. 2011, стр. 649-678.

[13] М. Уттинг и Б. Легард, *Практическое тестирование на основе моделей: инструментальный подход* , Морган Кауфманн, 2007.

- Последнее изменение этой страницы состоялось 24 августа 2015 г., в 19:23.