Лекция 6. Тестирование в течение жизненного цикла разработки ПО

Модели жизненного цикла разработки ПО

Модель жизненного цикла разработки программного обеспечения описывает виды активностей, выполняемых на каждом этапе процесса разработки программного обеспечения, а также то, как эти активности связаны друг с другом логически и хронологически. Существует несколько различных моделей жизненного цикла разработки программного обеспечения, каждый из которых требует различных подходов к тестированию.

Разработка и тестирование программного обеспечения

Важной частью работы тестировщика является знание распространенных моделей жизненного цикла разработки ПО, на основании которых происходит выбор соответствующих активностей по тестированию. Для любой модели жизненного цикла разработки существуют несколько показателей качественного тестирования:

● Для каждой активности разработки существует соответствующая активность тестирования.

● У каждого уровня тестирования есть цели, характерные для данного уровня.

● Анализ и проектирование тестов для определенного уровня тестирования начинаются на стадии соответствующих активностей разработки.

● Тестировщики должны участвовать в обсуждениях для определения и уточнения требований и дизайна, а также вовлекаться в рецензирование рабочих продуктов (например, требований, дизайна, пользовательских историй и т.п.), как только будут доступны первые их черновые версии. Независимо от того, какая модель жизненного цикла разработки выбрана, активности тестирования следует начинать на ранних стадиях жизненного цикла, придерживаясь принципа раннего тестирования. Эта программа обучения классифицирует распространенные модели жизненного цикла разработки следующим образом:

● Последовательные модели разработки

● Итерационные и инкрементальные модели разработки Последовательная модель разработки описывает процесс разработки программного обеспечения в качестве линейного, последовательного потока активностей. Это означает, что любую фазу процесса разработки следует начинать после того, как предыдущая фаза завершена. Теоретически фазы не

пересекаются друг с другом, но на практике бывает полезным получать раннюю обратную связь от следующей фазы. При работе по модели «Водопад», активности разработки (например, анализ требований, дизайн, написание кода, тестирование) выполняются каждая по окончании предыдущей. В данной модели активности тестирования осуществляются только после того, как все активности разработки выполнены. В отличие от модели «Водопад», V-модель интегрирует тестовый процесс на протяжении процесса разработки, реализуя принцип раннего тестирования. Более того, V-модель включает уровни тестирования, относящиеся к соответствующей фазе разработки, что также поддерживает раннее. В этой модели выполнение тестов, связанных с каждым уровнем тестирования, продолжается последовательно, но в некоторых случаях происходят пересечения. Результатом работы с применением последовательных моделей разработки является ПО, которое содержит полный набор функциональности, но для его поставки заказчику и пользователям зачастую уходят месяцы и даже годы разработки. Инкрементальная разработка подразумевает определение требований, дизайн, сборку и тестирование системы по частям. Это приводит к тому, что функциональность ПО растет постепенно. Размер таких функциональных приращений разнится в большую или меньшую сторону в зависимости от выбранных методов работы. Приращение функциональности может включать всего одно изменение в пользовательском интерфейсе или новую опцию запроса. Итеративная разработка используется в случае, когда разрабатываемый функционал должен быть определен, спроектирован, построен и протестирован в течение нескольких этапов, зачастую с фиксированной продолжительностью каждого из них. Итерации могут включать в себя как набор изменений функционала, реализованного в предыдущих итерациях, так и изменения всего разрабатываемого продукта в целом. На выходе каждой итерации получается рабочий программный продукт, который, по сути, является растущим набором общего функционала, и продолжается это до тех пор, пока не будет выпущена финальная версия этого продукта или разработка не будет остановлена. В качестве примера можно привести следующие методологии:

- RUP (Rational Unified Process): методология: каждая итерация может быть относительно долгой (например, от двух до трех месяцев), а приращения функциональности соответственно большими, к примеру, две или три группы связанных функциональностей.

- Скрам: каждая итерация может быть относительно короткой (например, часы, дни или несколько недель), а приращения функциональности соответственно небольшими, как, например, несколько улучшений и/или две или три новые функции.

● Канбан: осуществляется с использованием или без использования итераций фиксированной продолжительности, по завершению которых выпускается либо единственная доработка или функциональность, либо группа функциональностей, объединенных вместе.

● Спиральная модель разработки (прототипирование): подразумевает создание экспериментальных приращений, некоторые из которых могут быть значительно переработаны или даже отвергнуты в последующей работе по разработке. Компоненты или системы, разрабатываемые с использованием этих моделей, часто подразумевают пересечение и повторение уровней тестирования на протяжении процесса разработки. В идеале, каждая функциональность тестируется на нескольких уровнях тестирования по мере приближения к выпуску продукта. В некоторых случаях, команды используют непрерывную поставку или непрерывное развертывание, которые подразумевают существенную автоматизацию многих уровней тестирования в процессе поставки. Предпринимаемые усилия по разработке ПО с использованием этих методов, также включают концепцию самоорганизующихся команд, что может изменить как организацию работ по тестированию, так и взаимоотношения между тестировщиками и разработчиками. Применение этих методов в итоге формирует готовый продукт, который может быть выпущен конечным пользователям на основе постепенного приращения функциональности или более традиционным способом изготовления основного релиза. Независимо от того, поставлялись ли приращения конечным пользователям, важность регрессионного тестирования растет с ростом разрабатываемой системы. В отличие от результатов работы с применением последовательных моделей разработки, результатом работы с применением итеративных и инкрементальных моделей является ПО, готовое к использованию через недели или даже дни, но до поставки полного набора требований продукта могут уйти многие месяцы и даже годы. Для получения дополнительной информации о тестировании в контексте гибких методологий разработки предлагаем изучить ISTQB Базовый уровень. Тестировщик в сфере Гибких методологий, [Black 2017], [Crispin 2008] и [Gregory 2015].

Выбор модели жизненного цикла разработки в зависимости от ситуации Модели жизненного цикла разработки программного обеспечения должны выбираться и быть адаптированы к контексту характеристик проекта и продукта. Соответствующая модель жизненного цикла разработки должна быть выбрана и адаптирована на основе цели проекта, типа разрабатываемого продукта, бизнес-приоритетов (например, срок вывода продукта на рынок), и выявленных рисков продукта и проекта. Например, небольшую внутреннюю административную систему следует разрабатывать и тестировать иначе, нежели критически важную для безопасности систему, например, систему

управления тормозами автомобиля. Как другой пример, в некоторых случаях организационные и культурные аспекты могут осложнять общение между членами команды, что может препятствовать итеративной разработке. В зависимости от контекста проекта может потребоваться объединение или реорганизация уровней тестирования и/или тестовых активностей. Например, для интеграции готового коммерческого решения в более крупную систему покупатель этого решения может выполнить тестирование возможности взаимодействия на уровне системного интеграционного тестирования (например, интеграцию с инфраструктурой и другими системами) и на уровне приемочного тестирования (функциональное и/или нефункциональное, наряду с пользовательским приемочным и эксплуатационным приемочным тестированием). Кроме этого, сами модели жизненного цикла разработки могут сочетаться одна с другой. Например, V-модель может быть использована для разработки и тестирования систем серверного уровня и их интеграции, тогда как методология гибкой разработки может быть использована для разработки и тестирования пользовательского интерфейса (ПИ) и функциональности. На ранних стадиях проекта может быть использовано прототипирование, с переходом на инкрементальную модель разработки после завершения экспериментальной фазы. Системы с общим названием «Интернет вещей», которые состоят из множества различных объектов, таких как устройства, продукты и сервисы, часто разрабатываются с применением разных моделей жизненного цикла разработки для каждого объекта. Это ведет к определенной сложности для разработки версий таких систем. Кроме того, сильный акцент делается уже на поздние фазы жизненного цикла, после внедрения (например, фазы использования, обновления и вывода из эксплуатации).

Уровни тестирования

Уровни тестирования – это группы активностей тестирования, которые организуются и управляются как единое целое. **Каждый уровень тестирования** — это реализация процесса тестирования и исполняемого в отношении ПО, находящегося на конкретном уровне разработки, начиная с отдельных модулей и компонентов и заканчивая целыми системами или, где применимо, группами систем. Уровни тестирования связаны с другими активностями в рамках жизненного цикла разработки. Уровни, используемые в данной программе обучения, следующие:

- Компонентное тестирование
- Интеграционное тестирование
- Системное тестирование

● Приемочное тестирование Уровни тестирования характеризуются следующими признаками:

● Конкретные цели

● Базис тестирования, на который ссылаются для получения тестовых сценариев

● Объект тестирования (то есть то, что будет протестировано)

● Типичные дефекты и отказы

● Специфические подходы и зоны ответственности.

Для каждого уровня тестирования требуется подходящая среда тестирования. Например, при приемочном тестировании идеально использовать окружение, максимально приближенное к реальному. В тоже время при компонентном тестировании разработчики обычно используют собственную среду разработки.

Компонентное тестирование

Цели компонентного тестирования Компонентное тестирование (также известное как модульное тестирование) фокусируется на компонентах, которые могут быть проверены отдельно. Цели компонентного тестирования включают:

● Снижение риска

● Проверку, соответствует ли функциональное и нефункциональное поведение компонентов установленным проектным требованиям

● Укрепление уверенности в качестве компонента

● Обнаружение дефектов в компоненте

● Предотвращение пропуска дефектов на более высокие уровни тестирования

В некоторых случаях, особенно в инкрементных и итеративных моделях разработки (например, гибкой методологии разработки), где изменения кода происходят непрерывно, автоматизированные регрессионные компонентные тесты играют ключевую роль в создании уверенности в том, что изменения не повредили существующий функционал. Компонентное тестирование часто выполняется изолированно от остальной системы, в зависимости от модели жизненного цикла разработки программного обеспечения и системы, для которой могут потребоваться макеты разрабатываемых объектов, виртуализация служб, стенды, заглушки и драйверы. Компонентное тестирование может охватывать как функциональные (например,

правильность вычислений), так и нефункциональные характеристики (например, поиск утечек памяти) и структурные свойства (например, тестирование решений). Базис тестирования

Примеры рабочих продуктов, которые могут использоваться в качестве базиса тестирования для компонентного тестирования, включают:

● Детальный дизайн

● Код

● Модель данных

● Спецификации компонента

Объекты тестирования Типичными объектами для компонентного тестирования являются:

● Компоненты, модули

● Код и структуры данных

● Классы

● Модули БД Типичные дефекты и отказы

Примеры типичных дефектов и отказов при компонентном тестировании включают:

● Неправильная работа функциональности (например, не так, как описано в спецификации)

● Проблемы с потоками данных

● Неправильные код и логика

Дефекты обычно исправляются, как только они обнаруживаются, часто без оформления, в соответствующей системе управления дефектами. Однако, когда разработчики оформляют отчеты о дефектах, создается важная информация для анализа первопричин дефектов и улучшения процесса. Специфические подходы и зоны ответственности Компонентное тестирование обычно выполняется разработчиком, который написал код, но это как минимум требует доступа к тестируемому коду. Разработчики могут чередовать разработку компонентов с обнаружением и устранением дефектов. Разработчики часто пишут и выполняют тесты после написания кода для компонента. Тем не менее, написание автоматизированных тестовых сценариев для компонентов может предшествовать написанию кода приложения, особенно в методологии гибкой разработки. Например, рассмотрим разработку на основе тестов. Разработка на основе тестов является высоко итеративной, и основана на циклах разработки автоматизированных

тестов, затем построении и интеграции небольших фрагментов кода, а уже после – выполнении компонентного тестирования, исправлении проблем и рефакторинга кода. Этот процесс продолжается до тех пор, пока компонент не будет полностью собран и все компонентные тесты не пройдут успешно. Разработка на основе тестов является примером подхода «сначала тестирование». Хотя разработка на основе тестов берет начало в методологии экстремального программирования, она распространилась на другие формы гибких методологий разработки, а также на последовательные жизненные циклы (см. программу ISTQB Базовый уровень. Тестировщик в сфере Гибких методологий).

Интеграционное тестирование

Цели интеграционного тестирования Интеграционное тестирование фокусируется на взаимодействии между компонентами или системами. Цели интеграционного тестирования включают:

● Снижение риска

● Проверка, соответствует ли функциональное и нефункциональное поведение интерфейсов установленным проектным требованиям

● Повышение уверенности в качестве интерфейсов

● Обнаружение дефектов (которые могут быть в самих интерфейсах или внутри компонентов или систем)

● Предотвращение пропуска дефектов на более высокие уровни тестирования Как и при компонентном тестировании, в некоторых случаях автоматизированные регрессионные интеграционные тесты поддерживают уверенность в том, что изменения не повредили существующие интерфейсы, компоненты или системы.

В этой учебной программе описаны два разных уровня интеграционного тестирования, которые могут выполняться на тестовых объектах различного размера следующим образом:

● Компонентное интеграционное тестирование фокусируется на взаимодействиях и интерфейсах между интегрированными компонентами. Оно выполняется после компонентного и, как правило, автоматизируется. При итеративной и инкрементальной разработке компонентное интеграционное тестирование обычно является частью процесса непрерывной интеграции

● Системное интеграционное тестирование фокусируется на взаимодействиях и интерфейсах между системами, пакетами и микросервисами. Системное интеграционное тестирование также может охватывать взаимодействия и интерфейсы, предоставляемые сторонними

организациями (например, веб-сервисы). В этом случае организация-разработчик не контролирует внешние интерфейсы, которые могут создавать различные сложности для тестирования (например, проверяя, что блокирующие тесты дефекты устранены в коде сторонней организации, подготавливая тестовые среды и т. д.). Системное интеграционное тестирование может быть выполнено после системного тестирования или параллельно с выполняемыми активностями по системному тестированию (как в последовательной разработке, так и в итеративной и инкрементальной разработке).

Базис тестирования

Примеры рабочих продуктов, которые могут использоваться в качестве базиса тестирования для интеграционного тестирования, включают:

- Дизайн продукта и системы
- Диаграммы последовательности
- Спецификации интерфейса и протокола связи
- Сценарии использования системы
- Архитектура на уровне компонентов или системы
- Рабочие процессы
- Спецификации, описывающие внешние интерфейсы

Объекты тестирования

Типичными объектами тестирования при интеграционном тестировании являются:

- Подсистемы
- Базы данных
- Инфраструктура
- Интерфейсы
- Программные интерфейсы приложения (API)
- Микросервисы

Типичные дефекты и отказы

Примеры типичных дефектов и отказов при компонентном интеграционном тестировании включают:

● Некорректные данные, отсутствующие данные или неправильная кодировка данных

● Неверная последовательность или временные характеристики обращения к интерфейсам

● Несовместимость интерфейсов

● Сбои связи между компонентами

● Необработанные или неправильно обработанные сбои связи между компонентами

● Неправильные предположения о назначении, единицах или границах данных, передаваемых между компонентами

Примерами типичных дефектов и отказов для системного интеграционного тестирования являются:

● Несогласованные структуры сообщений между системами

● Некорректные данные, отсутствующие данные, или неправильная кодировка данных

● Несовместимость интерфейсов

● Сбои связи между системами

● Необработанные или неправильно обработанные сбои связи между системами

● Неправильные предположения о значении, единицах или границах данных, передаваемых между системами

● Несоблюдение обязательных правил безопасности

Специфические подходы и зоны ответственности Компонентные интеграционные и системные интеграционные тесты должны быть сосредоточены на интеграции как таковой. Например, если интегрировать модуль A с модулем B, тесты должны быть сосредоточены на связи между модулями, а не на функциональности отдельных модулей, поскольку они должны быть проверены во время компонентного тестирования. Если интегрировать систему X с системой Y, тесты должны быть сосредоточены на связи между системами, а не на функциях отдельных систем, поскольку они должны быть проверены во время системного тестирования. Для этого могут применяться функциональные, нефункциональные и структурные тесты. Компонентное интеграционное тестирование часто является обязанностью разработчиков. А системное интеграционного тестирование, как правило, обязанность тестировщиков. В идеале тестировщики, проводящие системное

интеграционного тестирование, должны понимать архитектуру системы и влиять на интеграционное планирование. Если интеграционные тесты и стратегия интеграции планируются до создания компонентов или систем, эти компоненты или системы могут быть построены в порядке, необходимом для наиболее эффективного тестирования. Стратегии систематической интеграции могут основываться на архитектуре системы (например, сверху вниз и снизу вверх), функциональных задачах, последовательностях обработки транзакций или другом аспекте работы системы или компонентов. Чтобы упростить локализацию дефектов и обнаруживать их на раннем этапе, интеграция должна обычно быть последовательной (т. е. небольшое количество дополнительных компонентов или систем за раз), а не методом «большого взрыва» (т. е. интеграция всех компонентов или систем сразу). Анализ рисков наиболее сложных интерфейсов может помочь определить фокус интеграционного тестирования. Чем больше объем интеграции, тем труднее становится выявлять дефекты конкретного компонента или системы, что может привести к увеличению риска и увеличению времени поиска неисправностей. Это одна из причин того, что непрерывная интеграция, когда программное обеспечение интегрируется методом «компонент за компонентом» (т.е. функциональная интеграция), стала обычной практикой. Такое непрерывное интегрирование часто подразумевает автоматическое регрессионное тестирование, в идеале на нескольких уровнях тестирования.

Системное тестирование

Цели системного тестирования Системное тестирование фокусируется на поведении и возможностях целой системы или продукта, часто учитывая сквозные задачи, которые может выполнять система, и нефункциональное поведение, которое она демонстрирует при выполнении этих задач. Цели системного тестирования включают:

● Снижение риска

● Проверка, соответствует ли функциональное и нефункциональное поведение системы установленным проектным требованиям, дизайну и спецификациям

● Проверка, что система реализована полностью и будет работать, как ожидалось

● Повышение уверенности в качестве системы в целом

● Обнаружение дефектов

● Предотвращение попадания дефектов на более высокие уровни тестирования или в среду эксплуатации.

Для определенных систем целью может быть проверка качества данных. Как и в случае компонентного и интеграционного тестирования, в некоторых случаях автоматическое регрессионное тестирование системы демонстрирует, что изменения не повредили существующий функционал или возможности конечных пользователей. Системное тестирование часто дает информацию, которая используется заинтересованными сторонами для принятия решения о релизе. Системное тестирование также может проверять выполнение законодательных или нормативных требований или стандартов. Тестовая среда для системного тестирования должна идеально соответствовать конечной целевой или эксплуатационной среде.

Базис тестирования

Примеры рабочих продуктов, которые могут использоваться в качестве базиса тестирования для системного тестирования, включают:

- Системные требования и требования к продукту (функциональные и нефункциональные)
- Отчеты об анализе рисков
- Сценарии использования
- Бизнес-потребности и пользовательские истории
- Модели поведения системы
- Диаграммы состояний
- Системные и пользовательские руководства

Объекты тестирования

Типичные объекты системного тестирования включают:

- Приложения
- Аппаратные / программные системы
- Тестируемая система
- Операционные системы
- Конфигурация системы и конфигурация данных

Типичные дефекты и отказы

Примеры типичных дефектов и сбоев при системном тестировании включают:

- Некорректные вычисления

● Некорректное или неожиданное функциональное или нефункциональное поведение системы

● Некорректное управление и/или передача данных внутри системы

● Невозможность правильно и полностью выполнить функциональные задачи конечными пользователями

● Неспособность системы работать правильно в среде эксплуатации

● Неспособность системы работать так, как описано в системных и пользовательских руководствах

Специфические подходы и зоны ответственности

Системное тестирование должно быть сосредоточено на общем (как функциональном, так и нефункциональном) поведении системы с точки зрения конечных пользователей. При системном тестировании следует использовать наиболее подходящие методы для конкретного аспекта тестируемой системы. Например, может быть применена таблица альтернатив, чтобы проверить, соответствует ли функциональное поведение бизнес-правилам. Системное тестирование часто проводит независимая группа тестировщиков. Ошибки в спецификациях (например, отсутствие пользовательских историй, неверно определенные бизнес-требования и т. д.) могут привести к отсутствию понимания или разногласиям насчет ожидаемого поведения системы. Такие ситуации могут вызывать ложно позитивные или ложно негативные результаты тестирования, которые отнимают время и снижают эффективность обнаружения дефектов. Раннее вовлечение тестировщиков в разработку пользовательских историй или в активности статического тестирования, таких как рецензирование, помогает снизить частоту возникновения таких ситуаций.

Приемочное тестирование

Цели приемочного тестирования

Приемочное тестирование, как и системное тестирование, обычно фокусируется на поведении и возможностях системы или продукта в целом. Цели приемочного тестирования включают:

● Продемонстрировать уверенность в качестве системы в целом

● Проверить, что система завершена и будет работать как ожидалось

● Проверить, соответствует ли функциональное и нефункциональное поведение системы установленным проектным требованиям

Приемочное тестирование может дать информацию для оценки готовности системы к развертыванию и использованию конечным

пользователем. Во время приемочного тестирования могут быть обнаружены дефекты, но их поиск зачастую не является целью, и обнаружение значительного количества дефектов во время приемочных испытаний может в некоторых случаях рассматриваться как основной риск проекта. Приемочное тестирование также может служить демонстрацией удовлетворения системы законодательным и нормативным требованиям или стандартам. Типичными формами приемочного тестирования являются:

- Пользовательское приемочное тестирование
- Эксплуатационное приемочное тестирование
- Контрактное и нормативное приемочное тестирование
- Альфа-тестирование и бета-тестирование

Пользовательское приемочное тестирование

Приемочное тестирование системы пользователями обычно сосредоточено на проверке пригодности использования системы предполагаемыми пользователями в реальной или моделируемой рабочей среде. Основная цель заключается в получении уверенности в том, что пользователи могут использовать систему для удовлетворения своих потребностей, а также в соответствии системы требованиям и ее способности выполнять поставленные бизнесом задачи с минимальными трудностями, затратами и рисками.

Эксплуатационное приемочное тестирование Приемочное тестирование системы сотрудниками или администраторами систем обычно проводятся в (имитируемой) среде эксплуатации. В тестах основное внимание уделяется эксплуатационным аспектам, которые могут включать:

- Тестирование резервного копирования и восстановления
- Установка, удаление и обновление
- Восстановление после полного отказа (краха) системы
- Управление пользователями
- Задачи сопровождения (обслуживания)
- Задачи загрузки и миграции данных
- Проверки уязвимостей
- Тестирование производительности

Основная цель эксплуатационного приемочного тестирования – получение уверенности в том, что операторы или системные администраторы смогут поддерживать работоспособность системы для пользователей в среде эксплуатации, даже в исключительных или сложных условиях

Контрактное и нормативное приемочное тестирование

Контрактное приемочное тестирование проводится в соответствии с указанными в контракте критериями приемки специализированного программного обеспечения. Критерии приемки должны определяться, когда стороны заключают контракт. Контрактное приемочное тестирование часто выполняется пользователями или независимой группой тестировщиков. Нормативное приемочное тестирование проводится в соответствии с любыми нормативами, которые должны соблюдаться, например, в отношении правительственных или юридических норм, а также норм безопасности. Нормативное приемочное тестирование часто выполняется пользователями или независимой группой тестировщиков, иногда с результатами, которые засвидетельствованы или проверяются регулирующими органами. Основная цель проведения контрактного и нормативного приемочного тестирования заключается в укреплении уверенности в том, что достигнуто соответствие контрактным или нормативным требованиям.

Альфа-тестирование и бета-тестирование

Альфа-тестирование и бета-тестирование обычно используются разработчиками готовых коммерческих решений, которые хотят получить обратную связь от потенциальных или существующих пользователей, клиентов и/или операторов до того, как программный продукт будет выставлен в коммерческую продажу. **Альфа-тестирование** проводится на мощностях компании разработчика, но не командой самого разработчика, а потенциальными или существующими клиентами и/или операторами или независимой группой тестирования. **Бетатестирование** проводится потенциальными или существующими клиентами и/или операторами на их собственных мощностях Бета-тестирование может проходить после альфа-тестирования или даже без предшествующего альфа-тестирования. Одной из целей альфа- и бета-тестирования является получение уверенности потенциальных или существующих клиентов и/или операторов в том, что они смогут использовать систему в нормальных, повседневных условиях и в эксплуатационных средах для достижения своих целей с минимальными трудностями, затратами и рисками. Другой целью может быть обнаружение дефектов, связанных с условиями и средой эксплуатации, в которых система будет использоваться, особенно когда команде разработчиков трудно воспроизвести эти условия и среды.

Базис тестирования

Примеры рабочих продуктов, которые могут использоваться в качестве базиса тестирования для любой формы приемочного тестирования, включают:

- Бизнес-процессы
- Пользовательские и бизнес-требования
- Нормативы, юридические контракты и стандарты
- Сценарии использования системы
- Системные требования
- Системная или пользовательская документация
- Процедуры установки программного обеспечения
- Отчеты анализа риска

Кроме того, в качестве тестового базиса разработки тестовых сценариев для эксплуатационного приемочного тестирования могут использоваться один или несколько следующих рабочих продуктов:

- Процедуры резервного копирования и восстановления
- Процедуры восстановления после полного отказа системы
- Нефункциональные требования
- Эксплуатационная документация
- Инструкции по развертыванию и установке
- Целевые показатели производительности
- Пакеты баз данных
- Стандарты или нормативы безопасности

Типичные объекты тестирования

Типичными объектами тестирования для любой формы приемочного тестирования являются:

- Тестируемая система
- Конфигурация системы и конфигурационные данные
- Бизнес-процессы для полностью интегрированной системы
- Восстановление системы и «горячего узла» (для непрерывности бизнес-процессов и аварийного восстановления)

● Операционные и эксплуатационные процессы

● Формы

● Отчеты

● Существующие и преобразованные производственные данные

Типичные дефекты и отказы

Примеры типичных дефектов для любой формы приемочного тестирования включают:

● Системные рабочие процессы не отвечают бизнес-требованиям или требованиям пользователей

● Бизнес-требования некорректно реализованы

● Система не соответствует контрактным и/или нормативным требованиям

● Нефункциональные сбои, такие как уязвимости в системе безопасности, недостаточная производительность при высоких нагрузках или неправильная работа на поддерживаемой платформе Специфические подходы и зоны ответственности Приемочное тестирование часто является обязанностью клиентов, бизнес-пользователей, владельцев продуктов или операторов системы, а также других заинтересованных сторон. В последовательном жизненном цикле разработки приемочное тестирование часто считается последним уровнем тестирования, но оно может проводиться и в другое время, например:

● Приемочное тестирование коммерческого готового программного обеспечения может проводиться, когда ПО установлено или интегрировано

● Приемочное тестирование нового функционального улучшения может производиться перед системным тестированием

При итеративной разработке проектные группы могут использовать различные формы приемочного тестирования как во время итерации, так и в ее конце, например, те, которые направлены на проверку новой функциональности по ее критериям приемки, и те, которые направлены на подтверждение того, что новая функциональность удовлетворяет потребностям пользователей. Кроме того, альфа-тесты и бета-тесты могут выполняться либо в конце каждой итерации, либо после завершения каждой итерации, либо после серии итераций. Пользовательское приемочное тестирование, эксплуатационное приемочное тестирование, а также контрактное и нормативное приемочное тестирование могут проводиться

аналогично либо по завершении каждой итерации, либо после завершения каждой итерации, либо после серии итераций.

Типы тестирования

Тип тестирования – это совокупность активностей тестирования, направленных на тестирование заданных характеристик системы или ее части, основываясь на конкретных целях.

К таким целям можно отнести следующие:

● Оценку функциональных характеристик качества системы, таких как полнота, корректность, целесообразность

● Оценку нефункциональных характеристик качества, таких как надежность, продуктивность работы, безопасность, совместимость и удобство использования

● Оценку правильности, полноты структуры или архитектуры компонента или системы, их соответствие спецификации

● Оценку влияния изменений, например, подтверждение того, что дефекты были исправлены (подтверждающее тестирование) и поиск непреднамеренных изменений в поведении, вызванных изменениями в программном обеспечении или окружении (регрессионное тестирование)

Функциональное тестирование

Функциональное тестирование системы включает тесты по оценке функций, которые должна выполнять система. Функциональные требования могут быть описаны в рабочих продуктах (требования, спецификация, бизнес-потребность, пользовательская история, сценарий использования) или в функциональной спецификации, а могут быть вообще не задокументированы. Функции системы дают ответ на вопрос «что делает система». Функциональные тесты должны выполняться на всех уровнях тестирования (например, тесты для компонентов системы могут основываться на спецификации компонента), поэтому фокус тестирования различается для каждого уровня.

Функциональное тестирование рассматривает поведение системы, поэтому для получения тестовых условий и тестовых сценариев могут быть использованы техники тестирования методом черного ящика. Оценить полноту функционального тестирования можно с использованием покрытия функциональности тестами. Покрытие функциональности – это мера, с которой конкретный тип элемента функциональности был охвачен тестами. Уровень покрытия вычисляется в процентах от общего количества элементов (или типов элементов). Например, используя трассируемость тестов и

соответствующих требований, можно вычислить процент требований, которые покрыты тестами, и выявить пробелы в покрытии. Проектирование и выполнение функциональных тестов может потребовать специальных навыков и знаний, относящихся к конкретной области бизнес-задачи, решаемой программным обеспечением (например, знания в области программного обеспечения геологического моделирования в нефте- и газодобывающей отрасли) или специфической предметной области (например, программное обеспечение компьютерных игр, которые предоставляют интерактивный развлекательный контент).

Нефункциональное тестирование

Нефункциональное тестирование системы выполняется для оценки таких характеристик системы и программного обеспечения, как удобство использования, производительность или безопасность. За классификацией характеристик качества программного обеспечения следует обратиться к стандарту ИСО (ISO/IEC 25010). **Нефункциональное тестирование** – это проверка того, «насколько хорошо работает система». Вопреки всеобщему заблуждению, нефункциональное тестирование может и, чаще всего, должно выполняться на всех уровнях тестирования, и как можно раньше. Несвоевременное обнаружение нефункциональных дефектов может быть угрозой успеха всего проекта. Для получения тестовых условий и тестовых сценариев в нефункциональном тестировании могут использоваться техники тестирования методом черного ящика . Например, анализ граничных значений может использоваться для определения пиковых нагрузок при тестировании производительности. Полноту нефункционального тестирования можно оценить через нефункциональное покрытие. **Нефункциональное покрытие** – это степень, с которой какая-либо нефункциональная характеристика покрыта тестами, которая выражается в процентном соотношении покрытых тестами характеристик к их общему числу. Например, используя трассируемость тестов к соответствующим им поддерживаемым устройствам для мобильного приложения, можно вычислить процент поддерживаемых устройств, используемых при тестировании совместимости, тем самым определив потенциальные пробелы в покрытии. Проектирование и выполнение нефункциональных тестов может потребовать специальных навыков и знаний, например, знаний о слабых сторонах, свойственных той или иной структуре проекта или технологии (это могут быть уязвимости безопасности, связанные с конкретным языком программирования), знание специфической базы пользователей (например, представления о пользователях управленческих систем в медицинских учреждениях).

Тестирование методом белого ящика

Тестирование методом белого ящика основывается на внутренней структуре системы или ее реализации.

Под внутренней структурой подразумевается программный код, архитектура, принципы работы и/или потоки данных внутри системы. Полноту тестирования методом белого ящика оценивают с помощью структурного покрытия.

Структурное покрытие – это мера, с которой какой-либо тип структурного элемента был покрыт тестами. Структурное покрытие измеряется в процентах от общего количества элементов, охваченного тестами. На уровне компонентного тестирования покрытие кода измеряется в процентах покрытия тестами кода компонента и может быть вычислено с точки зрения различных аспектов кода (элементов покрытия), таких как процент исполняемых операторов (покрытие операторов), тестируемых в компоненте, или процент покрытия разветвлений (покрытие альтернатив). Эти типы покрытия в совокупности называются покрытием кода.

На уровне интеграционного тестирования компонентов тестирование методом белого ящика может основываться на архитектуре системы, например, на взаимодействии между компонентами. В этом случае структурное покрытие измеряется в процентах от количества интерфейсов, задействованных в тестах. Проектирование и выполнение тестирования методом белого ящика может потребовать специальных навыков и знаний, таких как знание о методах сборки приложения (например, при использовании инструментов покрытия кода), о том, как хранятся данные (для оценки оптимальности запросов к базам данных) и умений пользоваться и интерпретировать результаты работы инструментов покрытия кода.

Тестирование, связанное с изменениями

Когда в систему вносятся изменения, выполненные для исправления дефекта, либо из-за новой или изменяющейся функциональности, необходимо провести тестирование, чтобы подтвердить, что изменения исправили дефект или что функциональность правильно реализована и изменения не вызвали каких-либо непредвиденных неблагоприятных последствий.

● **Подтверждающее тестирование**: после того как дефект исправлен, программное обеспечение может быть протестировано с использованием всех тех же тестовых сценариев, которые завершились с ошибкой из-за найденного дефекта. Эти тестовые сценарии должны быть повторно выполнены на новой версии программного обеспечения. Как минимум, на новой версии программного обеспечения должны быть повторно выполнены шаги по воспроизведению сбоев, вызванных дефектом. Целью подтверждающего

тестирования является удостоверение в том, что найденный дефект был исправлен.

● **Регрессионное тестирование**: может случиться так, что изменение, внесенное в одну часть кода, будь то исправление или что-либо другое, может случайно повлиять на поведение других частей кода, будь то внутри одного и того же компонента, в других компонентах одной и той же системы или даже в других системах. Доработки могут включать изменения в среде, такие как новая версия операционной системы или системы управления базами данных. Такие непреднамеренные побочные эффекты называются регрессиями. Регрессионное тестирование включает выполнение тестов для обнаружения таких непреднамеренных побочных эффектов. Подтверждающее тестирование и регрессионное тестирование проводятся на всех уровнях тестирования. Особенно в итеративных и инкрементальных жизненных циклах разработки (например, гибкая методология разработки), новые функции, изменения существующих функций и рефакторинг кода приводят к частым изменениям кода, что также требует тестирования, связанного с изменениями. Из-за меняющегося характера системы подтверждающее тестирование и регрессионное тестирование очень важны. Это особенно актуально для «Интернета вещей», где отдельные объекты (например, устройства) часто обновляются или заменяются.

Регрессионные тесты выполняются много раз и обычно проходят медленно, поэтому регрессионное тестирование — это серьезный кандидат на автоматизацию.

Типы и уровни тестирования

Любой из описанных выше типов тестирования можно выполнять на любом уровне тестирования. Для иллюстрации ниже будут даны примеры функциональных, нефункциональных, структурных тестов и тестов, связанных с изменениями для всех уровней тестирования банковского приложения, начиная с функциональных тестов:

● Компонентное тестирование: тесты разрабатываются на основе того, как компонент должен вычислять сложные проценты

● Тестирование интеграции компонентов: тесты разрабатываются на основе того, как информация учетной записи, созданная в пользовательском интерфейсе, передается модулю бизнес-логики

● Системное тестирование: тесты разрабатываются на основе того, как владельцы счетов могут подать заявку на кредитную линию на своих расчетных счетах

● Системное интеграционное тестирование: тесты разрабатываются на основе того, как система использует внешний микросервис для проверки кредитного рейтинга владельца счета

● Приемочное тестирование: тесты разрабатываются на основе того, как банкир обрабатывает одобрение или отклонение заявки на получение кредита Примерами нефункциональных тестов, являются:

● Компонентное тестирование: разрабатываются тесты производительности, предназначенные для оценки количества циклов ЦП, необходимых для выполнения расчета сложных процентов

● Тестирование интеграции компонентов: разрабатываются тесты на защищенность от уязвимостей при переполнении буфера данными, переданными из пользовательского интерфейса в модуль бизнес-логики

● Системное тестирование: разрабатываются тесты на переносимость, предназначенные для проверки того, работает ли слой представления во всех поддерживаемых браузерах и мобильных устройствах

● Системное интеграционное тестирование: разрабатываются тесты надежности, предназначенные для оценки работоспособности системы, если микросервис кредитной оценки не отвечает

● Приемочное тестирование: разрабатываются тесты практичности, предназначенные для оценки доступности интерфейса обработки банковских процессов для людей с ограниченными возможностями Примерами тестов методом белого ящика являются:

● Компонентное тестирование: разрабатываются тесты, предназначенные для обеспечения полного покрытия операторов и альтернатив для всех компонентов, которые выполняют финансовые расчеты
● Тестирование интеграции компонентов: разрабатываются тесты, предназначенные для того, чтобы проверить, как каждый экран в интерфейсе браузера передает данные на следующий экран и в модуль бизнес-логики

● Системное тестирование: разрабатываются тесты, предназначенные для покрытия последовательности веб-страниц, которые могут возникать при работе с кредитной линией

● Системное интеграционное тестирование: разрабатываются тесты, предназначенные для исполнения всех возможных типов запросов, отправляемых на микросервис кредитной оценки

● Приемочное тестирование: разрабатываются тесты, предназначенные для охвата всех поддерживаемых файловых структур финансовых данных и

диапазонов значений для банковских переводов. Наконец, примерами тестов, связанных с изменением, являются

- Компонентное тестирование: для каждого компонента разрабатываются автоматические регрессионные тесты и включаются в среду непрерывной интеграции

- Тестирование интеграции компонентов: разрабатываются тесты, для подтверждения исправлений дефектов, связанных со связями, по мере того как исправления проверяются в репозитории кода

- Системное тестирование: повторно выполняются все тесты для конкретного рабочего процесса, если любой из аспектов этого процесса меняется

- Системное интеграционное тестирование: ежедневно повторяются тесты для приложения, взаимодействующего с микросервисом кредитной оценки, в рамках непрерывного развертывания этого микросервиса

- Приемочное тестирование: все ранее неудавшиеся тесты повторно выполняются после устранения дефекта, обнаруженного при приемочных испытаниях. Хотя в этом разделе приведены примеры каждого типа теста на каждом уровне, не обязательно иметь каждый тип теста, представленный на каждом уровне для всех разрабатываемых систем. Тем не менее, важно выполнять соответствующие типы тестов на каждом уровне, особенно на самом раннем этапе.

Тестирование в период сопровождения.

После развертывания в эксплуатационных средах программное обеспечение и системы необходимо поддерживать. Практически невозможно избежать различных изменений в поставляемом программном обеспечении и системах, которые либо должны исправлять дефекты, обнаруженные при промышленной эксплуатации, либо добавлять новые функции, либо удалять или изменять уже существующий функционал. Сопровождение также необходимо для сохранения или улучшения нефункциональных характеристик качества компонента или системы в течение всего срока службы, особенно эффективности производительности, совместимости, надежности, безопасности и переносимости. В процессе сопровождения системы после внесения в нее изменений должно выполняться и тестирование как с целью оценки успеха, ради которого были сделаны изменения, так и для проверки возможных побочных эффектов (например, регрессий) в частях системы, которые остаются неизменными (что обычно является большей частью системы). Тестирование в период сопровождения фокусируется на тестировании изменений в системе, а также на тестировании неизмененных

частей, на которые могли повлиять изменения. Сопровождение может включать запланированные релизы и незапланированные релизы (срочные исправления). Релиз на этапе сопровождения системы может потребовать проведения тестирования в период сопровождения на нескольких уровнях тестирования с использованием различных типов тестов в зависимости от его объема.

Объем тестирования в период сопровождения зависит от:

● Степени риска изменения, например, степени, в которой измененная область программного обеспечения взаимодействует с другими компонентами или системами

● Размера существующей системы

● Величины внесенных изменений

Необходимые условия для тестирования в период сопровождения существует несколько причин, почему требуется сопровождение программного обеспечения и, соответственно, тестирование в период сопровождения как для запланированных, так и для незапланированных изменений. Мы можем классифицировать необходимые условия для тестирования в период сопровождения следующим образом:

● Модификации, такие как запланированные улучшения (например, базирующиеся на графике выпуска обновлений), корректирующие и аварийные изменения, изменения среды эксплуатации (например, запланированные обновления операционной системы или базы данных), обновления коммерческого готового программного обеспечения и исправления дефектов и уязвимостей

● Миграция, например, с одной платформы на другую, которая может потребовать проведения эксплуатационных тестов новой среды, а также измененного программного обеспечения или тестов преобразования данных, когда данные будут перенесены в поддерживаемую систему из другого приложения

● Снятие с эксплуатации, например, когда заканчивается жизненный цикл приложения. Когда приложение или система снимаются с эксплуатации, то может потребоваться тестирование переноса или архивирования данных, если требуются длительные периоды хранения данных. Также может потребоваться тестирование процедур восстановления после архивирования для случаев длительных периодов хранения. Кроме того, может потребоваться регрессионное тестирование, чтобы гарантировать, что все функциональные возможности, которые остаются в эксплуатации, все еще работают. Для «Интернета вещей» тестирование в период сопровождения может быть

вызвано внедрением в общую систему совершенно новых или модифицированных элементов, таких как аппаратные устройства и программные службы. Тестирование в период сопровождения для таких систем делает особый акцент на интеграционном тестировании на разных уровнях (например, сетевом уровне, уровне приложений) и на аспектах безопасности, в частности тех, которые касаются персональных данных.

Анализ влияния для тестирования в период сопровождения

Анализ влияния оценивает изменения, которые были сделаны для обновленной эксплуатируемой версии для определения предполагаемых последствий, а также ожидаемых и возможных побочных эффектов изменения, и определяет области в системе, на которые может повлиять изменение. Анализ влияния также может помочь определить воздействие изменения на существующие тесты. Побочные эффекты и затронутые области в системе необходимо проверить на регрессии, возможно, после обновления любых существующих тестов, затронутых изменением. Анализ влияния может быть выполнен до внесения изменений, чтобы решить, следует ли вносить изменения, исходя из возможных последствий в других областях системы. Анализ влияния может быть затруднен если:

● Спецификации (например, бизнес-требования, истории пользователей, архитектура) устарели или отсутствуют

● Тестовые сценарии не задокументированы или устарели

● Двунаправленная прослеживаемость между тестами и базисом тестирования не поддерживалась

● Инструментарий поддержки устарел или вовсе отсутствует

● Участники процесса не имеют основополагающих знаний о системе

● Во время разработки уделялось недостаточно внимания сопровождаемости программного обеспечения