Russian Software Testing Qualifications Board

Сертифицированный тестировщик

Программа обучения Базового уровня

Версия 2023

International Software Testing Qualifications Board

ISTQB® International Software Testing Qualifications Board

Уведомление об авторских правах

программа обучения может быть упомянута только после письменного уведомления об аккредитации материалов тренингов коллегиями, признанными ISTQB®.

- Любое частное лицо или группа частных лиц может использовать программу как основу для статей, книг или других производных письменных материалов, если авторы и ISTQB® упомянуты как источник и владельцы авторских прав программы.
- Любое другое использование этой программы запрещено без предварительного письменного одобрения ISTQB®.
- Любая коллегия, признанная ISTQB®, может переводить эту программу обучения при условии, что она воспроизводит вышеупомянутое Уведомление об авторских правах в переведенной версии программы.

История изменений

Версия	Дата	Содержание
RSTQB 2023.1	19 ноября 2023	Добавлен предметный указатель. Исправлены опечатки
RSTQB 2023	22 октября 2023	Программа обучения Сертифицированный тестировщик Базового уровня, обновление 2023 Перевод на русский язык.
CTFL v4.0	12 апреля 2023	Основная версия
CTFL v3.1.1	01 июля 2021	Обновление авторских прав и логотипа
CTFL v3.1	11 ноября 2019	Обновленный выпуск с минимальными изменениями
RSTQB 2018	24 февраля 2019	Программа обучения Сертифицированный тестировщик Базового уровня, обновление 2018 Перевод на русский язык.
ISTQB 2018	27 апреля 2018	Основная версия.
ISTQB 2018	12 февраля 2018	Бета версия
ISTQB 2018	19 января 2018	Перекрестная проверка внутренней версии 3.0
ISTQB 2018	15 января 2018	Предварительная перекрестная проверка внутренней версии 2.9, включающая правки основной команды.
ISTQB 2018	9 декабря 2017	Альфа-версия 2.5 – Техническое редактирование версии 2.0, новый контент не добавлен
ISTQB 2018	22 ноября 2017	Альфа-версия 2.0 – Программа обучения Сертифицированный тестировщик Базового уровня обновление 2018 – см. Приложение C - Замечания к выпуску
ISTQB 2018	12 июня 2017	Альфа-версия – Программа обучения Сертифицированный тестировщик Базового уровня обновление 2018 см. Приложение C - Замечания к выпуску
RSTQB 2011	13 апреля 2011	Программа обучения Сертифицированный тестировщик Базового уровня Перевод на русский язык
ISTQB 2011	1 апреля 2011	Программа обучения Сертифицированный тестировщик Базового уровня Выпуск сопровождения – см. Приложение E – Замечания к выпуску 2011
ISTQB 2010	30 марта 2010	Программа обучения Сертифицированный тестировщик Базового уровня Выпуск сопровождения – см. Приложение E – Замечания к выпуску 2010
RSTQB 2007	27 ноября 2007	Перевод на русский язык
ISTQB 2007	01 мая 2007	Программа обучения Сертифицированный тестировщик Базового уровня
ISTQB 2005	01 июля 2005	Программа обучения Сертифицированный тестировщик Базового уровня
ASQF V2.2	Июль 2003	Программа сертификации ASQF на специалиста по тестированию, базовый уровень, версия 2.2 “Lehrplan Grundlagen des Softwaretestens”
ISEB V2.0	25 февраля 1999	Программа сертификации ISEB на специалиста по тестированию, базовый курс, версия 2.0

Содержание

Благодарности

Перевод версии документа 2023 выполнен рабочей группой АНО «Коллегия экспертов по качеству программного обеспечения» (Russian Software Testing Qualifications Board, RSTQB): Маргарита Трофимова (руководитель группы), Александр Александров (редактор), Мария Колотева, Андрей Конушин, Михаил Костецкий, Елена Костина, Илья Кулаков, Вадим Луковатый и Светлана Юшина.

Документ был официально выпущен Генеральной Ассамблеей ISTQB 21 апреля 2023 года.

Этот документ был подготовлен объединенной командой рабочих групп ISTQB Базового уровня и Гибких методологий: Laura Albert, Renzo Cerquozzi (вице-председатель), Wim Decoutere, Klaudia Dussa-Zieger, Chintaka Indikadahena, Arnika Hryszko, Martin Klonk, Kenji Onishi, Michaël Pilaeten (сопредседатель), Meile Posthuma, Gandhinee Rajkomar, Stuart Reid, Eric Riou du Cosquer (сопредседатель), Jean-François Riverin, Adam Roman, Lucjan Stapp, Stephanie Ulrich (вице-председатель), Eshraka Zakaria.

Команда благодарит Stuart Reid, Patricia McQuaid и Leanne Howard за их техническое рецензирование, а также команду рецензентов и Национальные коллегии за их вклад и предложения.

Следующие специалисты принимали участие рецензировании, комментариях и обсуждении этой программы обучения: Adam Roman, Adam Scierski, Ágota Horváth, Ainsley Rood, Ale Rebon Portillo, Alessandro Collino, Alexander Alexandrov, Amanda Logue, Ana Ochoa, André Baumann, André Verschelling, Andreas Spillner, Anna Miazek, Arnd Pehl, Arne Becher, Attila Gyúri, Attila Kovács, Beata Karpinska, Benjamin Timmermans, Blair Mo, Carsten Weise, Chinthaka Indikadahena, Chris Van Bael, Ciaran O'Leary, Claude Zhang, Cristina Sobrero, Dandan Zheng, Dani Almog, Daniel Säther, Daniel van der Zwan, Danilo Magli, Darvay Tamás Béla, Dawn Haynes, Dena Pauletti, Dénes Medzihradszky, Doris Dötzer, Dot Graham, Edward Weller, Erhardt Wunderlich, Eric Riou Du Cosquer, Florian Fieber, Fran O'Hara, François Vaillancourt, Frans Dijkman, Gabriele Haller, Gary Mogyorodi, Georg Sehl, Géza Bujdosó, Giancarlo Tomasig, Giorgio Pisani, Gustavo Márquez Sosa, Helmut Pichler, Hongbao Zhai, Horst Pohlmann, Ignacio Trejos, Ilia Kulakov, Ine Lutterman, Ingvar Nordström, Iosif Itkin, Jamie Mitchell, Jan Giesen, Jean-Francois Riverin, Joanna Kazun, Joanne Tremblay, Joëlle Genois, Johan Klintin, John Kurowski, Jörn Münzel, Judy McKay, Jürgen Beniermann, Karol Frühauf, Katalin Balla, Kevin Kooh, Klaudia Dussa-Zieger, Klaus Erlenbach, Klaus Olsen, Krisztián Miskó, Laura Albert, Liang Ren, Lijuan Wang, Lloyd Roden, Lucjan Stapp, Mahmoud Khalaili, Marek Majernik, Maria Clara Choucair, Mark Rutz, Markus Niehammer, Martin Klonk, Márton Siska, Matthew Gregg, Matthias Hamburg, Mattijs Kemmink, Maud Schlich, May Abu-Sbeit, Meile Posthuma, Mette Bruhn-Pedersen, Michal Tal, Michel Boies, Mike Smith, Miroslav Renda, Mohsen Ekssir, Monika Stocklein Olsen, Murian Song, Nicola De Rosa, Nikita Kalyani, Nishan Portoyan, Nitzan Goldenberg, Ole Chr. Hansen, Patricia McQuaid, Patricia Osorio, Paul Weymouth, Pawel Kwasik, Peter Zimmerer, Petr Neugebauer, Piet de Roo, Radoslaw Smilgin, Ralf Bongard, Ralf Reissing, Randall Rice, Rik Marselis, Rogier Ammerlaan, Sabine Gschwandtner, Sabine Uhde, Salinda Wickramasinghe, Salvatore Reale, Sammy Kolluru, Samuel Ouko, Stephanie Ulrich, Stuart Reid, Surabhi Bellani, Szilard Szell, Tamás Gergely, Tamás Horváth, Tatiana Sergeeva, Tauhida Parveen, Thaer Mustafa, Thomas Eisbrenner, Thomas Harms, Thomas Heller, Tomas Rosenqvist, Werner Lieblang, Yaron Tsubery, Zhenlei Zuo и Zsolt Hargitai.

Рабочая группа Базового уровня ISTQB (Состав 2018): Klaus Olsen (председатель), Tauhida Parveen (вице-председатель), Rex Black (руководитель проекта), Eshraka Zakaria. Основная команда благодарит команду рецензентов (Debra Friedenberg, Ebbe Munk, Hans Schaefer, Judy McKay, Marie Walsh, Meile Posthuma, Mike Smith, Radoslaw Smilgin, Stephanie Ulrich, Steve Toms,

Corne Kruger, Dani Almog, Eric Riou du Cosquer, Igal Levi, Johan Klintin, Kenji Onishi, Rashed Karim, Stevan Zivanovic, Sunny Kwon, Thomas Müller, Vipul Kocher, Yaron Tsubery), а также все Национальные коллегии за их предложения.

Рабочая группа Базового уровня ISTQB (Состав 2011): Thomas Müller (председатель), Debra Friedenberg. Основная команда благодарит команду рецензентов (Dan Almog, Armin Beer, Rex Black, Julie Gardiner, Judy McKay, Tuula Pääkkönen, Eric Riou du Cosquier, Hans Schaefer, Stephanie Ulrich, Erik van Veenendaal), а также все Национальные коллегии за их предложения к текущей версии программы обучения.

Рабочая группа Базового уровня ISTQB (Состав 2010): Thomas Müller (председатель), Rahul Verma, Martin Klonk и Armin Beer. Основная команда благодарит команду рецензентов (Rex Black, Mette Bruhn-Pederson, Debra Friedenberg, Klaus Olsen, Judy McKay, Tuula Pääkkönen, Meile Posthuma, Hans Schaefer, Stephanie Ulrich, Pete Williams, Erik van Veenendaal), а также все Национальные коллегии за их предложения.

Рабочая группа Базового уровня ISTQB (Состав 2007): Thomas Müller (председатель), Dorothy Graham, Debra Friedenberg и Erik van Veenendaal. Основная команда благодарит команду рецензентов (Hans Schaefer, Stephanie Ulrich, Meile Posthuma, Anders Pettersson и Wonil Kwon), а также все Национальные коллегии за их предложения.

Рабочая группа Базового уровня ISTQB (Состав 2005): Thomas Müller (председатель), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson и Erik van Veenendaal. Основная команда благодарит команду рецензентов, а также все Национальные коллегии за их предложения.

0.Предисловие к программе обучения

0.1 Цель этого документа

Эта программа представляет основу международной сертификации на квалификацию Базового уровня ISTQB®. ISTQB® предоставляет эту программу:

1. Национальным коллегиям для перевода на национальный язык и аккредитации организаторов обучения. Национальные коллегии могут адаптировать программу обучения к особенностям местных языков и определить ссылки для адаптации к местным публикациям.
2. Экзаменационным комиссиям для формирования экзаменационных вопросов на местном языке, адаптированные к целям обучения каждого курса.
3. Организаторам обучения для разработки программы обучения и определения соответствующих методов обучения.
4. Кандидатам на получение сертификатов для подготовки к экзамену (в рамках программы обучения или независимо).
5. Международному сообществу разработки ПО и систем для продвижения профессии тестировщика ПО и систем, и использования как основы для книг и статей.

0.2 Сертифицированный тестировщик Базового уровня

Квалификация Базового уровня предназначена для всех, связанных с тестированием ПО, включая такие роли как тестировщики, тест аналитики, инженеры тестировщики, консультанты тестирования, руководители тестирования, разработчики ПО и члены команд разработки. Данная квалификация Базового уровня также подходит для всех, кто хочет получить понимание основ тестирования ПО, например, для руководителей проектов, руководителей по качеству, владельцев продуктов, руководителей разработки ПО, бизнес-аналитиков, ИТ-директоров и консультантов по управлению. Обладатели сертификата Базового уровня могут готовиться к более высокой квалификации в тестировании ПО.

0.3 Карьерный путь тестировщиков

Программа ISTQB® обеспечивает поддержку специалистов по тестированию на всех этапах их карьеры, предлагая как широту, так и глубину знаний. Специалисты, получившие сертификат ISTQB® Базового уровня, также могут быть заинтересованы в Продвинутом уровне (Тест-аналитик, Технический тест-аналитик и Руководитель тестирования), а затем и в Экспертном уровне (Управление тестированием или Улучшение процесса тестирования). Любой, кто хочет развить навыки в области тестирования в Agile-среде, может рассмотреть возможность получения сертификата Технический тестировщик в гибких методологиях или Масштабирование тестирования в организации с гибкими методологиями. Направление «Специалист» предлагает глубокое погружение в области, где используются конкретные подходы к тестированию и действия по тестированию (например, автоматизация тестирования, тестирование искусственного интеллекта, тестирование на основе моделей, тестирование мобильных приложений), которые связаны с конкретными областями тестирования (например, тестирование производительности, тестирование удобства использования, приемочное тестирование, тестирование безопасности) или ноу-хау тестирования для определенных областей промышленности (например, автомобилестроения или

игр). Посетите сайт www.istqb.org для получения последней информации о программе сертифицированных тестировщиков ISTQB.

0.4 Бизнес-результаты

В этом разделе перечислены 14 бизнес-результатов, ожидаемые от кандидата, имеющего сертификат Базового уровня.

FL-BO1 Понимать, что такое тестирование и почему оно полезно
FL-BO2 Понимать фундаментальные концепции тестирования ПО
FL-BO3 Определять подход к тестированию и действия, которые необходимо реализовать в зависимости от контекста тестирования
FL-BO4 Оценивать и улучшать качество документации
FL-BO5 Повышать эффективность и результативность тестирования
FL-BO6 Согласовывать процесс тестирования с жизненным циклом разработки ПО
FL-BO7 Понимать принципы управления тестированием
FL-BO8 Формировать и передавать четкие и понятные отчеты о дефектах
FL-BO9 Понимать факторы, влияющие на приоритеты и усилия, связанные с тестированием.
FL-BO10 Работать как часть кросс-функциональной команды
FL-BO11 Знать риски и преимущества, связанные с автоматизацией тестирования
FL-BO12 Определять основные навыки, необходимые для тестирования
FL-BO13 Понимать влияние риска на тестирование
FL-BO14 Эффективно сообщать о ходе и качестве тестирования

0.5 Проверяемые цели обучения и когнитивные уровни знаний

Цели обучения поддерживают достижение бизнес-результатов, а также используются для разработки сертификационных экзаменов Базового уровня. В целом все главы 1-6 данной программы обучения содержат цели обучения для уровня К1. Другими словами, кандидат на сертификат должен распознать, запомнить и воспроизвести термин или понятие, упоминаемое в любом из шести разделов. Конкретные уровни целей обучения указаны в начале каждой главы и классифицируются следующим образом:

- К1: Запомнить
- К2: Понять
- К3: Применить

Более подробная информация и примеры целей обучения приведены в Приложении А. Все термины, перечисленные в качестве ключевых слов сразу под заголовками глав, должны быть запомнены (К1), даже если они не упомянуты явно в целях обучения.

0.6 Экзамен Базового уровня

Сертификационные экзамены Базового уровня основаны на данной программе обучения. Ответы на экзаменационные вопросы могут потребовать использования материала, основанного более чем на одной главе этой программы обучения. Все разделы программы обучения поддаются экзаменационной проверке, за исключением введения и приложений. Стандарты и книги включены в качестве ссылок (Глава 7), но их содержание не проверяется на экзамене, за исключением того, что обобщенно в этой

программе обучения из этих стандартов и книг. См. документ «Структура и правила экзамена Базового уровня».

0.7 Аккредитация

Коллегия-член ISTQB® может аккредитовать обучающие организации, учебные материалы которых соответствуют этой программе обучения. Обучающие организации должны получить руководство по аккредитации от коллегии-члена или органа, который проводит аккредитацию. Аккредитованный курс признается соответствующим этой программе, и в рамках курса разрешается сдавать экзамен ISTQB®. Руководство по аккредитации для этой программы обучения соответствует общему Руководству по аккредитации, опубликованному Рабочей группой по управлению процессами и соответствию ISTQB®.

0.8 Поддерживаемые стандарты

В программе обучения Базового уровня есть ссылки на стандарты (например, IEEE, ISO и т.д.). Цель этих ссылок — предоставить основу (как в ссылках на ISO 25010 в отношении характеристик качества) или предоставить источник дополнительной информации, если читатель пожелает. Стандарты не предназначены для проверки на экзамене. Дополнительную информацию о стандартах см. в главе 7. Ссылки.

0.9 Актуальность программы обучения

Индустрия программного обеспечения быстро меняется. Чтобы справиться с этими изменениями и предоставить заинтересованным сторонам доступ к релевантной и актуальной информации, рабочие группы ISTQB® разместили ссылки на сайте www.istqb.org, которые относятся к вспомогательным документам, изменениям в стандартах и новым событиям в отрасли. Эта информация не подлежит проверке в рамках этой программы обучения.

0.10 Уровень детализации

Уровень детализации этой программы обучения позволяет проводить согласованные на международном уровне курсы и экзамены. Для достижения этой цели программа обучения состоит из:

- Общих целей обучения, описывающих намерения сертифицированного тестировщика базового уровня.
- Списка терминов, которые кандидаты должны запомнить.
- Целей обучения для каждой области знаний с описанием когнитивных результатов обучения, которые должны быть достигнуты.
- Описание ключевых понятий, включая ссылки на такие источники, как литература или стандарты.

Содержание программы не является описанием всей области знаний по тестированию ПО; оно отражает уровень детализации, который необходимо охватить в учебных курсах базового уровня. Основное внимание уделяется областям тестирования и методам, которые можно применить к большинству проектов разработки ПО независимо от используемого жизненного цикла.

0.11 Как организована эта программа обучения

Программа обучения содержит шесть глав с подлежащим изучению содержанием. В заголовке верхнего уровня каждой главы указано время обучения для этой главы. Ниже уровня главы время не указывается. Для аккредитованных учебных курсов программа требует минимум 1135 минут (18 часов 55 минут) обучения, распределенных по шести главам следующим образом:

- Глава 1: Основы тестирования (180 минут)
    - Обучающийся изучает основные принципы тестирования, причины необходимости тестирования и цели тестирования.
    - Обучающийся понимает процесс тестирования, основные действия по тестированию и тестовое окружение.
    - Обучающийся понимает необходимые навыки для тестирования.
- Глава 2: Тестирование в жизненном цикле разработки программного обеспечения (130 минут)
    - Обучающийся узнает, как тестирование включается в различные подходы к разработке.
    - Обучающийся изучает концепции подхода опережающего проектирования тестов, а также DevOps.
    - Обучающийся узнает о различных уровнях тестирования, типах тестов и тестировании в период сопровождения.
- Глава 3: Статическое тестирование (80 минут)
    - Обучающийся узнает об основах статического тестирования, процессе обратной связи и рецензирования.
- Глава 4: Анализ и проектирование тестов (390 минут)
    - Обучающийся учится применять методы тестирования «черного ящика», «белого ящика» и методы тестирования, основанные на опыте, для получения тестовых сценариев из различных артефактов ПО.
    - Обучающийся узнает о подходе к тестированию, основанном на сотрудничестве.
- Глава 5: Управление тестированием (335 минут)
    - Обучающийся учится планировать тесты в целом и оценивать трудозатраты на тестирование.
    - Обучающийся узнает, как риски могут повлиять на объем тестирования.
    - Обучающийся учится отслеживать и контролировать активности тестирования.
    - Обучающийся узнает, как управление конфигурацией поддерживает тестирование.
    - Обучающийся учится четко и понятно сообщать о дефектах.
- Глава 6: Инструменты тестирования (20 минут)
    - Обучающийся учится классифицировать инструменты и понимать риски и преимущества автоматизации тестирования.

1. Основы тестирования – 180 минут

Ключевые слова

Анализ тестирования, базис тестирования, валидация, верификация, выполнение теста, дефект, завершение тестирования, качество, контроль тестирования, мониторинг тестирования, обеспечение качества, объект тестирования, отказ, отладка, ошибка, первопричина, планирование тестирования, покрытие, проектирование теста, процедура тестирования, реализация теста, результат теста, тестирование, тестовое обеспечение, тестовое условие, тестовые данные, тестовый сценарий, цель тестирования

Цели обучения главы 1:

1.1 Что такое тестирование?

FL-1.1.1 (K1) Определить типичные цели тестирования
FL-1.1.2 (K2) Различить тестирование и отладку

1.2 Почему тестирование необходимо?

FL-1.2.1 (K2) Объяснить на примерах, почему тестирование необходимо
FL-1.2.2 (K1) Запомнить связь между тестированием и обеспечением качества
FL-1.2.3 (K2) Выделить различия между первопричиной, ошибкой, дефектом и отказом

1.3 Принципы тестирования

FL-1.3.1 (K2) Объяснить семь принципов тестирования

1.4 Активности тестирования, тестовое обеспечение и роли в тестировании

FL-1.4.1 (K2) Обобщить различные активности и задачи тестирования
FL-1.4.2 (K2) Объяснить влияние контекста на процесс тестирования
FL-1.4.3 (K2) Различить тестовое обеспечение, поддерживающее активности тестирования
FL-1.4.4 (K2) Объяснить ценность обеспечения трассируемости
FL-1.4.5 (K2) Сравнить различные роли в тестировании

1.5 Основные навыки и передовой опыт в тестировании

FL-1.5.1 (K2) Привести примеры основных навыков, необходимых для тестирования
FL-1.5.2 (K1) Запомнить преимущества командного подхода
FL-1.5.3 (K2) Выделить преимущества и недостатки независимости тестирования

1.1 Что такое тестирование?

Программные системы являются неотъемлемой частью нашей повседневной жизни. Большинство людей сталкивались с программным обеспечением, которое не работало должным образом. Программное обеспечение, работающее неправильно, может привести ко многим проблемам, включая потерю денег, времени или деловой репутации, а в крайних случаях даже к потере здоровья или смерти. Тестирование программного обеспечения оценивает качество программного обеспечения и помогает снизить риск отказа программного обеспечения в работе.

Тестирование программного обеспечения — это процесс в рамках жизненного цикла разработки программного обеспечения, который оценивает качество компонента или системы, а также связанных с ними рабочих продуктов. При тестировании эти рабочие продукты называются объектами тестирования. Распространенное заблуждение о тестировании заключается в том, что оно состоит только из выполнения тестов (т.е. запуска программного обеспечения и проверки результатов тестов). Однако тестирование программного обеспечения также включает другие действия и должно быть согласовано с жизненным циклом разработки программного обеспечения (см. главу 2).

Еще одно распространенное заблуждение о тестировании заключается в том, что тестирование полностью сосредоточено на верификации объекта тестирования. Хотя тестирование включает верификацию, т.е. проверку того, соответствует ли система заданным требованиям, оно также включает и валидацию, что означает проверку того, соответствует ли система потребностям пользователей и других заинтересованных сторон в ее операционной среде.

Тестирование может быть динамическим или статическим. Динамическое тестирование включает выполнение программного обеспечения, а статическое тестирование — нет. Статическое тестирование включает рецензирование (см. главу 3) и статический анализ. Динамическое тестирование использует различные методы и подходы к тестированию для получения тестовых сценариев (см. главу 4).

Тестирование — это не только техническая активность. Необходимо также надлежащим образом планировать, управлять, оценивать, отслеживать и контролировать тестирование (см. главу 5).

Тестировщики используют инструменты (см. главу 6), но важно помнить, что тестирование — это в значительной степени интеллектуальная деятельность, требующая от тестировщиков специальных знаний, использования аналитических навыков и применения критического и системного мышления (Myers 2011, Roman 2018).

Дополнительную информацию о концепциях тестирования программного обеспечения можно найти в стандарте ISO/IEC/IEEE 29119-1.

1.1.1 Цели тестирования

Типичные цели тестирования:

- Оценка рабочих продуктов, таких как требования, пользовательские истории, проекты и код.
- Провоцирование отказов и обнаружение дефектов.
- Обеспечение необходимого покрытия объекта тестирования.
- Снижение уровня риска ненадлежащего качества программного обеспечения.
- Проверка выполнения зафиксированных требований.

- Проверка того, что объект тестирования соответствует контрактным, юридическим и нормативным требованиям.
- Предоставление информации заинтересованным сторонам для принятия обоснованных решений.
- Создание уверенности в качестве объекта тестирования.
- Проверка того, завершен ли объект тестирования и работает ли он так, как ожидают заинтересованные стороны.

Цели тестирования могут варьироваться в зависимости от контекста, который включает тестируемый рабочий продукт, уровень тестирования, риски, применяемый жизненный цикл разработки программного обеспечения (SDLC), а также факторы, связанные с бизнес-контекстом, например, корпоративная структура, условия конкуренции или время выхода на рынок.

1.1.2 Тестирование и отладка

Тестирование и отладка — это разные действия. Тестирование может спровоцировать отказы, вызванные дефектами в программном обеспечении (динамическое тестирование), или может обнаруживать дефекты непосредственно в объекте тестирования (статическое тестирование).

Динамическое тестирование (см. главу 4) вызывает отказ, в то время как отладка связана с поиском причин этого отказа (дефектов), анализом этих причин и их устранением. Типичный процесс отладки в этом случае включает:

- Воспроизведение отказа
- Диагностика (поиск первопричины)
- Устранение причины

Последующее подтверждающее тестирование проверяет, действительно ли исправления устранили проблему. Предпочтительно проводить подтверждающее тестирование тем же лицом, которое проводило исходное тестирование. Также может быть проведено последующее регрессионное тестирование, чтобы проверить, не вызывают ли сделанные исправления отказы в других частях объекта тестирования (дополнительную информацию о подтверждающем и регрессионном тестировании см. в разделе 2.2.3).

Статическое тестирование выявляет дефект, в то время как отладка связана с его устранением. Поскольку статическое тестирование непосредственно находит дефекты и не может привести к отказам, нет необходимости в воспроизведении или диагностике дефектов, (см. главу 3).

1.2 Почему тестирование необходимо?

Тестирование как форма контроля качества, помогает в достижении согласованных целей в пределах установленного объема работ, времени, уровня качества и бюджетных ограничений. Вклад тестирования в успех не должен ограничиваться действиями команды тестировщиков. Любая заинтересованная сторона может использовать свои навыки тестирования, чтобы приблизить проект к успеху. Тестирование компонентов, систем и сопутствующей документации помогает выявить дефекты в программном обеспечении.

1.2.1 Вклад тестирования в общий успех

Тестирование обеспечивает экономически эффективные средства обнаружения дефектов. Затем эти дефекты могут быть устранены (путем отладки — действия, не являющегося активностью тестирования), поэтому тестирование способствует повышению качества объектов тестирования лишь косвенно.

Тестирование предоставляет средства прямой оценки качества объекта тестирования на различных этапах ЖЦ ПО. Эти средства используются как часть более крупной деятельности по управлению проектом, способствуя принятию решений о переходе к следующему этапу ЖЦ ПО, например, решению о выпуске.

Тестирование предоставляет пользователям косвенное представление о проекте разработки. Тестировщики уверены, что их понимание потребностей пользователей учитывается на протяжении всего жизненного цикла разработки. Альтернативой является привлечение репрезентативного набора пользователей в рамках проекта разработки, что трудно осуществить из-за высоких затрат и отсутствия подходящих пользователей.

Тестирование также может потребоваться для проверки соблюдения договорных или юридических требований или нормативных стандартов.

1.2.2 Тестирование и обеспечение качества (QA)

Хотя люди часто используют термины «тестирование» и «обеспечение качества» (QA) взаимозаменяемо — это не одно и то же. Тестирование является видом контроля качества (QC).

Контроль качества (QC) — это корректирующий подход, ориентированный на продукт, который сосредотачивается на действиях, поддерживающих достижение надлежащего уровня качества. Тестирование является основным видом контроля качества, в то время как другие виды включают формальные методы (проверку модели и доказательство правильности), моделирование и прототипирование.

Обеспечение качества (QA) — это превентивный подход, ориентированный на процесс, который сосредотачивается на внедрении и улучшении процессов. Он предполагает, что если правильно следовать хорошему процессу, то будет создан хороший продукт. Обеспечение качества применяется как к процессам разработки, так и к процессам тестирования, и за него несет ответственность каждый участник проекта.

Результаты тестов используются как QA, так и QC. В QC они используются для исправления дефектов, а в QA они обеспечивают обратную связь, насколько хорошо выполняются процессы разработки и тестирования.

1.2.3 Ошибки, дефекты, отказы и первопричины

Люди совершают ошибки, которые порождают дефекты (недочеты, «баги»), а дефекты, в свою очередь, могут привести к отказам. Люди совершают ошибки по разным причинам, например, нехватка времени, сложность рабочих продуктов, процессов, инфраструктуры или взаимодействий, или просто потому, что они устали или не имеют надлежащей подготовки.

Дефекты можно найти в документации (спецификации требований или автоматизированном сценарии тестирования), в исходном коде или в сопутствующем рабочем продукте (например, файл сборки). Дефекты в рабочих продуктах, созданных ранее в ЖЦ ПО, часто приводят к появлению дефектов в рабочих продуктах на более поздних этапах жизненного цикла, если их не обнаружить вовремя. Если дефект в коде выполняется, это может (но не обязательно во всех ситуациях)

привести к отказу. Например, для некоторых дефектов требуются очень специфические входные данные или предварительные условия, чтобы вызвать отказ, который может произойти редко или никогда.

Ошибки и дефекты — не единственная причина отказов. Например, радиация, электромагнитные поля и загрязнения могут вызвать отказ в программно-аппаратных средствах или повлиять на выполнение программного обеспечения, изменяя условия работы аппаратных средств.

Первопричина — это самые ранние действия или условия, которые способствовали созданию дефектов. Первопричины выявляются методом анализа первопричин, который обычно применяется при возникновении отказа или обнаружении дефекта. Обращая внимание на наиболее существенные первопричины, анализ первопричин может привести к улучшению процессов, которые предотвратят или уменьшат частоту появления значительного числа будущих дефектов.

1.3 Принципы тестирования

За прошедшие десятилетия был предложен ряд принципов тестирования, которые являются общим руководством для тестирования в целом. В этой программе описаны семь таких принципов.

1. **Тестирование демонстрирует наличие дефектов, а не их отсутствие**. Тестирование может показать наличие дефектов в объекте тестирования, но не может доказать их отсутствие (Buxton, 1970). Тестирование снижает вероятность того, что дефекты в объекте тестирования останутся необнаруженными, но даже если дефекты не были обнаружены, тестирование не доказывает корректности объекта тестирования.

2. **Исчерпывающее тестирование невозможно**. Полное тестирование с использованием всех комбинаций вводов и предусловий физически невыполнимо, за исключением тривиальных случаев (Manna, 1978). Вместо того, чтобы пытаться провести исчерпывающее тестирование, следует использовать методы тестирования (см. главу 4), расстановку приоритетов тестовых сценариев (см. раздел 5.1.5) и тестирование, основанное на рисках (см. раздел 5.2), чтобы сосредоточить усилия по тестированию.

3. **Раннее тестирование экономит время и деньги**. Дефекты, устраненные на ранней стадии процесса, не вызовут последующих дефектов в производных рабочих продуктах. Стоимость качества будет снижена, так как позже в ЖЦ ПО будет происходить меньше отказов (Boehm 1981). Для раннего обнаружения дефектов как можно раньше следует начинать как статическое тестирование (см. главу 3), так и динамическое тестирование (см. главу 4).

4. **Кластеризация дефектов**. Обычно небольшое количество системных компонентов содержит большинство обнаруженных дефектов или порождает большинство эксплуатационных отказов. (Enders, 1975). Это явление является иллюстрацией принципа Парето. Предсказанные и фактические кластеры дефектов, наблюдаемые в ходе тестирования или эксплуатации, являются важными входными данными для тестирования, основанного на рисках (см. раздел 5.2).

5. **Тесты устаревают**. Если одни и те же тесты повторяются много раз, они становятся все более неэффективными в обнаружении новых дефектов (Beizer 1990). Для обнаружения новых дефектов может потребоваться изменение существующих тестов и тестовых данных, а также написание новых тестов. Однако в некоторых случаях повторение одних и тех же тестов может иметь положительный результат, например, при автоматизированном регрессионном тестировании (см. раздел 2.2.3).

6. **Тестирование зависит от контекста**. Не существует единого универсального подхода к тестированию. Тестирование выполняется по-разному в зависимости от контекста (Kaner 2011).

7. **Заблуждение об отсутствии дефектов**. Было бы ошибкой ожидать, что верификация программного обеспечения обеспечит успех системы. Тщательное тестирование всех указанных требований и исправление всех обнаруженных дефектов может привести к созданию системы, которая не будет соответствовать потребностям и ожиданиям пользователей, не будет помогать в достижении бизнес-целей заказчика и будет уступать другим конкурирующим системам. Также в дополнение к верификации следует проводить валидацию (Boehm 1981).

1.4 Активности тестирования, тестовое обеспечение и роли в тестировании

Тестирование зависит от контекста, но на верхнем уровне существуют общие наборы активностей тестирования, без которых тестирование вряд ли достигнет поставленных целей. Эти наборы активностей тестирования и образуют процесс тестирования. Процесс тестирования может быть адаптирован к конкретной ситуации на основе различных факторов. Обычно при планировании тестирования для конкретной ситуации решается, какие активности входят в процесс тестирования, как они реализуются и когда происходят (см. раздел 5.1).

В следующих разделах описываются общие аспекты процесса тестирования с точки зрения активностей тестирования и задач, влияния контекста, тестового обеспечения, трассируемости между базисом тестирования и тестовым обеспечением, а также ролей тестирования.

Дополнительную информацию о процессах тестирования предоставляет стандарт ISO/IEC/IEEE 29119-2.

1.4.1 Активности и задачи тестирования

Процесс тестирования обычно состоит из основных групп активностей, описанных ниже. Хотя многие из этих действий могут выглядеть логически последовательными, они часто реализуются итеративно или параллельно. Кроме того, обычно требуется адаптация этих активностей тестирования к системе и проекту.

Планирование тестирования состоит из определения целей тестирования и последующего выбора подхода, который лучше всего помогает достигать этих целей в рамках ограничений, налагаемых общим контекстом. Планирование тестирования более подробно объясняется в разделе 5.1.

Мониторинг и контроль тестирования. Мониторинг тестирования включает постоянную проверку всех активностей тестирования и сравнение фактического хода работ с планом тестирования. Контроль тестирования включает выполнение действий, необходимых для достижения целей тестирования. Мониторинг и контроль тестирования более подробно описаны в разделе 5.3.

Анализ тестирования включает анализ базиса тестирования для определения тестируемых функций, а также для выявления и расстановки приоритетов связанных тестовых условий, соответствующих рисков и их уровней (см. раздел 5.2). Базис и объекты тестирования также оцениваются на предмет выявления дефектов, которые они могут содержать, и для оценки их тестируемости. Анализ тестирования часто поддерживается использованием методик тестирования (см. главу 4). Анализ тестирования отвечает на вопрос «что тестировать?» с точки зрения измеримых критериев покрытия.

Проектирование тестов включает разработку тестовых условий в тестовых сценариях и другом тестовом обеспечении (например, концепции тестирования). Эта деятельность включает идентификацию элементов покрытия, которые служат руководством для определения входных

данных тестового сценария. Для поддержки этой деятельности могут использоваться различные методики тестирования (см. главу 4). Проектирование тестов также включает определение требований к тестовым данным, проектирование тестового окружения и определение любой другой необходимой инфраструктуры и инструментов. Проектирование тестов отвечает на вопрос «как тестировать?».

Реализация тестов включает создание или приобретение тестового обеспечения, необходимого для выполнения тестирования (например, тестовых данных). Тестовые сценарии могут быть организованы в виде процедур тестирования и часто собраны в наборы тестов. При реализации тестов создаются ручные и автоматизированные сценарии тестирования. Для эффективного тестирования процедуры тестирования упорядочиваются и организуются в расписание выполнения тестов (см. раздел 5.1.5). В рамках реализации тестов также создается и проверяется правильность настройки тестового окружения.

Выполнение тестов включает запуск тестов в соответствии с расписанием выполнения тестов (прогоны тестов). Выполнение тестов может быть ручным или автоматизированным, принимать разные формы, включая непрерывное тестирование или сеансы парного тестирования. Фактические результаты тестов сравниваются с ожидаемыми. Результаты тестов протоколируются. Аномалии анализируются для выявления их вероятных причин. Этот анализ позволяет сообщать об аномалиях на основе наблюдаемых отказов (см. раздел 5.5).

Действия по завершению тестирования обычно происходят на вехах проекта (например, выпуск, завершение итерации, завершение уровня тестирования), а для любых не устраненных дефектов создаются запросы на изменение или элементы бэклога. В рамках завершения тестирования идентифицируется и архивируется или передается соответствующим командам любое тестовое обеспечение, которое может быть полезно в будущем. Тестовое окружение сворачивается до согласованного состояния. Для извлечения уроков и улучшений в будущих итерациях, выпусках или проектах анализируются активности тестирования (см. раздел 2.1.6). Создается и передается заинтересованным сторонам итоговый отчет о тестировании.

1.4.2 Процесс тестирования в контексте

Тестирование не проводится изолированно. Активности тестирования являются неотъемлемой частью процессов разработки, осуществляемых внутри организации. Тестирование также финансируется заинтересованными сторонами, и его конечная цель — помочь удовлетворить бизнес-потребности заинтересованных сторон. Таким образом, способ проведения тестирования будет зависеть от ряда контекстных факторов, включая:

- Заинтересованные стороны (потребности, ожидания, требования, готовность к сотрудничеству и т.д.)
- Члены команды (навыки, знания, уровень опыта, доступность, потребности в обучении и т.д.)
- Предметная область (критичность объекта тестирования, выявленные риски, потребности рынка, конкретные правовые нормы и т.д.)
- Технические факторы (тип программного обеспечения, архитектура продукта, используемая технология и т.д.)
- Ограничения проекта (объем, время, бюджет, ресурсы и т.д.)
- Организационные факторы (организационная структура, существующие политики, используемые методики и т.д.)

- Жизненный цикл разработки программного обеспечения (инженерные практики, методы разработки и т.д.)
- Инструменты (доступность, удобство использования, соответствие и т. д.)

Эти факторы будут влиять на многие вопросы, связанные с тестированием, включая: стратегию тестирования, используемые методы тестирования, степень автоматизации тестирования, требуемый уровень покрытия, уровень детализации документации тестирования, отчетность и т. д.

1.4.3 Тестовое обеспечение

Тестовое обеспечение создается как выходной рабочий продукт активностей тестирования, описанных в разделе 1.4.1. Существуют значительные различия в том, как разные организации создают, формируют, называют, организуют и управляют своими рабочими продуктами. Надлежащее управление конфигурацией (см. раздел 5.4) обеспечивает согласованность и целостность рабочих продуктов. Следующий список рабочих продуктов не является исчерпывающим:

- **Рабочие продукты планирования тестирования** включают: план тестирования, расписание тестирования, реестр рисков, критерии входа и выхода (см. раздел 5.1). Реестр рисков представляет собой список рисков с их вероятностью, влиянием и информацией о снижении рисков (см. раздел 5.2). Расписание тестирования, реестр рисков и критерии входа и выхода часто являются частью плана тестирования.
- **Рабочие продукты мониторинга и контроля тестирования** включают: отчеты о ходе тестирования (см. раздел 5.3.2), документацию по директивам управления (см. раздел 5.3) и информацию о рисках (см. раздел 5.2).
- **Рабочие продукты анализа тестирования** включают: (упорядоченные) тестовые условия (например, критерии приемки, см. раздел 4.5.2) и любые отчеты о дефектах, касающиеся базиса тестирования (если они не исправлены сразу).
- **Рабочие продукты проектирования тестов** включают: (упорядоченные) тестовые сценарии, концепцию тестирования, элементы покрытия, требования к тестовым данным и требования к тестовому окружению.
- **Рабочие продукты реализации тестирования** включают: процедуры тестирования, автоматизированные тестовые сценарии, наборы тестов, тестовые данные, расписание выполнения тестов и элементы тестового окружения. Примеры элементов тестового окружения включают: заглушки, драйверы, симуляторы и виртуализации служб.
- **Рабочие продукты выполнения тестов** включают: протоколы тестирования и отчеты о дефектах (см. раздел 5.5).
- **Рабочие продукты завершения тестирования** включают: итоговые отчет о тестировании (см. раздел 5.3.2), действия по улучшению последующих проектов или итераций, документированные извлеченные уроки и запросы на изменение (например, в виде элементов бэклога продукта).

1.4.4 Трассируемость между базисом тестирования и тестовым обеспечением

Чтобы внедрить эффективный мониторинг и контроль тестирования, важно установить и поддерживать на протяжении всего процесса тестирования трассируемость между элементами

базиса тестирования, тестовым обеспечением, связанным с этими элементами (например, тестовыми условиями, рисками, тестовыми наборами), результатами тестирования и обнаруженными дефектами.

Хорошая трассируемость поддерживает оценку покрытия, поэтому очень полезно, если в базисе тестирования определены измеримые критерии покрытия. Критерии покрытия могут служить ключевыми индикаторами производительности, которые показывают, в какой степени были достигнуты цели тестирования (см. раздел 1.1.1). Например:

- Трассируемость от тестовых сценариев к требованиям может подтвердить, что требования покрываются тестовыми сценариями.
- Трассируемость от результатов тестов к рискам может использоваться для оценки уровня остаточного риска в объекте тестирования.

В дополнение к оценке покрытия хорошая трассируемость позволяет определить влияние изменений, облегчает аудиты тестирования и помогает соответствовать критериям управления ИТ. Хорошая трассируемость также делает отчеты о ходе и завершении тестирования более понятными благодаря включению статуса элементов базиса тестирования. Она также может помочь в доведении технических аспектов тестирования до заинтересованных сторон в понятной форме. Трассируемость предоставляет информацию для оценки качества продукта, возможностей процесса и хода проекта в соответствии с бизнес-целями.

1.4.5 Роли в тестировании

В этой программе рассматриваются две основные роли в тестировании: руководитель тестирования и тестировщик. Активности и задачи, назначенные этим двум ролям, зависят от таких факторов, как контекст проекта и продукта, навыки людей, выполняющих роли, а также организация.

Руководитель тестирования берет на себя общую ответственность за процесс тестирования, группу тестирования и руководство активностями по тестированию. Руководитель тестирования в основном сосредоточен на планировании, мониторинге и контроле, а также завершении тестирования. Способ выполнения роли руководителя тестирования зависит от контекста. Например, в гибкой разработке программного обеспечения некоторые задачи руководителя тестирования могут выполняться самой командой. Задачи, охватывающие несколько команд или всю организацию, могут выполняться руководителями по тестированию, не входящими в группу разработчиков.

Тестировщик берет на себя общую ответственность за инженерный (технический) аспект тестирования. Тестировщик в основном сосредоточен на деятельности по анализу тестирования, разработке, реализации и выполнении тестов.

В разное время эти роли могут брать на себя разные люди. Например, роль руководителя тестирования может выполняться руководителем группы, руководителем тестирования, руководителем разработки и т.д. Также допустимо, чтобы один человек одновременно выполнял функции тестировщика и руководителя тестирования.

1.5 Основные навыки и передовой опыт в тестировании

Навык — это способность делать что-то хорошо, которая исходит из знаний, практики и способностей. Хорошие тестировщики должны обладать определенными необходимыми навыками, чтобы хорошо выполнять свою работу. Хорошие тестировщики должны быть эффективными

командными игроками и должны уметь выполнять тестирование на разных уровнях независимости тестирования.

1.5.1 Общие навыки, необходимые для тестирования

Следующие навыки являются общими, но особенно важны для тестировщиков:

- Знания о тестировании (для повышения эффективности тестирования, например, с помощью методов тестирования),
- Тщательность, внимательность, любознательность, внимание к деталям, методичность (чтобы выявлять дефекты, особенно те, которые трудно найти),
- Хорошие коммуникативные навыки, умение слушать, умение работать в команде (чтобы эффективно взаимодействовать со всеми заинтересованными сторонами, передавать информацию другим, быть понятым, а также сообщать о дефектах и обсуждать их),
- Аналитическое мышление, критическое мышление, креативность (для повышения эффективности тестирования),
- Технические знания (для повышения эффективности тестирования, например, за счет использования соответствующих инструментов тестирования),
- Знание предметной области (чтобы понимать и общаться с конечными пользователями/представителями бизнеса).

Тестировщики часто являются носителями плохих новостей. А обвинять того, кто приносит плохие новости, - обычная человеческая черта. Это делает коммуникативные навыки критически важными для тестировщиков. Сообщение результатов тестирования может быть воспринято как критика продукта и его автора. Предвзятость автора продукта может затруднить принятие информации, которая не согласуется с его текущими убеждениями. Некоторые люди могут воспринимать тестирование как деструктивную деятельность, даже несмотря на то, что оно в значительной степени способствует успеху проекта и качеству продукта. Чтобы попытаться улучшить эту точку зрения, информацию о дефектах и сбоях следует сообщать конструктивным образом.

1.5.2 Командный подход

Одним из важных навыков тестировщика является способность эффективно работать в команде и вносить положительный вклад в достижение целей команды. Весь командный подход — практика, пришедшая из экстремального программирования (см. раздел 2.1), — строится на этом навыке.

При общекомандном подходе любую задачу может выполнить любой член команды, обладающий необходимыми знаниями и навыками, и каждый несет ответственность за качество. Члены команды используют общее рабочее пространство (физическое или виртуальное), поскольку совместное размещение облегчает общение и взаимодействие. Командный подход улучшает командную динамику, общение и сотрудничество внутри команды, а также создает синергию, позволяя использовать различные наборы навыков в команде на благо проекта.

Тестировщики тесно сотрудничают с другими членами команды, чтобы обеспечить достижение желаемого уровня качества ПО. Например, тестировщики сотрудничают с представителями бизнеса, чтобы помочь им создать подходящие приемочные тесты, а также с разработчиками, чтобы согласовать стратегию тестирования и выбрать подходы к автоматизации тестирования. Таким образом, тестировщики могут передавать знания о тестировании другим членам команды и влиять на разработку продукта.

В зависимости от контекста командный подход не всегда может быть уместным. Например, в некоторых ситуациях, связанных с критической безопасностью, может потребоваться высокий уровень независимости тестирования.

1.5.3 Независимость тестирования

Определенная степень независимости делает тестировщика более эффективным в обнаружении дефектов из-за различий между когнитивными убеждениями автора и тестировщика (ср. Salman 1995). Однако независимость не заменяет знания, например, разработчики могут эффективно находить множество дефектов в своем собственном коде.

Рабочие продукты могут быть протестированы их автором (отсутствие независимости), коллегами автора из той же команды (некоторая независимость), тестировщиками вне команды автора, но внутри организации (высокая степень независимости) или тестировщиками вне организации (очень высокая автономность). Для большинства проектов обычно лучше проводить тестирование с несколькими уровнями независимости (например, разработчики, выполняющие тестирование компонентов и их интеграции, группа тестирования, выполняющая системное и системное интеграционное тестирование, и представители бизнеса, выполняющие приемочное тестирование).

Основное преимущество независимого тестирования заключается в том, что независимые тестировщики, по сравнению с разработчиками, лучше выявляют разные виды отказов и дефектов из-за их различающегося опыта, технических взглядов и убеждений. Кроме того, независимый тестировщик может проверить, оспорить или опровергнуть предположения, сделанные заинтересованными сторонами во время разработки спецификации или внедрения системы.

Однако есть и некоторые недостатки независимости. Независимые тестировщики могут быть изолированы от команды разработчиков, что может привести к отсутствию сотрудничества, проблемам со связью или враждебным отношениям с командой разработчиков. Разработчики могут потерять чувство ответственности за качество. Независимых тестировщиков могут начать рассматривать как узкое место или обвинять в задержках выпуска.

2. Тестирование в жизненном цикле разработки программного обеспечения – 130 минут

Ключевые слова

Интеграционное тестирование, компонентное тестирование, нефункциональное тестирование, объект тестирования, подтверждающее тестирование, приемочное тестирование, регрессионное тестирование, сдвиг влево, системное интеграционное тестирование, тестирование в период сопровождения, тестирование интеграции компонентов, тестирование методом белого ящика, тестирование методом черного ящика, тип тестирования, уровень тестирования, функциональное тестирование

Цели обучения для главы 2:

2.1 Тестирование в контексте жизненного цикла разработки программного обеспечения

FL-2.1.1 (K2) Объяснить влияние выбранного жизненного цикла разработки программного обеспечения на тестирование

FL-2.1.2 (K1) Запомнить эффективные практики тестирования, применяющиеся во всех жизненных циклах разработки программного обеспечения

FL-2.1.3 (K1) Запомнить примеры подходов опережающего проектирования тестов в разработке программного обеспечения

FL-2.1.4 (K2) Обобщить как DevOps может влиять на тестирование

FL-2.1.5 (K2) Объяснить подход сдвига влево

FL-2.1.6 (K2) Объяснить, как ретроспективы могут быть использованы в качестве механизма для улучшения процесса

2.2 Уровни тестирования и типы тестирования

FL-2.2.1 (K2) Выделить различные уровни тестирования

FL-2.2.2 (K2) Выделить различные типы тестирования

FL-2.2.3 (K2) Различать подтверждающее тестирование и регрессионное тестирование

2.3 Тестирование в период сопровождения

FL-2.3.1 (K2) Обобщить тестирование в период сопровождения и его предпосылки

2.1 Тестирование в контексте жизненного цикла разработки программного обеспечения

Модель жизненного цикла разработки программного обеспечения — это абстрактное, высокоуровневое представление процесса разработки программного обеспечения. Модель жизненного цикла разработки программного обеспечения описывает как различные фазы разработки программного обеспечения и типы активностей, осуществляемые в рамках этого процесса, логически и хронологически связаны друг с другом. В качестве примеров моделей можно привести: последовательные модели разработки (водопадная модель, V-модель), итерационные модели разработки (спиральная модель, прототипирование) и инкрементальные модели разработки (Rational Unified Process).

Некоторые активности в рамках процессов разработки программного обеспечения также могут быть описаны более детализированными методами разработки программного обеспечения и практиками гибкой методологии. В качестве примеров можно привести: разработка через приемочное тестирование (acceptance test-driven development, ATDD), разработка на основе поведения (behavior-driven development, BDD), проектирование на основе знания предметной области (domain-driven design, DDD), экстремальное программирование (extreme programming, XP), разработка на основе функционала (feature-driven development, FDD), Kanban, Lean IT, Scrum и разработка на основе тестов (test-driven development, TDD).

2.1.1 Влияние жизненного цикла разработки программного обеспечения на тестирование

Для достижения успеха тестирование должно быть адаптировано к выбранному жизненному циклу разработки программного обеспечения. Жизненный цикл разработки программного обеспечения влияет на:

- Объем и расписание активностей тестирование (например, уровней тестирования и типов тестирования)
- Уровень детализации документации тестирования
- Выбор техник тестирования и подходов к тестированию
- Роль и обязанности тестировщика

В последовательных моделях разработки ПО на начальных этапах тестировщик обычно участвует в тестировании требований, анализе тестирования и проектировании тестов. Динамическое тестирование не выполняется на этих этапах, так как исполняемый код создается позже.

В некоторых итерационных и инкрементальных моделях разработки программного обеспечения подразумевается, что каждая итерация предоставляет работоспособный прототип или продукт с дополнительной функциональностью. Это означает, что в каждой итерации динамическое и статическое тестирование могут быть осуществлены на всех уровнях тестирования. Частая поставка инкрементов требует быстрой обратной связи и обширного регрессионного тестирования.

Гибкая методология разработки программного обеспечения допускает, что изменения могут вноситься на протяжении всего проекта. Ввиду этого в проектах, работающих по гибкой методологии, предпочитают упрощенную документацию рабочего продукта и обширную автоматизацию тестирования для того, чтобы сделать проще выполнение регрессионного тестирования. Также большую часть ручного тестирования стремятся выполнять с использованием

методик тестирования на основе опыта (см. раздел 4.4), не требующих обширных предварительных активностей, включающих анализ тестирования и проектирование тестов.

2.1.2 Жизненный цикл разработки программного обеспечения и лучшие практики тестирования

Эффективные практики тестирования, не зависящие от выбранной модели жизненного цикла разработки программного обеспечения, включают следующие:

- Каждой активности разработки программного обеспечения соответствует активность тестирования, таким образом все активности разработки программного обеспечения являются объектами контроля качества.
- У различных уровней тестирования (см. раздел 2.2.1) - различные и конкретные цели тестирования, что позволяет тестированию быть достаточно полным, но не избыточным.
- Анализ тестирования и проектирование тестов для конкретного уровня тестирования начинаются во время соответствующего этапа жизненного цикла разработки программного обеспечения, таким образом соблюдается принцип раннего тестирования (см. раздел 1.3).
- Тестировщики рецензируют рабочие продукты, как только появляются черновые версии этих документов, таким образом раннее тестирование и выявление дефектов обеспечивают стратегию сдвига влево (см. раздел 2.1.5).

2.1.3 Тестирование как движущая сила разработки программного обеспечения

Разработка на основе тестов, разработка через приемочное тестирование и разработка на основе поведения - одинаковые подходы к разработке, в которых тесты определены как направляющее разработку средство. Каждый из этих подходов применяет принцип раннего тестирования (см. раздел 1.3) и следует подходу сдвига влево (см. раздел 2.1.5), так как тесты разрабатываются до написания кода.

Разработка на основе тестов:

- Управляет кодированием при помощи тестовых сценариев (вместо масштабного проектирования программного обеспечения) (Beck, 2003)
- Сначала пишутся тесты, потом пишется соответствующий им код, а затем осуществляется рефакторинг тестов и кода

Разработка через приемочное тестирование (см. раздел 4.5.3):

- Получает тесты из критериев приемки как часть процесса проектирования системы (Gärtner 2011)
- Тесты пишутся до разработки даже части приложения, которое должно им соответствовать

Разработка на основе поведения:

- Отражает желаемое поведение приложения при помощи тестовых сценариев, написанных на легко понятном заинтересованным сторонам языке, в формате Дано/Когда/Затем (Chelimsky 2010)
- Потом тестовые сценарии автоматизируются.

Тесты могут продолжать существовать как автоматизированные тесты во всех этих подходах с целью гарантировать качество кода в случае будущих адаптаций и/или рефакторинга.

2.1.4 DevOps и тестирование

DevOps — это организационный подход, направленный на создание синергии путем совместной работы разработки (включая тестирование) и эксплуатации для достижения совокупности общих целей. DevOps требует культурного сдвига внутри организации, чтобы устранить разрыв между разработкой (включая тестирование) и эксплуатацией, рассматривая их функции как равноценные. DevOps продвигает самостоятельность команд, быструю обратную связь, интегрированные наборы инструментальных средств и такие технические практики как непрерывная интеграция и непрерывная поставка. Это позволяет командам быстрее собирать, тестировать и выпускать высококачественный код при помощи конвейера поставки DevOps.

С точки зрения тестирования DevOps имеет следующие преимущества:

- Быстрая обратная связь по качеству кода и насколько внесенные изменения влияют на существующий код
- Непрерывная интеграция способствует подходу сдвига влево в тестировании (см. раздел 2.1.5), побуждая разработчиков предоставлять высококачественный код, сопровождая его компонентными тестами и статическим анализом
- Продвижение автоматизированных процессов, таких как непрерывная интеграция и непрерывная поставка, которые упрощают создание стабильных сред тестирования
- Расширение представления о нефункциональных качественных характеристиках (например, производительность, надежность)
- Автоматизация при помощи конвейера поставки уменьшает необходимость повторяющегося ручном тестировании
- Риск внесения регрессии минимизирован с учетом масштаба и диапазона автоматизированных регрессионных тестов

DevOps не лишен рисков и проблем, среди которых следующие:

- Конвейер поставки DevOps должен быть спроектирован и развернут
- Инструменты непрерывной интеграции и непрерывной поставки должны быть внедрены и поддерживаться
- Автоматизация тестирования требует дополнительных ресурсов и может быть непростой при развертывании и поддержке

Хотя DevOps предусматривает высокий уровень автоматизации тестирования, ручное тестирование, особенно с точки зрения пользователя, все еще необходимо.

2.1.5 Сдвиг влево

Принцип раннего тестирования (см. раздел 1.3) иногда называется сдвигом влево, потому что в этом подходе тестирование выполняется на ранних этапах жизненного цикла разработки программного обеспечения. Как правило сдвиг влево предполагает, что тестирование должно быть выполнено на ранних этапах (например, не ожидая появления кода или интеграции компонентов), но это не значит, что нужно пренебречь тестированием на более поздних этапах жизненного цикла разработки программного обеспечения.

Лучшие практики, поясняющие как достигнуть сдвига влево в тестировании, включают следующие:

- Рецензирование спецификации с точки зрения тестирования. Активности по рецензированию спецификаций часто находят потенциальные дефекты, такие как неоднозначность, неполнота и несоответствие
- Написание тестовых сценариев до написания кода и запуск кода в тестовой обвязке во время его написания

- Использование непрерывной интеграции, а еще лучше непрерывной поставки, так как они включают быструю обратную связь и автоматизированные компонентные тесты, сопровождающие код при его добавлении в репозиторий
- Выполнение статического анализа исходного кода до начала динамического тестирования как часть процесса автоматизации
- Выполнение нефункционального тестирования, начиная с уровня компонентного тестирования там, где это возможно. Это форма сдвига влево, так как обычно выполняют типы нефункционального тестирования на поздних этапах жизненного цикла разработки программного обеспечения, когда доступны вся система и типичная среда тестирования

Применение подхода сдвига влево может привести к дополнительным обучению, трудозатратам и/или расходам на ранних этапах процесса, но ожидается сокращение трудозатрат и/или расходов на более поздних этапах процесса.

Для подхода сдвига влево важно, чтобы заинтересованные стороны были убеждены и приняли эту концепцию.

2.1.6 Ретроспективы и улучшение процесса

Ретроспективы (также известные как “послепроектные совещания” и проектные ретроспективы) обычно проводятся в конце проекта, итерации, при достижении конкретного этапа проекта или в случае необходимости. График и организация ретроспектив зависит от выбранной модели жизненного цикла разработки программного обеспечения. На этих совещаниях участники (не только тестировщики, но и разработчики, архитекторы, менеджеры продукта, бизнес-аналитики) обсуждают:

- Что имело успех и должно быть использовано в дальнейшем?
- Что не имело успеха и может быть улучшено?
- Как внедрить улучшения и сохранить успехи в будущем?

Результаты должны быть записаны и обычно отражаются в итоговом отчете о тестировании (см. раздел 5.3.2). Ретроспективы имеют решающее значение для успешной организации непрерывного улучшения, и важно, чтобы любые рекомендуемые улучшения внедрялись.

Типичные преимущества для тестирования включают следующие:

- Увеличение эффективности/производительности (например, за счет внедрения предложений по улучшению процесса)
- Улучшение качества тестового обеспечения (например, за счет совместного рецензирования процессов тестирования)
- Командообразование и обучение (например, за счет возможности задавать вопросы и предлагать направления для улучшения)
- Улучшение качества базиса тестирования (например, за счет того, что недостатки в объеме и качестве требований могут быть назначены и устранены)
- Улучшение сотрудничества между разработкой и тестированием (например, за счет регулярного рецензирования и оптимизации совместной работы).

2.2 Уровни тестирования и типы тестирования

Уровни тестирования – это группы активностей тестирования, которые организованы и управляются как единое целое. Каждый уровень тестирования — это реализация процесса тестирования

программного обеспечения, находящегося на конкретном уровне разработки, начиная с отдельных модулей и компонентов и заканчивая целыми системами или, где применимо, системами систем.

Уровни тестирования связаны с другими активностями в рамках жизненного цикла разработки программного обеспечения. В последовательных моделях жизненного цикла разработки программного обеспечения уровни тестирования зачастую определены таким образом, что критерий выхода одного уровня является частью критерия входа для следующего за ним уровня. Для некоторых итерационных моделей это не применимо. Активности разработки могут охватывать несколько уровней тестирования. Уровни тестирования могут пересекаться по времени.

Типы тестирования – это группы активностей тестирования, направленных на тестирование заданных характеристиках качества программного обеспечения, и большая часть этих активностей тестирования может быть осуществлена на каждом уровне тестирования.

2.2.1 Уровни тестирования

В данной программе обучения описаны следующие пять уровней тестирования:

- **Компонентное тестирование** (также известное как модульное тестирование) фокусируется на тестировании компонентов по отдельности. Зачастую оно требует специфической поддержки, такой как тестовые обвязки или интегрированные среды модульного тестирования. Компонентное тестирование обычно выполняется разработчиками в их собственных окружениях.
- **Тестирование интеграции компонентов** (также известное как тестирование интеграции модулей) фокусируется на тестировании интерфейсов и взаимодействии между компонентами. Тестирование интеграции компонентов сильно зависит от выбора подхода к стратегии интеграции - снизу вверх, сверху вниз или методом “большого взрыва”.
- **Системное тестирование** фокусируется на поведении и возможностях системы или продукта целиком, часто включая функциональное тестирование сквозных задач и нефункциональное тестирование характеристик качества программного обеспечения. Предпочтительно проведение тестирования некоторых нефункциональных качественных характеристик (например, удобства использования) с использованием готовой системы и среды тестирования, близкой к эксплуатационной среде. Также возможно использование имитаций подсистем. Системное тестирование может осуществляться независимой командой тестирования и относится к спецификациям системы.
- **Системное интеграционное тестирование** фокусируется на тестировании интерфейсов тестируемой системы, взаимодействующих с другими системами и внешними сервисами. Системное интеграционное тестирование требует соответствующих тестовых сред, максимально похожих на среду эксплуатации.
- **Приемочное тестирование** фокусируется на проверке и демонстрации готовности системы к развертыванию, что означает удовлетворение системой пользовательских бизнес-потребностей. В идеале приемочное тестирование должно выполняться предполагаемыми пользователями. Основными формами приемочного тестирования являются: пользовательское приемочное тестирование, эксплуатационное приемочное тестирование,

контрактное и нормативное приемочное тестирование, альфа-тестирование и бета-тестирование.

Уровни тестирования во избежание пересечения активностей тестирования различаются по следующему расширяемому списку признаков:

- Объект тестирования
- Цели тестирования
- Базис тестирования
- Дефекты и отказы
- Подход и зоны ответственности

2.2.2 Типы тестирования

Существует много типов тестирования, которые могут быть применены в проектах. В данной программе обучения описаны следующие четыре типа тестирования:

Функциональное тестирование оценивает функции, которые компонент или система должны выполнять. Функции дают ответ на вопрос «что делает система». Основная цель функционального тестирования — это проверка функциональной полноты, функциональной правильности и функционального соответствия.

Нефункциональное тестирование оценивает признаки компонента или системы, отличные от функциональных характеристик. Нефункциональное тестирование – это проверка того, «насколько хорошо работает система». Основная цель - нефункционального тестирования — это проверка нефункциональных характеристик качества программного обеспечения. Стандарт ISO/IEC 25010 дает следующую классификацию нефункциональных качественных характеристик ПО:

- Производительность
- Совместимость
- Удобство использования
- Надежность
- Безопасность
- Сопровождаемость
- Переносимость

Иногда уместно начинать нефункциональное тестирование на ранних этапах жизненного цикла (например, как часть рецензирования, компонентного тестирования или системного тестирования). Многие нефункциональные тесты основаны на функциональных тестах, так как они используют одни и те же функции, но проверяют, что во время выполнения функции нефункциональные ограничения соблюдены (например, проверка, что функция выполняется за определенное время или что функция может быть перенесена на другую платформу). Несвоевременное обнаружение нефункциональных дефектов может быть угрозой успеха всего проекта. Для выполнения

нефункционального тестирования иногда необходимо очень специфичное тестовое окружение, например, специальные лаборатории для тестирования удобства использования.

Тестирование методом черного ящика (см. раздел 4.2) основано на поведении и внешней по отношению к объекту тестирования документации. Основная цель тестирования методом черного ящика — это проверка соответствия поведения системы ее спецификациям.

Тестирование методом белого ящика (см. раздел 4.3) основано на реализации системы или ее внутренней структуре (например, программном коде, архитектуре, принципах работы и потоках данных). Основная цель тестирования методом белого ящика - покрыть основную структуру тестами на приемлемом уровне.

Все четыре вышеупомянутые типы тестирования могут быть применены на всех уровнях тестирования, но цели будут отличаться для каждого уровня. Различные методы проектирования тестов используются для получения тестовых условий и тестовых сценариев для всех упомянутых типов тестирования.

2.2.3 Подтверждающее тестирование и регрессионное тестирование

Изменения вносятся в систему или компонент, чтобы улучшить ее, добавив новую функциональность, или исправить ее, устранив дефект. В таком случае тестирование должно включать подтверждающее тестирование и регрессионное тестирование.

Подтверждающее тестирование проверяет, что исходный дефект был успешно исправлен. В зависимости от степени риска тестирование может проводиться несколькими способами, среди которых:

- выполнение всех тестовых сценариев, завершившихся с ошибкой из-за дефекта
- добавление новых тестовых сценариев для покрытия исправляющих дефект изменений

Между тем в случае нехватки времени или финансирования при подтверждающем тестировании можно ограничиться выполнением шагов по воспроизведению сбоя, вызванного дефектом, и проверкой, что сбой не воспроизводится.

Регрессионное тестирование подтверждает, что внесенные в систему изменения не имели негативных последствий, включая исправление дефекта, уже прошедшего подтверждающее тестирование. Эти негативные последствия могут влиять на тот же компонент, в котором были сделаны изменения, на другие компоненты той же системы или даже на связанные системы. Регрессионное тестирование может не ограничиваться только самим объектом тестирования, но и распространяться на окружение. Целесообразно сначала провести анализ влияния для оптимизации объемов регрессионного тестирования. Анализ влияния показывает, какие части системы могут быть затронуты.

Регрессионные наборы тестов многократно выполняются и в общем случае количество регрессионных сценариев тестирования увеличивается с каждой итерацией или выпуском программного продукта, таким образом регрессионное тестирование - серьезный кандидат на автоматизацию. Автоматизация этих тестов должна начинаться на ранних этапах проекта. Там, где используется практика непрерывной интеграции, например, как в технологии DevOps (см. раздел

2.1.4), хорошей практикой считаются автоматизированные регрессионные тесты. В зависимости от ситуации могут включаться регрессионные тесты на различных уровнях тестирования.

Подтверждающее тестирование и/или регрессионное тестирование объекта тестирования необходимо проводить на всех тех уровнях тестирования, на которых были исправлены дефекты или внесены изменения.

2.3 Тестирование в период сопровождения

Существуют различные категории сопровождения - оно может быть корректирующим, адаптивным к изменениям в среде или улучшающим быстродействие, или сопровождаемость (см. ISO/IEC 14764). Таким образом сопровождение может включать в себя запланированные выпуски программного продукта/развертывания и незапланированные выпуски программного продукта/развертывания (срочные исправления). Анализ влияния может быть проведен до внесения изменения для принятия решения о целесообразности его внесения, основываясь на потенциальных последствиях для других областей системы. Тестирование изменений в системе после ввода в эксплуатацию включает оценку успеха реализации изменения и регрессионное тестирование в неизменяемых частях системы, которые составляют большую ее часть.

Объем тестирования в период сопровождения обычно зависит от:

- Степени риска изменения
- Размера существующей системы
- Размера внесенных изменений

Условия для сопровождения и тестирования в период сопровождения классифицируются следующим образом:

- Модификации, такие как запланированные улучшения (например, базирующиеся на графике выпусков), корректирующие изменения или срочные исправления.
- Обновления или миграции эксплуатационной среды, например, с одной платформы на другую, которые могут потребовать проведения тестов новой среды, а также измененного программного обеспечения, или тестов преобразования данных, когда данные будут перенесены в поддерживаемую систему из другого приложения.
- Вывод из эксплуатации, например, когда заканчивается жизненный цикл приложения. После снятия системы с эксплуатации может потребоваться тестирование архивирования данных в случае необходимости длительного периода хранения данных. Также может потребоваться тестирование процедур восстановления после архивирования для случаев длительных периодов хранения, если данные могут потребоваться во время периода хранения данных.

3. Статическое тестирование - 80 минут

Ключевые слова

Аномалия, динамическое тестирование, инспекция, неформальное рецензирование, разбор, рецензирование, статический анализ, статическое тестирование, технический анализ, формальное рецензирование

Цели обучения для главы 3:

3.1 Основы статического тестирования

FL-3.1.1 (K1) Узнавать типы продуктов, которые можно проанализировать с помощью разных методов статического тестирования
FL-3.1.2 (K2) Объяснить ценность статического тестирования
FL-3.1.3 (K2) Сравнить и противопоставить статическое и динамическое тестирование

3.2 Работа с обратной связью и процесс рецензирования

FL-3.2.1 (K1) Выявить преимущества ранней и частой обратной связи от заинтересованных сторон
FL-3.2.2 (K2) Обобщить мероприятия процесса рецензирования
FL-3.2.3 (K1) Запомнить обязанности основных участников процесса рецензирования
FL-3.2.4 (K2) Сравнить и противопоставить разные типы рецензирования
FL-3.2.5 (K1) Запомнить факторы, способствующие успешному рецензированию

3.1 Основы статического тестирования

В отличие от динамического тестирования для статического тестирования проверяемое программное обеспечение не нуждается в запуске. Код, спецификация процессов или системной архитектуры и другие рабочие оцениваются путем ручной экспертизы (например, во время рецензирования) или с помощью инструментов (например, статического анализа). Цели тестирования включают улучшение качества, обнаружение дефектов и оценку характеристик, таких как читаемость, полнота, корректность, тестируемость и последовательность. Статическое тестирование может быть применено как к верификации, так и к валидации.

Тестировщики, представители бизнеса и разработчики работают вместе во время составления примеров, совместного написания пользовательских историй и доработки бэклога, чтобы убедиться в том, что пользовательские истории и связанные рабочие продукты соответствуют установленным критериям, например, Определение Готовности (пункт 5.1.3). Техники рецензирования могут быть применены для обеспечения полноты и понятности пользовательских историй и наличия тестируемых критериев приемки. Задавая правильные вопросы, тестировщики изучают, оспаривают и помогают улучшать предлагаемые пользовательские истории.

Статический анализ помогает выявлять проблемы до этапа динамического тестирования, при этом часто требуя меньших усилий, так как нет необходимости писать тестовые сценарии и в основном используются инструменты (см. главу 6). Статический анализ часто включается в рамки непрерывной интеграции (см. секцию 2.1.4). В то время как в основном он используется для обнаружения конкретных дефектов кода, статический анализ также применяется для оценки сопровождаемости и безопасности. Проверка правописания и инструменты для улучшения читаемости являются другими примерами инструментов статического анализа.

3.1.1 Рабочие продукты, которые можно проверить с помощью статического тестирования

Практический любой рабочий продукт можно проверить с помощью статического тестирования. Сюда можно включить документы спецификации требований, исходный код, планы тестирования, тестовые сценарии, элементы бэклога продукта, концепции тестирования, проектная документация, контракты и модели.

Любой рабочий продукт, который можно прочитать и понять, может быть предметом рецензирования. Хотя для статического анализа продукты должны иметь структуру, по которой их можно будет проверить (например, модели, код или официальный текст).

К рабочим продуктам, которые не подходят для статического тестирования, можно отнести те, что трудно интерпретировать человеку и не следует анализировать инструментами (например, код, который исполняет третья сторона из-за правовых оснований).

3.1.2 Ценность статического тестирования

Статическое тестирование помогает находить дефекты на ранних стадиях цикла разработки по, что реализует принцип раннего тестирования (см. секцию 1.3). Оно также обнаруживает проблемы, которые не выявляются с помощью динамического тестирования (например, нечитаемый код,

паттерны проектирования не реализованы как следовало, дефекты в продуктах, не предназначенные для исполнения).

Статическое тестирование позволяет оценить качество продукта и приобрести в нем уверенность. Верификация задокументированных требований позволяет заинтересованным сторонам убедиться, что они отражают их реальные требования. Поскольку статическое тестирование может быть выполнено на ранних этапах жизненного цикла разработки программного обеспечения, можно создать общее понимание среди заинтересованных сторон, участвующих в этом процессе. Уровень взаимодействия также улучшится. По этой причине рекомендуется привлекать все заинтересованные стороны на этапе статического тестирования.

Несмотря на то, что проведение рецензирования может быть дорогостоящим, итоговая стоимость проекта в результате ниже, так как меньше сил и времени будет потрачено на нахождение дефектов на более последующих стадиях цикла разработки.

Статический анализ помогает находить дефекты в коде более эффективно, чем динамическое тестирование, снижая не только количество ошибок в коде, но и общее количество усилий, затраченных на разработку продукта.

3.1.3 Отличия статического тестирования от динамического

Статическое и динамическое тестирование дополняют друг друга. У них есть схожие цели, например, поддержка обнаружения дефектов в рабочих продуктах (см. раздел 1.1.1), но также есть и некоторые различия:

- как статическое, так и динамическое тестирование (с анализом ошибок) ведут к обнаружению дефектов, однако существуют типы дефектов, которые можно обнаружить только с помощью статического или динамического тестирования
- статическое тестирование обнаруживает дефекты напрямую, в то время как динамическое тестирование провоцирует ошибки в работе продукта, чтобы с помощью дальнейшего анализа можно было идентифицировать дефект
- с помощью статического тестирования можно найти дефекты в коде, который редко исполняется или до которого трудно добраться путем динамического тестирования
- статическое тестирование, в отличии от динамического, может быть применимо к продуктам, не требующим исполнения
- статическое тестирование может быть использовано для измерения характеристик качества (например, удобства обслуживания), которые не зависят от исполняемости кода, в то время как динамическое тестирование помогает в оценке характеристик качества (например, эффективности производительности), которые зависят от исполняемости кода.

Дефекты, которые легче и/или дешевле находить с помощью статического тестирования, включают в себя:

- дефекты в требованиях (например, непоследовательность, двусмысленность, противоречия, упущения, неточности, повторения)
- дефекты в дизайне (например, неэффективная структура базы данных, слабая модуляция)
- некоторые типы дефектов в коде (например, переменные с не присвоенными значениями, необъявленные переменные, недостижимый код, чрезмерная сложность кода)

- отклонения от стандартов (например, несоблюдение стандартов оформления кода)
- некорректная спецификация пользовательского интерфейса (например, несоответствие номеров, типов или порядка параметров)
- специфические типы уязвимостей в безопасности (например, переполнение буфера)
- пробелы или неточности в базисе для тестового покрытия (например, пропущены приемочные тесты).

3.2 Обратная связь и процесс рецензирования

3.2.1 Преимущества ранней и частой обратной связи от заинтересованных сторон

Ранняя и частая обратная связь позволяет раньше начать обсуждение касательно потенциальных проблем с качеством. Если заинтересованные стороны не принимают активного участия в процессе цикла разработки ПО, то разрабатываемый продукт может не соответствовать их изначальному или текущему видению. Неудачная реализация желаний заказчика может привести к дорогостоящим доработкам, пропущенным срокам, перекладыванию ответственности и даже к полной неудаче проекта.

Частая обратная связь на протяжении всего цикла разработки ПО поможет устранить недопонимания касательно требований и обеспечить понимание необходимых изменений и их раннее внедрение. Это поможет команде разработчиков лучше понять, разработкой какого продукта они занимаются. Это позволит им сосредоточить внимание на тех функциях, которые являются самыми важными для заказчика и которые оказывают наиболее положительное воздействие на выявленные риски.

3.2.2 Мероприятия процесса рецензирования

Стандарт ISO/IEC 20246 описывает универсальный процесс рецензирования, предоставляющий структурированную, но гибкую модель, которая может быть адаптирована под конкретную ситуацию. Если необходимо провести более формальное рецензирование, тогда нужно будет провести большее количество мероприятий для описанных задач.

Размер многих рабочих продуктов делает их слишком большими, чтобы их можно было покрыть одним рецензированием. Процесс рецензирования может быть запущен несколько раз, чтобы проанализировать весь продукт.

Мероприятия процесса рецензирования включают:

- **Планирование.** Во время фазы планирования должен быть определен объем рецензирования, включающий цель, рабочий продукт, который будет проанализирован, качественные характеристики, которые будут оцениваться, области, на которых следует сосредоточиться, критерии завершения, вспомогательная информация, такая как стандарты, усилия и временные рамки для обзора
- **Инициирование рецензирования.** Во время этой фазы стоит задача убедиться в том, что все и всё готово к началу рецензирования. А именно, убедиться, что все рецензенты имеют доступ к оцениваемому продукту, понимают свои роли и обязанности и получили всё необходимое для проведения рецензирования

- **Индивидуальное рецензирование.** Каждый рецензент проводит индивидуальную оценку качества продукта и идентифицирует аномалии, дает рекомендации и задает вопросы, применяя техники рецензирования (например, рецензирование на основе чек-листа, рецензирование на основе сценариев). Стандарт ISO/IEC 20246 предоставляет более подробную информацию о различных техниках рецензирования. Рецензенты регистрируют все обнаруженные аномалии, рекомендации и вопросы.
- **Коммуникация и анализ.** Поскольку аномалии, выявленные во время рецензирования, не обязательно являются дефектами, все эти аномалии должны быть проанализированы и обсуждены. Для каждой аномалии необходимо принять решение о ее статусе, владельце и необходимых действиях. Обычно это делается на совещании по рецензированию, во время которого участники также определяют уровень качества рассматриваемого рабочего продукта и необходимые последующие действия. Возможно, потребуется дополнительный этап рецензирования.
- **Исправление и отчетность.** Для каждого дефекта должен быть создан отчет о дефекте, чтобы можно было осуществить корректирующие действия. После достижения критериев завершения, рабочий продукт может быть принят. Результаты рецензирования сообщаются.

3.2.3 Роли и обязанности в процессе рецензирования

В процессе рецензирования принимают участие различные заинтересованные стороны, которые могут выполнять несколько ролей. Основные роли и их обязанности:

- **Руководитель** - определяет, что будет подвергаться рецензированию, и предоставляет ресурсы, такие как персонал и время, для проведения рецензирования
- **Автор** - создает и исправляет рецензируемый рабочий продукт
- **Модератор** (также известный как фасилитатор) - обеспечивает эффективное проведение совещаний по рецензированию, включая посредничество, управление временем и безопасную среду для рецензирования, в которой каждый может свободно высказываться
- **Секретарь** (также известный как регистратор) - собирает аномалии от рецензентов и регистрирует информацию о процессе рецензирования, такую как принятые решения и новые аномалии, найденные во время совещания по рецензированию
- **Рецензент** - выполняет рецензирование. Рецензентом может быть коллега, работающий над проектом, эксперт в данной области или любое другое заинтересованное лицо
- **Лидер рецензирования** - несет общую ответственность за процесс, например, решает, кто будет вовлечен, и организовывает время и место проведения рецензирования.

Возможны и другие, более подробные роли, описанные в стандарте ISO/IEC 20246.

3.2.4 Виды рецензирования

Существует множество типов рецензирования разной степени формальности, от неформального до формального рецензирования. Требуемая степень формальности зависит от таких факторов, как методология жизненного цикла разработки, зрелость процесса разработки, критичность и сложность

подлежащего обзору рабочего продукта, наличие правовых или регуляторных требований и необходимость в аудиторской записи. Один и тот же рабочий продукт может быть подвергнут разным видам рецензирования, например, сначала неформальному, а затем более формальному.

Выбор правильного типа рецензирования является ключевым для достижения требуемых целей рецензирования (см. раздел 3.2.5). Выбор основан не только на целях, но также на факторах, таких как потребности проекта, доступные ресурсы, тип рабочего продукта и риски, деловая сфера и корпоративная культура.

Некоторые распространенные типы рецензирования включают:

- **Неформальное рецензирование**. Неформальное рецензирование не следует определенному процессу и не требуют формального документирования результатов. Основная цель - обнаружение аномалий.
- **Разбор**. Разбор, который проводит автор, может служить различным целям, таким как оценка качества и установление доверия к рабочему продукту, обучение рецензентов, достижение согласия, генерация новых идей, мотивация и возможность для авторов улучшить продукт, а также обнаружение аномалий. Рецензенты могут провести индивидуальное рецензирование перед разбором, но это не обязательно.
- **Технический анализ**. Технический анализ выполняется технически квалифицированными рецензентами под руководством модератора. Цели технического анализа включают достижение согласия и принятие решений относительно технической проблемы, а также обнаружение аномалий, оценку качества и установление доверия к рабочему продукту, генерацию новых идей, мотивацию и создание возможностей для авторов улучшить работу.
- **Инспекция.** Поскольку инспекция является наиболее формальным типом рецензирования, она следует полному общему процессу (см. раздел 3.2.2.). Основная цель - найти максимальное количество аномалий. Другие цели включают оценку качества, установление доверия к рабочему продукту, мотивацию и создание возможностей для авторов улучшить работу. Собираются и используются метрики для улучшения цикла разработки программного обеспечения, включая процесс инспекции. Во время инспекций автор не может быть лидером или секретарем процесса рецензирования.

3.2.5 Факторы успеха рецензирования

Существует несколько факторов, которые определяют успех рецензирования:

- Определение четких целей и измеримых критериев завершения. Оценка участников никогда не должна быть целью
- Выбор соответствующего типа рецензирования для достижения поставленных целей и соответствия типу рабочего продукта, рецензентам, потребностям проекта и контексту
- Проведение рецензирования по небольшим фрагментам, чтобы обозреватели не теряли концентрацию во время отдельного этапа рецензирования и/или собрания по рецензированию (если оно проводится)
- Предоставление обратной связи по рецензированию заинтересованным сторонам и авторам, чтобы они могли улучшить продукт и свою деятельность (см. раздел 3.2.1)

- Предоставление достаточного времени участникам для подготовки к процессу рецензирования
- Поддержка руководства в процессе рецензирования
- Включение рецензирования в культуру организации для поощрения обучения и улучшения процесса
- Проведение соответствующей подготовки всех участников, чтобы они понимали, как исполнить свою роль
- Поддержка проведения встреч.

4. Анализ и проектирование тестов — 390 минут

Ключевые слова

Анализ граничных значений, исследовательское тестирование, критерии приемки, метод проектирования тестов, метод проектирования тестов на основе опыта, подход к тестированию, основанный на совместной работе, покрытие, покрытие ветвей, покрытие операторов, предположение об ошибках, разработка тестов методом белого ящика, разработка через приемочное тестирование, тестирование методом черного ящика, тестирование на основе чек-листов, тестирование с помощью таблицы решений, тестирование таблицы переходов, эквивалентное разбиение, элемент покрытия

Цели обучения для главы 4:

4.1 Методы тестирования. Общие сведения

FL-4.1.1 (K2) Выделить различие между тестированием методом черного ящика, тестированием методом белого ящика и тестированием на основе опыта.

4.2 Тестирование методом черного ящика

FL-4.2.1 (K3) Использовать метод эквивалентного разбиения для создания тестовых сценариев
FL-4.2.2 (K3) Использовать метод анализа граничных значений для создания тестовых сценариев
FL-4.2.3 (K3) Использовать метод таблицы решений для создания тестовых сценариев
FL-4.2.4 (K3) Использовать метод таблицы переходов для создания тестовых сценариев

4.3 Тестирование методом белого ящика

FL-4.3.1 (K2) Объяснить, в чем заключается метод тестирования операторов
FL-4.3.2 (K2) Объяснить, в чем заключается метод тестирования ветвей
FL-4.3.3 (K2) Объяснить важность тестирования методом белого ящика

4.4 Тестирование на основе опыта

FL-4.4.1 (K2) Объяснить, в чем заключается метод предположения об ошибках
FL-4.4.2 (K2) Объяснить, в чем заключается метод исследовательского тестирования
FL-4.4.3 (K2) Объяснить, в чем заключается метод тестирования на основе чек-листов

4.5 Подходы к тестированию, основанные на совместной работе

FL-4.5.1 (K2) Объяснить, как создавать пользовательские истории через совместную работу с разработчиками и представителями заказчика
FL-4.5.2 (K2) Классифицировать различные способы по созданию критериев приемки
FL-4.5.3 (K2) Использовать разработку через приемочное тестирование для создания тестовых сценариев

4.1 Методы тестирования. Общие сведения

Методы тестирования помогают тестировщику в анализе (что тестировать) и проектировании тестов (как тестировать). С их помощью становится возможной систематическая разработка небольшого, но достаточного набора тестовых сценариев. В процессе анализа и проектирования тестов методы тестирования помогают тестировщику выявлять тестовые условия, элементы покрытия и определить тестовые данные. Более подробную информацию о методах проектирования тестов и соответствующих метриках можно найти в стандарте ISO/IEC/IEEE 29119-4 и по ссылкам из списка источников: Beizer (1990), Craig (2002), Copeland (2004), Koomen (2006), Jorgensen (2014), Ammann (2016), Forgács (2019).

В данной программе обучения методы тестирования подразделяются на методы черного ящика, методы белого ящика и методы, основанные на опыте.

Методы черного ящика (также известные как методы, основанные на спецификации) опираются на анализ заданного для объекта тестирования поведения безотносительно его внутреннего устройства. Таким образом, тестовые сценарии не зависят от программной реализации. Как следствие, когда меняется программная реализация, но не меняется требуемое поведение, тестовые сценарии остаются применимы.

Методы белого ящика (методы на основе структуры) опираются на анализ внутренней структуры объекта тестирования и процесса обработки данных. Поскольку тестовые сценарии зависят от того, как спроектировано программное обеспечение, они могут быть созданы только после завершения проектирования и/или программной реализации объекта тестирования.

Методы, основанные на опыте, по сути, задействуют знания и опыт тестировщиков для проектирования и реализации тестовых сценариев. Эффективность этих методов в значительной степени зависит от профессиональных навыков тестировщика. Методы, основанные на опыте, позволяют обнаруживать дефекты, которые могут быть пропущены при тестировании методами черного ящика и белого ящика. Таким образом, методы, основанные на опыте, дополняют методы черного ящика и белого ящика.

4.2 Методы черного ящика

В последующих разделах рассматриваются такие наиболее распространенные методы черного ящика, как:

- эквивалентное разбиение
- анализ граничных значений
- таблица решений
- таблица переходов

4.2.1 Эквивалентное разбиение

Эквивалентное разбиение подразумевает разделение данных на группы (так называемые классы эквивалентности), исходя из предположения, что все элементы одной группы обрабатываются объектом тестирования схожим образом. Теоретическое обоснование этого метода таково: если при выполнении тестового сценария обнаруживается дефект для одного из значений, относящихся к какому-либо классу эквивалентности, тот же дефект должен быть обнаружен и для любых других значений из того же класса. Следовательно, достаточно одного теста на каждый класс.

Классы эквивалентности можно задать для любого элемента данных, относящегося к объекту тестирования, будь то входные данные, выходные данные, элементы конфигурации, внутренние

значения, значения, привязанные ко времени, параметры интерфейса. Классы могут быть непрерывными и дискретными, упорядоченными и неупорядоченными, конечными и бесконечными. Классы не должны пересекаться, а также не должны быть пустыми.

Для несложного объекта тестирования эквивалентное разбиение может быть простым. Но на практике зачастую очень сложно понять, каким образом объект тестирования будет поступать с различными значениями. Именно поэтому разбиение нужно выполнять с большим вниманием.

Класс, содержащий позитивные значения, называется позитивным классом. Класс, содержащий негативные значения, называется негативным классом. Понимание негативных и позитивных значений может отличаться в разных командах и организациях. Например, позитивными значениями можно считать такие, которые объект тестирования должен обрабатывать, или такие, для которых обработка заложена в спецификации. Негативными значениями можно считать такие, которые должны быть проигнорированы или не могут быть приняты в обработку объектом тестирования, либо такие, для которых в тестовой спецификации не определено, как они должны обрабатываться.

В методе эквивалентного разбиения элементами покрытия являются классы эквивалентности. Для достижения 100%-го покрытия при использовании этого метода тестовые сценарии должны выполнять проверку всех заданных классов (включая негативные), то есть покрывать каждый класс как минимум одной проверкой. Покрытие вычисляется как отношение числа классов, на которых выполнен как минимум один тестовый сценарий, к общему числу заданных классов, выраженное в процентах.

Объекты тестирования часто имеют множественные наборы классов эквивалентности (например, объекты тестирования с более чем одним входным параметром). Это означает, что отдельно взятый тестовый сценарий будет покрывать классы эквивалентности из разных наборов. Самый простой критерий покрытия в таком случае называется «покрытием каждого варианта» (Ammann, 2016). Покрытие каждого варианта предполагает, что тестовые сценарии выполняются для каждого класса из каждого набора классов по меньшей мере один раз. При этом не берутся во внимание комбинации различных классов.

4.2.2 Анализ граничных значений

Анализ граничных значений — это метод, в котором проверка выполняется для границ классов эквивалентности. Отсюда следует, что данный метод может использоваться только для упорядоченных классов. Граничными значениями класса являются его минимальное и максимальное значения. В данном методе принимается, что если два элемента принадлежат к одному классу, то и все элементы, расположенные между ними, принадлежат к тому же классу.

Граничные значения важны по той причине, что разработчики с большей вероятностью допускают ошибки именно для них. Метод обнаруживает типичные дефекты там, где программная реализация выходит за пределы значений, которые предполагаются в качестве граничных, либо граничные значения вообще отсутствуют.

Данная программа обучения охватывает два варианта упомянутого метода: с определением двух граничных значений и с определением трех граничных значений. Они отличаются количеством приходящихся на каждую границу элементов, для которых необходимо выполнить тест, чтобы достичь 100%-го покрытия.

В варианте метода с определением двух граничных значений (Craig, 2002; Myers, 2011) для каждого граничного значения берется два элемента покрытия: само граничное значение и наиболее близкое соседнее значение из смежного класса. Для достижения 100%-го покрытия в методе с парами

граничных значений тестовые сценарии должны выполняться на всех элементах покрытия, то есть для всех заданных граничных значений. Покрытие вычисляется как отношение числа граничных значений, для которых выполнялось тестирование, к общему числу заданных граничных значений, выраженное в процентах.

В варианте метода с определением трех граничных значений (Koomen, 2006; O'Regan, 2019) для каждого граничного значения берется три элемента покрытия: само граничное значение и два соседних значения. Для достижения 100%-го покрытия в методе с определением трех граничных значений тестовые сценарии должны выполняться на всех элементах покрытия, то есть для всех заданных граничных значений и для всех соседних к ним значений. Покрытие вычисляется как отношение числа значений, граничных и соседних к ним, для которых выполнялось тестирование, к общему числу таких значений, выраженное в процентах.

Вариант метода с определением трех граничных значений является более тщательным, чем вариант с двойками, так как он может обнаружить дефекты, которые последним могут быть упущены из виду. К примеру, если условие задано как «if (x≤10) …», а программная реализация выполнена неверно — как «if (x=10) …», то тестовые данные в методе с двойками (x = 10, x =11) не смогут обнаружить этот дефект. Однако данное x = 9, используемое в методе с тройками, наверняка обнаружит его.

4.2.3 Таблица решений

Таблицы решений используются для тестирования программной реализации требований к приложению (системе), где последние определяют какие комбинации условий приводят к каким результатам. Таблицы решений — это наглядный способ представления сложных логических правил, например, бизнес-правил.

При создании таблицы решений в системе определяются исходные условия и результирующие действия. Эти элементы формируют строки таблицы. Каждый столбец представляет собой правило, задающее уникальную комбинацию условий, а также относящиеся к нему действия. В таблицах с краткой записью все значения, представляющие условия и действия (кроме несущественных или нереализуемых; см. ниже), отображаются в виде логических значений (истина или ложь). В то же время в таблицах с подробной записью некоторые или все условия и действия могут содержать расширенные значения (например, диапазоны чисел, области эквивалентности, дискретные значения).

Условия обозначаются следующим образом: «T» («true», истина) обозначает выполнение условия. Символ «F» («false», ложь) обозначает невыполнение условия. Символ «–» означает, что содержание условия не оказывает влияния на результирующее действие. Обозначение «N/A» («Not Available» — недоступно) означает, что условие невозможно реализовать для данного правила. Символ «X» в блоке действий означает, что должна выдаваться ошибка. Пустая ячейка означает, что действие не произойдет. Могут использоваться и другие обозначения.

Полная таблица решений имеет достаточно столбцов для покрытия всех комбинаций условий. Таблица может быть упрощена посредством удаления столбцов, содержащих невыполнимые комбинации условий. Таблица также может быть сокращена посредством объединения столбцов, в которых отдельные условия не влияют на результат, в один столбец. Алгоритмы по минимизации таблиц решений выходят за рамки данной программы обучения.

В тестировании с помощью таблицы решений элементами покрытия являются столбцы, содержащие действительные, выполнимые комбинации условий. Для достижения 100%-го покрытия при использовании данного метода тестовые сценарии должны выполняться на всех таких

столбцах. Покрытие вычисляется как отношение числа столбцов, для которых выполнялось тестирование, к общему числу столбцов с выполнимыми условиями, выраженное в процентах.

Преимущество тестирования на основе таблицы решений заключается в том, что этот метод обеспечивает систематический подход к определению всех возможных комбинаций условий, некоторые из которых в противном случае могли бы быть упущены из виду. Он также позволяет найти пробелы или противоречия в требованиях. Если условий много, выполнение тестовых сценариев для всех правил таблицы может требовать большого времени, так как число правил растет экспоненциально с ростом числа условий. В таком случае, чтобы уменьшить число правил для тестирования, можно воспользоваться методом минимизации таблицы решений либо подходом, основанным на оценке рисков.

4.2.4 Таблица переходов

Диаграмма перехода состояний воспроизводит модель поведения системы, показывая возможные состояния и разрешенные переходы между этими состояниями. Любой переход вызывается событием, которое может к тому же сопровождаться ограничением. Подразумевается, что переходы происходят мгновенно и в некоторых случаях вызывают определенные действия программы. Типичный синтаксис, маркирующий переход, следующий: «событие [ограничение] / действие». Ограничения и действия могут быть опущены, если их нет или если тестировщик считает их неважными.

Таблица переходов — модель, эквивалентная диаграмме перехода состояний. Ее строки представляют собой состояния, а столбцы — события (вместе с контрольными условиями, если таковые имеются). Записи в таблице (ячейки) представляют собой переходы и содержат конечное состояние, а также результирующие действия, если они определены. В отличие от диаграммы состояний и переходов, таблица переходов однозначно отображает недействительные переходы, которые представлены пустыми ячейками.

Тестовый сценарий, основанный либо на диаграмме перехода состояний, либо на таблице переходов, обычно представлен последовательностью событий, которые приводят к последовательности переходов между состояниями (и к действиям, если необходимо). Один тестовый сценарий может покрывать (и обычно покрывает) несколько переходов состояний.

В тестировании перехода состояний существует много критериев, используемых для задания покрытия. Данная программа обучения рассматривает три из них.

При покрытии всех состояний элементами покрытия являются состояния. Для достижения 100%-го покрытия состояний тестовые сценарии должны обеспечивать проход по всем состояниям. Покрытие вычисляется как отношение числа пройденных состояний к общему числу состояний, выраженное в процентах.

При покрытии всех разрешенных переходов (или «одношаговом» покрытии) элементами покрытия являются однократные переходы из числа разрешенных. Для достижения 100%-го покрытия разрешенных переходов тестовые сценарии должны выполняться для всех разрешенных переходах. Покрытие вычисляется как отношение числа пройденных разрешенных переходов к их общему числу, выраженное в процентах.

При покрытии всех переходов элементами покрытия являются все переходы, имеющиеся в таблице переходов. Для достижения 100%-го покрытия переходов тестовые сценарии должны выполняться на всех разрешенных переходах, а также делать попытку пройти недействительные переходы. Тестирование только одного недействительного перехода в рамках каждого теста помогает избежать замаскированных дефектов - ситуации, в которой один дефект мешает обнаружить другой. Покрытие вычисляется как отношение числа переходов, как разрешенных, так и недействительных,

как пройденных, так и тех, для которых была предпринята попытка их пройти, к общему числу разрешенных и недействительных переходов; выраженное в процентах.

Покрытие всех состояний является менее эффективным по сравнению с покрытием разрешенных переходов, поскольку оно может быть выполнено без прохождения всех переходов. Покрытие разрешенных переходов — наиболее распространенный критерий покрытия. Обеспечение полного покрытия разрешенных переходов гарантирует и полное покрытие состояний. Обеспечение полного покрытия переходов гарантирует как полное покрытие состояний, так и полное покрытие разрешенных переходов и должно быть требованием необходимого минимума для программного обеспечения с критически важными функциями и программного обеспечения с повышенными требованиями к безопасности.

4.3 Методы белого ящика

Данный раздел уделяет внимание двум методам белого ящика, ориентированным на программный код, ввиду их популярности и простоты:

- тестирование операторов
- тестирование ветвей

Существуют и более сложные методы, которые используются — в некоторых средах с повышенными требованиями к безопасности и надежности либо средах с критически важными функциями — для достижения более тщательного покрытия кода. Существуют также методы белого ящика для тестирования на более высоких уровнях программной организации (например, тестирование с помощью API), а также методы, в которых покрытие не привязано к программному коду (например, в тестировании нейронной сети учитывается покрытие нейронов). Эти методы в данной программе обучения не рассматриваются.

4.3.1 Тестирование операторов и покрытие операторов

В тестировании операторов элементами покрытия являются операторы. При проектировании тестовых сценариев ставится цель по выполнению операторов в программном коде, пока не будет достигнут приемлемый уровень покрытия. Покрытие вычисляется как отношение числа операторов, для которых тестирование было выполнено, к общему числу операторов в программном коде, выраженное в процентах.

Достижение 100%-го покрытия означает, что все исполнимые операторы в программном коде были выполнены по крайней мере один раз. В частности, из этого следует, что будет выполнен каждый оператор, имеющий дефект, и это может выявить проблему или ошибку, вызываемую наличием такого дефекта. Однако не во всех случаях выполнение оператора в рамках тестового сценария обнаружит дефекты. Например, оно может не обнаружить дефекты, зависящие от данных (так, деление на ноль только тогда приведет к ошибке, когда делителю будет присвоено нулевое значение). Кроме того, 100% покрытие операторов не означает, что будут протестированы все логические альтернативы, так как, например, при этом могут выполняться не все ветви кода (см. раздел 4.3.2).

4.3.2 Тестирование ветвей и покрытие ветвей

Ветвь представляет собой передачу управления между двумя узлами в графе потока управления, которая показывает возможные последовательности выполнения операторов в исходном коде объекта тестирования. Передача управления может быть либо не вызванной никаким условием

(безусловной, то есть на линейном участке программы), либо вызванной каким-либо условием (условной, то есть как результат выбора альтернативы).

В тестировании ветвей элементами покрытия являются ветви, и при проектировании сценариев тестирования целью ставится выполнение ветвей в программном коде, пока не будет достигнут приемлемый уровень покрытия. Покрытие вычисляется как отношение числа ветвей, для которых тестирование было выполнено, к общему числу ветвей, выраженное в процентах.

Достижение 100% покрытия означает, что в рамках тестовых сценариев выполняются все ветви программного кода, вызываются они каким-либо условием или нет. Разбиение на условные ветви обычно происходит в зависимости от результата (истина/ложь) при выполнении оператора «if... then», либо оператора ветвления «switch/case», либо при выборе альтернативы по выходу из цикла или продолжению работы цикла. Однако не во всех случаях выполнение ветви в рамках тестового сценария обнаружит дефекты. Например, оно может не обнаружить дефекты, проявляющиеся только при специфической последовательности выполнения кода.

Покрытие ветвей является надмножеством по отношению к покрытию операторов. Это означает, что любой набор тестовых сценариев со 100% покрытием ветвей также будет иметь 100% покрытие операторов (но не наоборот).

4.3.3 Важность тестирования методом белого ящика

Принципиальное преимущество всех методов белого ящика заключается в том, что при тестировании во внимание берется полная программная реализация, что повышает выявляемость дефектов, даже при условии расплывчатой, устаревшей или неполной спецификации к программному обеспечению. Соответственно, недостатком этих методов является то, что при отсутствии части требований к программному обеспечению тестирование может не обнаружить дефекты или недоработки (Watson, 1996).

Методы тестирования белого ящика могут использоваться при статическом тестировании (например, при сухом прогоне кода). Они хорошо подходят для рецензирования кода, который еще не готов к запуску (Hetzel, 1988), а также псевдокода и другой высокоуровневой или иерархической логики, которая может быть представлена с помощью графа потока управления.

Применение исключительно тестирования методом черного ящика не позволяет измерять фактическое покрытие кода. Методы белого ящика обеспечивают объективное измерение покрытия и дают необходимую информацию для создания дополнительных тестов, позволяющих увеличить это покрытие, и, как следствие, повышают уверенность в написанном коде.

4.4 Тестирование на основе опыта

В последующих разделах рассматриваются такие общеупотребительные методы тестирования на основе опыта:

- предположение об ошибках
- исследовательское тестирование
- тестирование на основе чек-листов

4.4.1 Предположение об ошибках

Предположение об ошибках — это способ предотвращения ошибок, дефектов и отказов, основанный на знаниях тестировщика, таких как:

- знания о том, как приложение работало в прошлом

- типы ошибок, которые склонны допускать разработчики, и типы дефектов, которые могут проистекать из таких ошибок
- типы отказов, которые происходили в других или похожих приложениях

В большинстве случаев ошибки, дефекты и отказы могут относиться: к вводу данных (например, корректный ввод не принимаются; какие-либо параметры неверны или отсутствуют), к выводу (например, неверный формат, неправильный результат), к логике приложения (например, пропущенные сценарии, неправильный оператор), к вычислениям (например, неверный операнд, неверные вычисления), к интерфейсу (например, несоответствие параметров, несовместимые типы данных), к данным приложения (например, некорректная инициализация, неверный тип данных).

Методический подход, с помощью которого реализуется метод предположения об ошибках, заключается в воспроизведении сбоев. В этой методике от тестировщика требуется создать или получить список возможных ошибок, дефектов и отказов и спроектировать тесты, которые будут обнаруживать дефекты, связанные с обозначенными ошибками, выявлять обозначенные дефекты и провоцировать обозначенные отказы. Эти списки могут составляться на основе прошлого опыта, с помощью заведомо ошибочных или поврежденных данных, или на основе знаний о том, что приводит приложение к отказу.

См. Whittaker (2002), Whittaker (2003), Andrews (2006) для получения дальнейшей информации о методе предположения об ошибках и методике воспроизведении сбоев.

4.4.2 Исследовательское тестирование

В исследовательском тестировании тесты одновременно разрабатываются, выполняются и получают оценку результата в процессе того, как тестировщик изучает объект тестирования. Такое тестирование используется для того, чтобы узнать больше информации об объекте тестирования, исследовать его более глубоко с помощью прицельных тестов и создать тесты для не покрытых тестированием областей.

Исследовательское тестирование может проводиться в виде сеансов тестирования для того, чтобы организовать тестирование в структуру. При использовании такого подхода исследовательское тестирование выполняется в течение определенного промежутка времени. При этом тестировщик руководствуется концепцией тестирования, в которой содержатся цели тестирования. За сеансом тестирования обычно следует подведение итогов, заключающееся в обсуждении между тестировщиком и ключевыми лицами, для которых важны результаты такого сеанса. В данном подходе цели тестирования могут расцениваться как тестовые условия верхнего уровня. Элементы покрытия устанавливаются и тестируются в процессе сеанса. Тестировщик может вести протокол сеанса для того, чтобы задокументировать пройденные шаги и то, что им было обнаружено.

Исследовательское тестирование полезно в случаях, когда требования представлены мало или недостаточно, а также когда тестирование проводится в сжатые сроки. Исследовательское тестирование также полезно в качестве дополнения к более формальным методам тестирования. Исследовательское тестирование будет наиболее эффективно, если тестировщик обладает опытом, знаниями о предметной области и развитыми навыками, такими как аналитические навыки, любопытство и изобретательность (см. раздел 1.5.1).

Исследовательское тестирование может включать использование других методов тестирования (например, эквивалентного разбиения). Подробную информацию об исследовательском тестировании можно найти в (Kaner 1999, Whittaker 2009, Hendrickson 2013).

4.4.3 Тестирование на основе чек-листов

При тестировании по чек-листу тестировщик проектирует, реализует и выполняет тесты, покрывающие тестовые условия, указанные в чек-листе. Чек-листы могут составляться на основе опыта, понимания того, что важно для пользователя, или знаний о том, отчего и как программное обеспечение дает сбой. В чек-листы нежелательно включать элементы, которые могут быть проверены автоматически, элементы, которые больше подходят в качестве входных или выходных критериев, а также элементы, которые являются слишком общими (Brykczynski, 1999).

Элементы чек-листа чаще всего формулируются в форме вопроса. Необходимо иметь возможность проверить каждый элемент отдельно и напрямую. Элементы могут ссылаться на требования, свойства графического интерфейса, характеристики качества или другие формы тестовых условий. Чек-листы могут создаваться для удобства проведения различных видов тестирования, включая функциональное и нефункциональное (например, 10 эвристик для тестирования практичности программного обеспечения; Nielsen, 1994).

Некоторые записи в чек-листе могут постепенно утрачивать свою полезность, поскольку разработчики учатся избегать одних и тех же ошибок. Может возникнуть необходимость добавить новые записи, чтобы учесть особо критические обнаруженные дефекты. По этой причине чек-листы следует регулярно обновлять, основываясь на анализе дефектов. Однако следует делать это взвешенно, чтобы избежать их излишнего разрастания (Gawade, 2009).

В отсутствие детальных сценариев тестирование с помощью чек-листов помогает определить направления тестирования и обеспечить некоторую систематичность. Если чек-листы сформулированы в слишком общих чертах, то ход выполнения теста на практике, вероятно, будет варьироваться и таким образом увеличивать покрытие, но снижать стабильность воспроизводимости теста.

4.5 Подходы к тестированию на основе совместной работы

Каждый из обозначенных выше методов (см. разделы 4.2, 4.3, 4.4) имеет определенную цель, направленную на обнаружение дефектов. Тестирование как совместная работа, в свою очередь, преследует цель предотвращения дефектов за счет совместной работы и коммуникации.

4.5.1 Совместное создание пользовательских историй

Пользовательская история представляет собой отдельно взятую функцию, которая является полезной для пользователя или клиента программного обеспечения. У пользовательских историй есть три важные составляющие («три Си», Jeffries, 2000):

- карточка (Card) — информационный носитель, который содержит описание пользовательской истории (например, учетная карточка или запись на электронной доске)
- диалог (Conversation) — разъясняет, как программное обеспечение будет использоваться (в документальной или устной форме)
- подтверждение (Confirmation) — критерии приемки (см. раздел 4.5.2)

Наиболее общий формат пользовательской истории: «В качестве [роли], мне нужно [задача, которая должна быть выполнена], для того чтобы [итоговая польза для бизнеса в этой роли]», после чего следуют критерии приемки.

При коллективном создании пользовательской истории могут использоваться такие методы как мозговой штурм или составление диаграммы связей. Совместная работа позволяет команде

достигать общего понимания того, какой результат необходимо предоставить, учитывая различные точки зрения со стороны бизнеса, разработки и тестирования.

Пользовательская история считается удачной, если она: обладает независимостью, допускает обсуждение, приносит ценность (Valuable), позволяет планировать/оценить ресурсы на ее выполнение, имеет небольшой размер, может быть протестирована.

4.5.2 Критерии приемки

Критерии приемки в пользовательской истории — это условия, которым должна удовлетворять пользовательская история, чтобы ее могли одобрить заинтересованные стороны. В этом смысле можно рассматривать критерии приемки как условия, которые должны быть проверены тестированием. Критерии приемки обычно формируются как результат диалога (см. раздел 4.5.1).

Цель критериев приемки:

- определить область применения пользовательской истории
- выработать единое мнение среди ключевых лиц
- описать как позитивные, так и негативные сценарии
- послужить основой для приемочного тестирования пользовательских историй (см. раздел 4.5.3)
- сделать планирование и оценку ресурсов более точными

Существует несколько вариантов того, как можно описывать критерии приемки для пользовательской истории. Два наиболее типичных формата таковы:

- ориентация на сценарий (например, дано/когда/тогда — такой формат используется в разработке на основе поведения, см. раздел 2.1.3)
- ориентация на правила (например, список верификации, табличная форма, карта соответствия вход — выход)

В большинстве случаев критерии приемки могут быть задокументированы в одном из этих двух форматов. Однако команда может использовать и свой собственный формат, если это позволяет определить критерии четким и однозначным образом.

4.5.3 Разработка через приемочное тестирование

Разработка через приемочное тестирование — это подход опережающего проектирования тестов (см. раздел 2.1.3). Тестовые сценарии создаются до разработки пользовательской истории. Тестовые сценарии создаются членами команды с разными ролями, например, клиентами, разработчиками, тестировщиками (Adzic, 2009). Тестовые сценарии могут выполняться вручную или автоматически.

Первый шаг — это совместная работа над спецификацией, в ходе которой члены команды анализируют, обсуждают и записывают пользовательскую историю, а также ее критерии приемки, если они еще не были заданы. В ходе этого процесса должны быть найдены решения для обнаруженных недочетов, неоднозначностей и дефектов в пользовательской истории. Следующий шаг — создание тестовых сценариев, чем может заняться вся команда целиком или отдельный тестировщик. Тестовые сценарии основываются на критериях приемки и представляют собой примеры того, как приложение должно работать. Это поможет команде реализовать пользовательскую историю правильно.

Поскольку примеры и тесты – это одно и то же, эти термины часто взаимозаменяемы. В ходе проектирования тестов могут применяться методы, описанные в разделах 4.2, 4.3 и 4.4.

Как правило, первые сценарии — позитивные сценарии, описывающие корректное поведения без исключений или ошибок, и включающие последовательность выполненных действий, если все идет ожидаемым образом. После того как готовы позитивные тесты, команде следует провести негативное тестирование. Наконец, команде следует также покрыть тестами нефункциональные требования, или характеристики качества (например, производительность, практичность интерфейса). Тестовые сценарии должны быть сформулированы таким образом, чтобы все ключевые лица могли понять их смысл. Как правило, тестовые сценарии состоят из утверждений на естественном языке с добавлением необходимых предусловий (если – то), входных данных, постусловий.

Тестовые сценарии должны покрывать все аспекты пользовательской истории и не выходить за ее пределы. Однако, критерии приемки могут детальнее описывать некоторые вопросы, касающиеся истории. Кроме того, не должно быть двух сценариев, которые описывают одни и те же аспекты пользовательской истории.

Если тестовые сценарии переведены в формат, пригодный для использования в среде автоматизации тестирования, разработчики могут автоматизировать их, написав вспомогательный код в процессе реализации функционала пользовательской истории. Приемочные тесты в этом случае становятся практической реализацией требований.

5. Управление тестированием – 335 минут

Ключевые слова

Анализ рисков, вероятность рисков, итоговый отчет о тестировании, квадранты тестирования, контроль рисков, контроль тестирования, критерии входа, критерии выхода, мониторинг рисков, мониторинг тестирования, определение рисков, отчет о дефекте, отчет о ходе тестирования, оценка рисков, пирамида тестирования, план тестирования, планирование тестирования, подход к тестированию, риск, риск продукта, риск проекта, смягчение рисков, тестирование, основанное на рисках, управление дефектами, управление рисками, уровень риска

Цели обучения для главы 5:

5.1 Планирование тестирования

FL-5.1.1 (K2) Привести примеры цели и содержания плана тестирования
FL-5.1.2 (K1) Узнать, как тестировщик повышает ценность итерации и планирования выпуска
FL-5.1.3 (K2) Сравнить и сопоставить критерии входа и выхода.
FL-5.1.4 (K3) Использовать методы оценки для определения необходимых трудозатрат для тестирования
FL-5.1.5 (K3) Применять определение приоритетов тестовых сценариев
FL-5.1.6 (K1) Запомнить концепцию пирамиды тестирования
FL-5.1.7 (K2) Обобщить квадранты тестирования и их взаимосвязь с уровнями и типами тестирования.

5.2 Управление рисками

FL-5.2.1 (K1) Определить уровень рисков, используя вероятность и влияние рисков
FL-5.2.2 (K2) Выделить риски проекта и риски продукта.
FL-5.2.3 (K2) Объяснить как анализ рисков может повлиять на тщательность и объем тестирования
FL-5.2.4 (K2) Объяснить, какие меры могут быть приняты по результатам анализа рисков продукта

5.3 Мониторинг, контроль и завершение тестирования

FL-5.3.1 (K1) Запомнить метрики используемые для тестирования
FL-5.3.2 (K2) Обобщить цель, содержание и целевые аудитории отчетов о тестировании
FL-5.3.3 (K2) Привести примеры как сообщать о статусе тестирования

5.4 Управление конфигурацией

FL-5.4.1 (K2) Обобщить, как управление конфигурацией поддерживает тестирование

5.5 Управление дефектами

FL-5.5.1 (K3) Подготовить отчет о дефекте

5.1 Планирование тестирования

5.1.1 Цель и содержание плана тестирования

План тестирования описывает цели, ресурсы и процессы для проекта тестирования. План тестирования:

- Документирует средства и график достижения целей тестирования
- Помогает убедиться, что выполняемые активности тестирования будут соответствовать установленным критериям
- Служит средством коммуникации с членами команды и другими заинтересованными сторонами
- Демонстрирует, что тестирование будет проводиться в соответствии с существующими политикой тестирования и стратегией тестирования (или объясняет, почему тестирование будет отклоняться от них)

Планирование тестирования определяет мышление тестировщиков и заставляет их решать будущие проблемы, связанных с рисками, расписанием, людьми, инструментами, затратами, усилиями и т.д. Процесс подготовки плана тестирования — это полезный способ продумать усилия, необходимые для достижения целей проекта тестирования.

Обычно содержание плана тестирования включает:

- Контекст тестирования (например, область применения, цели, ограничения, базис тестирования)
- Допущения и ограничения проекта тестирования
- Заинтересованные стороны (например, роли, обязанности, отношение к тестированию, найму и обучению)
- Коммуникацию (например, формы и частота взаимодействия, шаблоны документации)
- Реестр рисков (например, риски продукта, риски проекта)
- Подход к тестированию (например, уровни тестирования, типы тестирования, методы тестирования, результаты тестирования, критерии входа и выхода, независимость тестирования, необходимые метрики, требования к тестовым данным, требования к тестовому окружению, отклонения от политики и стратегии тестирования организации)
- Бюджет и расписание

Более подробную информацию о плане тестирования и его содержании можно найти в стандарте ISO/IEC/IEEE 29119-3.

5.1.2 Вклад тестировщика в планирование итераций и выпуска

В итеративных жизненных циклах разработки программного обеспечения обычно выполняются два вида планирования: планирование выпуска версии продукта и планирование итераций.

Планирование выпуска рассматривает перспективы выпуска продукта, определяет и пересматривает набор задач продукта и может включать в себя детализацию больших пользовательских историй в коллекцию небольших пользовательских историй. Он также служит основой для подхода к тестированию и плана тестирования на всех итерациях. Тестировщики,

участвующие в планировании выпуска, участвуют в написании тестируемых пользовательских историй и критериев приемки (см. раздел 4.5), участвуют в анализе рисков проекта и качества (см. раздел 5.2), оценивают затраты на тестирование, связанные с пользовательскими историями (см. раздел 5.1.4), определяют подход к тестированию и планируют тестирование для выпуска.

Планирование итерации рассматривает перспективы завершения отдельной итерации и связано с набором задач итерации. Тестировщики, участвующие в планировании итерации, участвуют в детальном анализе рисков пользовательских историй, определяют тестируемость пользовательских историй, разбивают пользовательские истории на задачи (в частности, задачи тестирования), оценивают затраты на тестирование для всех задач тестирования, а также выявляют и уточняют функциональные и нефункциональные аспекты объекта тестирования.

5.1.3 Критерии входа и критерии выхода

Критерии входа определяют предварительные условия для осуществления запланированных работ. Если критерии входа не будут соблюдены, вполне вероятно, что запланированные работы окажутся более сложными, отнимающими много времени, дорогостоящими и рискованными. Критерии выхода определяют, что должно быть достигнуто для того, чтобы объявить запланированные работы завершенными. Критерии входа и критерии выхода должны быть определены для каждого уровня тестирования и будут отличаться в зависимости от целей тестирования.

Обычно критерии входа включают: доступность ресурсов (например, людей, инструментов, окружения, тестовых данных, бюджета, времени), доступность тестового обеспечения (например, базиса тестирования, тестируемых требований, пользовательских историй, тестовых сценариев) и начальный уровень качества тестируемого объекта (например, все тесты на дым прошли).

Обычно критерии выхода включают: метрики глубины (например, достигнутый уровень покрытия, количество неисправленных дефектов, плотность дефектов, количество неудачных тестовых сценариев) и критерии завершения (например, запланированные тесты выполнены, статическое тестирование выполнено, обо всех обнаруженных дефектах сообщено, все регрессионные тесты автоматизированы).

Недостаток времени или бюджета также может рассматриваться как допустимые критерии выхода. Даже при отсутствии других критериев выхода может быть приемлемым прекращение тестирования, если заинтересованные стороны рассмотрели и приняли на себя риск перехода в эксплуатацию без дальнейшего тестирования.

В разработке программного обеспечения по гибким методологиям критерии выхода часто называют критериями завершения, определяя объективные показатели команды для выпускаемого продукта. Критерии входа, которым должна соответствовать пользовательская история, чтобы начать разработку и/или тестирование, называются критериями готовности.

5.1.4 Методы оценки

Оценка затрат на тестирование включает в себя прогнозирование объема работы, связанной с тестированием, и необходимой для достижения целей проекта тестирования. Важно разъяснить заинтересованным сторонам, что оценка основана на ряде допущений и всегда имеет погрешности. Оценка для небольших задач обычно более точна, чем для крупных. Следовательно, при оценке большой задачи ее можно разбить на ряд более мелких задач, которые в свою очередь можно оценить.

В этой программе описываются следующие четыре метода оценки.

Оценка на основе коэффициентов. В этом методе, основанном на метриках, данные собираются по предыдущим проектам внутри организации, что позволяет получить «стандартные» коэффициенты для аналогичных проектов. Коэффициенты собственных проектов организации (например, взятые из исторических данных), как правило, являются наилучшим источником для использования в процессе оценки. Эти стандартные соотношения затем могут быть использованы для оценки затрат на тестирование нового проекта. Например, если в предыдущем проекте соотношение затрат на разработку и тестирование составляло 3:2, а в текущем проекте ожидается, что затраты на разработку составят 600 человеко-дней, то затраты на тестирование можно оценить в 400 человеко-дней.

Экстраполяция. В этом методе, основанном на метриках, измерения проводятся как можно раньше в текущем проекте для сбора данных. Имея достаточное количество наблюдений, затраты, необходимые для выполнения оставшейся работы, могут быть приблизительно оценены путем экстраполяции этих данных (обычно применяя математическую модель). Этот метод очень подходит для итеративных моделей жизненного цикла разработки программного обеспечения. Например, команда может экстраполировать затраты на тестирование в предстоящей итерации как усредненные затраты за последние три итерации.

Широкополосный метод Дельфи. В этом итеративном методе, основанном на экспертизе, эксперты делают оценки, основанные на опыте. Каждый эксперт по отдельности оценивает трудозатраты. Результаты собираются, и, если есть отклонения, выходящие за пределы согласованных границ, эксперты обсуждают свои текущие оценки. Затем изолировано каждого эксперта, просят сделать новую оценку с учетом обратной связи. Этот процесс повторяется до тех пор, пока не будет достигнут консенсус. Разновидность широкополосного метода Дельфи — это покер-планирование, широко используемого при разработке программного обеспечения гибкими методологиями. Во время покера планирования оценки обычно делаются с использованием карточек с цифрами, которые отражают объем трудозатрат.

Оценка по трем точкам. В этом методе, основанном на экспертизе, эксперты делают три оценки: наиболее оптимистичную оценку (a), наиболее вероятную оценку (m) и наиболее пессимистичную оценку (b). Окончательная оценка (E) — это их средневзвешенное арифметическое значение. В наиболее популярной версии этого метода оценка рассчитывается как E = (a + 4*m + b) / 6. Преимущество этого метода заключается в том, что он позволяет экспертам рассчитать погрешность измерения: SD = (b – a) / 6. Например, если оценки (в человеко-часах) следующие: a=6, m=9 и b=18, то окончательная оценка составляет 10±2 человеко-часа (т.е. от 8 до 12 человеко-часов), поскольку E = (6 + 4*9 + 18) / 6 = 10 и SD = (18 – 6) / 6 = 2.

Смотрите (Kan 2003, Koomen 2006, Westfall 2009) для этих и многих других методов оценки тестирования.

5.1.5 Определение приоритетов тестовых сценариев

Когда тестовые сценарии и процедуры тестирования определены и собраны в наборы тестов, эти наборы тестов располагаются в соответствии с расписанием тестирования, определяющее порядок, в котором они должны запускаться. При определении приоритетов тестовых сценариев учитываются различные факторы. Наиболее часто используемые следующие стратегии определения приоритетов тестовых сценариев:

- Определение приоритетов на основе рисков, когда порядок выполнения тестов основан на результатах анализа рисков (см. раздел 5.2.3). Сначала выполняются тестовые сценарии, покрывающие наиболее важные риски.

- Определение приоритетов на основе покрытия, когда порядок выполнения тестов основан на покрытии (например, покрытие операторов). Сначала выполняются тестовые сценарии, достигающие максимального покрытия. В другом варианте, называемом определением приоритетов дополнительного покрытия, первым выполняется тестовый сценарий, обеспечивающий максимальное покрытие; каждый последующий тестовый сценарий — это тот, который обеспечивает максимальное дополнительное покрытие.
- Определение приоритетов на основе требований, когда порядок выполнения тестов основан на приоритетах требований, прослеживаемых до соответствующих тестовых сценариев. Приоритеты требований определяются заинтересованными сторонами. Сначала выполняются тестовые сценарии, относящиеся к наиболее важным требованиям.

В идеальном случае тестовые сценарии упорядочиваются на основе их приоритетов, используя, например, одну из вышеупомянутых стратегий определения приоритетов. Однако эта практика может не работать, если тесты или тестируемые функции имеют зависимости. Если тестовый сценарий с более высоким приоритетом зависит от тестового сценария с более низким приоритетом, то сначала выполняется тестовый сценарий с более низким приоритетом.

Порядок выполнения тестов также должен учитывать доступность ресурсов. Например, необходимые инструменты тестирования, тестовые среды или сотрудники, которые могут быть доступны только в течение определенного периода времени.

5.1.6 Пирамида тестирования

Пирамида тестирования — это модель, показывающая, что разные тесты могут иметь разную степень детализации. Модель пирамиды тестирования помогает команде в автоматизации тестирования и распределении затрат на тестирование, показывая, что разные уровни автоматизации тестирования поддерживают разные цели. Уровни пирамиды представляют группы тестов. Чем выше уровень, тем ниже детализация и изоляция теста, а также низкая скорость выполнения. Тесты на нижнем уровне небольшие, изолированные, быстрые и проверяют небольшую часть функциональности, поэтому обычно их требуется много для достижения разумного покрытия. Тесты на верхнем уровне - сложные, высокоуровневые, сквозные. Как правило, эти высокоуровневые тесты выполняются медленнее, чем тесты на нижних уровнях, и проверяют большую часть функциональности, поэтому для достижения разумного покрытия требуется всего несколько. Количество и название уровней может отличаться. Например, первоначальная модель пирамиды тестирования (Cohn 2009) определяет три уровня: "модульные тесты", "сервисные тесты" и "тесты пользовательского интерфейса". Другая популярная модель определяет модульные (компонентные) тесты, интеграционные тесты (интеграции компонентов) и сквозные тесты. Также могут использоваться другие уровни (см. раздел 2.2.1).

5.1.7 Квадранты тестирования

Квадранты тестирования, определенные Брайаном Мариком (Marick 2003, Crispin 2008), группируют уровни тестирования с соответствующими типами тестирования, активностями, методами тестирования и продуктами работы при разработке программного обеспечения по гибким методологиям. Визуализация модели поддерживает управление тестированием, и позволяет убедиться, что все соответствующие типы и уровни тестирования включены в жизненный цикл разработки программного обеспечения, и понимать, что некоторые типы тестирования более актуальны для определенных уровней тестирования, чем другие. Эта модель также предоставляет способ различать и описывать типы тестирования всем заинтересованным сторонам, включая разработчиков, тестировщиков и представителей бизнеса.

В этой модели тесты могут быть ориентированы на бизнес или технологии. Также тесты могут поддерживать команду (т.е. направлять разработку) или критиковать продукт (т.е. оценивать его поведение в соответствии с ожиданиями). Сочетание этих двух точек зрения определяет четыре сектора:

- Квадрант Q1 (ориентация на технологии, поддержка команды). Этот квадрант содержит компонентное тестирование и тестирование интеграции компонентов. Эти тесты должны быть автоматизированы и включены в процесс непрерывной интеграции (CI).
- Квадрант Q2 (ориентация на бизнес, поддержка команды). Этот квадрант содержит функциональное тестирование, примеры, тестирование на основе пользовательских историй, прототипы пользовательского интерфейса, тестирование API и моделирование. Эти тесты проверяют критерии приемки и могут быть ручными или автоматизированными.
- Квадрант Q3 (ориентация на бизнес, критика продукта). Этот квадрант содержит исследовательское тестирование, тестирование практичности, пользовательское приемочное тестирование. Эти тесты ориентированы на пользователя и часто выполняются вручную.
- Квадрант Q4 (ориентация на технологии, критика продукта). Этот квадрант содержит тесты на дым и нефункциональное тестирование (за исключением тестирования практичности). Эти тесты часто автоматизированы.

5.2 Управление рисками

Организации сталкиваются с большим количеством внутренних и внешних факторов, которые привносят неопределенность в понимание того, когда и будут ли вообще достигнуты цели организации (ISO 31000). Управление рисками позволяет организациям увеличить вероятность достижения целей, улучшить качество своих продуктов и повысить уровень доверия со стороны заинтересованных сторон.

Основные виды деятельности по управлению рисками таковы:

- Анализ рисков (включающий определение и оценку рисков; см. Раздел 5.2.3)
- Контроль рисков (включающий смягчение рисков и мониторинг рисков; см. Раздел 5.2.4)

Подход к тестированию, при котором выбор активностей тестирования, определение приоритетов и управление производится исходя из анализа рисков и контроля за ними называется тестированием, основанным на рисках.

5.2.1 Определение и атрибуты риска

Риск — это потенциальное негативное событие, случайность, фактор или ситуация, возникновение которых может иметь неблагоприятные последствия. Риск характеризуется двумя показателями:

- Вероятность риска — вероятность, что событие, представляющее собой риск, наступит (значение больше нуля и меньше единицы)
- Влияние риска (ущерб) — результат наступления такого события

Эти два показателя выражают уровень риска, который является мерилом риска. Чем выше уровень риска, тем важнее справиться с ним.

5.2.2 Риски проекта и риски продукта

В тестировании программного обеспечения, как правило, вызывают озабоченность два типа рисков: риски проекта и риски продукта.

Риски проекта связаны с управлением проектом и контролем над ним. Риски проекта включают:

- Организационные проблемы (например, задержки поставки рабочих продуктов, неточные оценки, сокращение бюджета)
- Проблемы, связанные с человеческими ресурсами (например, недостаточные навыки, конфликты, проблемы коммуникации, сокращение штата)
- Технические проблемы (например, расползание рамок проекта, некачественный инструментарий)
- Проблемы с поставщиками (например, несостоявшаяся поставка от сторонней организации, банкротство поставщика)

Риски продукта связаны с характеристиками качества продукта (например, описанными в модели качества ISO 25010). Примеры рисков продукта включают: пропущенный или неверно работающий функционал, неверные вычисления, ошибки, проявляющиеся при выполнении программы, неудовлетворительное качество архитектуры, неэффективные алгоритмы, неприемлемое время отклика, неудовлетворительное качество реализации пользовательского интерфейса, уязвимости безопасности. Если наступают события, связанные с рисками продукта, то негативные последствия могут быть различны:

- Недовольство пользователей
- Падение дохода доверия, репутации
- Урон третьим лицам
- Высокие затраты на сопровождение программного обеспечения, высокая нагрузка на подразделение техподдержки
- Наказание за нарушение законодательства (включая уголовное)
- В крайних случаях — физический ущерб, травмы и даже смерть

5.2.3 Анализ рисков продукта

С точки зрения тестирования анализ рисков продукта нацелен на повышение осведомленности о рисках продукта, что позволяет направить тестирование на снижение остаточного уровня этих рисков. В идеальном случае анализ рисков продукта начинается на ранней стадии жизненного цикла разработки.

Анализ рисков продукта включает определение рисков и оценку рисков. Определение рисков заключается в составлении списка рисков. Заинтересованные стороны определяют риски, используя различные методики и инструменты, например, мозговой штурм, совещания, опросы, составление причинно-следственных диаграмм. Оценка рисков включает: категоризацию установленных рисков, определение их вероятности, влияния и уровня риска, определение приоритетов, предположения о способах устранения рисков. Категоризация помогает в выборе действий по смягчению и предотвращению, поскольку, как правило, риски одной и той же категории требуют похожего подхода к их устранению.

В оценке рисков могут использоваться количественный или качественный подход, а также их сочетание. При количественном подходе уровень риска вычисляется как произведение вероятности риска и влияния риска. При качественном подходе уровень риска определяется с помощью матрицы рисков.

Анализ рисков продукта может влиять на тщательность и охват тестирования. Его результаты используются для следующих задач:

- Определение охвата предстоящего тестирования
- Определение конкретных уровней тестирования и выбор типов тестирования, которые необходимо произвести
- Определение методов тестирования и покрытия, которого необходимо достичь
- Оценка затрат на тестирование, необходимых для каждой задачи
- Определение приоритетов тестирования с целью обнаружения критических дефектов на самой ранней возможной стадии
- Решение о необходимости дополнительных работ для уменьшения риска

5.2.4 Контроль рисков продукта

Контроль рисков продукта включает все возможные меры, которые принимаются в отношении установленных и оцененных рисков продукта. Контроль рисков продукта состоит из смягчения рисков и мониторинга рисков. Смягчение рисков подразумевает исполнение шагов, предложенных в ходе оценки рисков, для уменьшения уровня риска. Цель мониторинга рисков — проконтролировать, что действия по смягчению рисков эффективны, получить дальнейшую информацию для улучшения оценки рисков и определить другие возникающие риски.

Что касается контроля за рисками продукта, как только риск был проанализирован, возможны несколько вариантов ответных действий, например, смягчение риска благодаря тестированию, принятие риска, передача риска или запасной план (Veenendaal, 2012). Действия, направленные на смягчение рисков продукта через тестирование, включают:

- Подбор тестировщиков с необходимыми для данного типа риска опытом и навыками,
- Обеспечение соответствующего уровня независимости тестирования,
- Проведение рецензирования и выполнение статического анализа,
- Применение соответствующих методов тестирования и уровней покрытия,
- Применение соответствующих типов тестирования, направленных на затрагиваемые характеристики качества,
- Выполнение динамического тестирования, включая регрессионное.

5.3 Контроль и мониторинг тестирования. Завершение тестирования

Мониторинг тестирования связан со сбором информации о тестировании. Данная информация используется для оценки хода тестирования и для количественного определения того, достигнуты ли критерии выхода либо завершены ли задачи тестирования, связанные с критериями выхода, такие как достижение целевого покрытия рисков продукта, требований и критериев приемки.

В контроле тестирования используется информация мониторинга тестирования для руководства процессом в форме управляющих директив и осуществления необходимых корректирующих действий с целью обеспечения наиболее эффективного и результативного тестирования. Примеры управляющих директив включают:

- Повторное определение приоритетов тестов в случае, если установленный риск возник и представляет реальную проблему
- Переоценку элемента тестирования для соответствия критериям входа и выхода в случае его доработки

- Корректировку графика тестирования с учетом задержки в развертывании среды тестирования
- Добавление новых ресурсов, когда и где это необходимо

На стадии завершения тестирования собираются данные по проведенным работам с целью обобщения опыта, тестовой модели, артефактов тестирования и другой важной информации. Работы по завершению тестирования проводятся в моменты прохождения ключевых отметок проекта, таких как завершение тестирования какого-либо уровня, завершение итерации (при использовании гибкой методологии), завершение проекта тестирования (или его прекращение, выпуск в эксплуатацию комплекса программных средств, завершение выпуска версии с исправлением ошибок.

5.3.1 Метрики тестирования

Метрики собираются для того, чтобы зафиксировать прогресс в ходе тестирования согласно запланированному графику и бюджету, а также для оценки текущего уровня качества объекта тестирования и оценки эффективности активностей тестирования в соответствии с целями или планами итерации.

Мониторинг тестирования предполагает сбор разнообразных метрик, которые помогают процессам контроля тестирования и завершения тестирования.

Общепринятые метрики тестирования включают:

- Показатели хода выполнения проекта (например, завершение задач, использование ресурсов, затраты на тестирование)
- Показатели хода тестирования (например, ход реализации тестовых сценариев, ход подготовки среды тестирования; количество выполненных и невыполненных тестовых сценариев, тест пройден / не пройден, время выполнения тестов)
- Показатели качества продукта (например, доступность, время отклика, среднее время наработки на отказ)
- Метрики дефектов (например, количество и приоритет найденных и исправленных дефектов, плотность дефектов, процент обнаружения дефектов)
- Метрики рисков (например, остаточный уровень риска)
- Метрики покрытия (например, покрытие требований, кода)
- Стоимостные показатели (например, стоимость тестирования, организационные затраты на качество)

5.3.2 Цели, содержание отчетов о тестировании и их целевая аудитория

Цель отчетности по тестированию заключается в обобщении и сообщении информации во время и после тестирования. Отчеты о ходе тестирования помогают вести текущий контроль тестирования и должны предоставлять достаточно информации для внесения корректировок в график тестирования, ресурсы и план тестирования, когда такие изменения необходимы ввиду отклонений от плана или изменившихся обстоятельств. Отчеты о завершении тестирования обобщают информацию о конкретной стадии тестирования (например, об уровне тестирования, цикле тестирования, итерации) и могут содержать сведения, полезные для последующего тестирования.

В течение мониторинга и контроля тестирования команда тестировщиков создает отчеты о ходе тестирования для заинтересованных сторон, чтобы держать их в курсе. Отчеты о ходе тестирования обычно создаются регулярно (например, ежедневно, еженедельно и т. п.) и включают:

- Период тестирования
- Отметку о ходе прогресса (например, с опережением графика или с отставанием) с указанием существенных отклонений
- Препятствия в тестировании и способы их обхода (временные решения)
- Метрики тестирования (см. Раздел 5.3.1 для примера)
- Новые и изменившиеся риски в течение периода тестирования
- Объем тестирования на будущий период

Отчет о завершении тестирования готовится во время завершения тестирования, когда завершается проект, уровень тестирования или тип тестирования, при этом могут быть достигнуты критерии выхода. Этот отчет использует как данные из отчетов о ходе тестирования, так и другие данные. Типичный отчет о завершении тестирования включает:

- Резюме по тестированию
- Оценка тестирования и качества продукта на основе исходного плана тестирования (то есть целей тестирования и критериев выхода)
- Отклонения от плана тестирования (например, разница с запланированным графиком, разница в продолжительности, в затратах)
- Препятствия в тестировании и временные решения для них
- Метрики тестирования на основе отчетов о ходе тестирования
- Риски, которые не были смягчены или устранены, количество неисправленных дефектов
- Полученные уроки, относящиеся к тестированию

Для разной целевой аудитории в отчетах требуется различная информация, что влияет на степень формализации отчетности и ее частоту. Доведение отчетности о ходе тестирования до других членов команды обычно является более частым и неформальным, в то время как отчетность о тестировании по завершении проекта должна следовать установленному шаблону и составляться единожды.

Стандарт ISO/IEC/IEEE 29119-3 включает шаблоны и примеры отчетов о ходе тестирования (так называемые отчеты о статусе тестирования), а также отчетов о завершении тестирования.

5.3.3 Донесение информации о текущем статусе тестирования

Наиболее подходящий способ донесения информации о статусе тестирования может варьироваться в зависимости от соображений руководителей тестирования, стратегий тестирования внутри организации, нормативных документов и, в случае самоорганизующихся команд, от самих команд (см. раздел 1.5.2). Возможные варианты включают:

- Устную коммуникацию с членами команды и другими ключевыми лицами
- Сводные таблицы (например, сводные таблицы по процессам непрерывной интеграции и непрерывного развертывания, доску задач, диаграммы «выгорания задач»)
- Электронные каналы коммуникации (например, электронную почту, чат)
- Документацию онлайн
- Официальные отчеты о тестировании (см. Раздел 5.3.2)

Может использоваться один или несколько из этих вариантов. Наиболее формальный способ коммуникации уместен для распределенных команд, где не всегда возможно общение лицом к лицу из-за географической удаленности и разницы в часовых поясах. Как правило, ключевые лица заинтересованы в разных типах информации, поэтому процесс коммуникации следует адаптировать.

5.4 Управление конфигурацией

В процессах тестирования управление конфигурацией обеспечивает порядок того, как идентифицировать элементы конфигурации, а также их контроль и отслеживание; такие элементы включают рабочие продукты: планы тестирования, стратегии тестирования, тестовые условия, тестовые сценарии, автоматизированные тестовые сценарии, результаты тестирования, протоколы тестирования и отчеты о тестировании.

Для сложных элементов конфигурации (например, для среды тестирования) в процессе управления ведут запись того, из каких элементов состоит данный элемент, а также связей между ними и их версии. Если элемент конфигурации утвержден для тестирования, он становится базовой (опорной) версией и может быть изменен только через процесс внесения изменений.

Если создается новая базовая версия, процесс управления подразумевает фиксацию изменений в элементах конфигурации. При этом остается возможность вернуться к предыдущей базовой версии для воспроизведения результатов предыдущего тестирования.

С целью надлежащей поддержки тестирования процесс управления конфигурацией обеспечивает следующее:

- Все элементы конфигурации, включая элементы тестирования (отдельные части объекта тестирования), идентифицированы однозначным образом, их версия контролируется, а изменения в них отслеживаются; они связаны с другими элементами конфигурации таким образом, который обеспечивает отслеживание на протяжении всего процесса тестирования.
- Все установленные элементы документации и программного обеспечения однозначно идентифицированы в документации по тестированию.

Непрерывные процессы интеграции, поставки, развертывания и связанное с ними тестирование обычно реализуются в рамках автоматизированного конвейера (см. раздел 2.1.4), в который также обычно включают и управление конфигурацией.

5.5 Управление дефектами

Поскольку нахождение дефектов — одна из основных целей тестирования, налаженный процесс управления дефектами крайне важен. И, хотя здесь упоминаются «дефекты», аномалии, которые были обнаружены, они могут оказаться как настоящими дефектами, так и чем-то другим (например, ложным срабатыванием, целенаправленным внесением изменения) — такие вопросы исправляются в процессе работы с отчетами о дефектах. Отчетность об аномалиях может предоставляться на любой стадии жизненного цикла разработки программного обеспечения, и ее форма зависит от него. Как минимум процесс управления дефектами включает рабочую процедуру по обращению с отдельными аномалиями, начиная с их обнаружения до закрытия, а также правила их классификации. Эта процедура обычно заключается в протоколировании обнаруженных аномалий, их анализе и классификации, решении о надлежащих мерах реагирования, таких как исправление аномалии или оставление ее как есть, и, наконец, закрытии отчета о дефекте. Процессу управления дефектами должны следовать все вовлеченные в него стороны. Рекомендуется поступать похожим образом с дефектами, полученными в результате статического тестирования (особенно статического анализа).

Типичный отчет о дефекте преследует следующие цели:

- Предоставление лицам, ответственным за управление и исправление обнаруженных дефектов, достаточной информации по исправлению

- Предоставление средств отслеживания качества рабочего продукта
- Предоставление идей улучшения разработки и процесса тестирования

Отчет о дефекте, составленный во время динамического тестирования, обычно включает:

- Уникальный идентификатор дефекта
- Заголовок с кратким описанием обнаруженной аномалии
- Дату, когда произошла аномалия, название организации, выпускающей отчет, автора и его роль
- Идентификационные данные объекта тестирования и среды тестирования
- Контекст, связанный с дефектом (например, тестовый сценарий, который был выполнен, наименование производимых работ по тестированию, этап жизненного цикла разработки программного обеспечения и другую актуальную информацию, например, какой использовался метод тестирования, чек-лист или тестовые данные)
- Описание отказа с целью его воспроизведения и разрешения, включая шаги, с помощью которых была обнаружена аномалия, относящиеся к ней протоколы тестирования, выгрузки базы данных, снимки экрана, записи звука/видео
- Ожидаемые и фактические результаты
- Влияние критичности дефекта (степень серьезности) на заинтересованные сторон или проверяемые требования
- Приоритет исправления
- Текущий статус дефекта (например, открыт, отложен, дубликат, ожидает исправления, ожидает проверки, открыт повторно, закрыт, отменен)
- Ссылки (например, на тестовый сценарий)

Некоторые из этих данных могут добавляться автоматически при использовании инструментов управления дефектами (например, идентификатор, дата, автор и начальный статус). Шаблоны документов, такие как отчет о дефекте и примеры таких отчетов, можно найти в стандарте ISO/IEC/IEEE 29119-3, где отчеты о дефекте именуются отчетами об инциденте.

6. Инструменты тестирования - 20 минут

Ключевые слова

Автоматизация тестирования

Цели обучения для главы 6:

6.1 Инструменты для поддержки тестирования
FL-6.1.1 (K2) Объяснить, как различные типы инструментов поддерживают тестирование

6.2 Преимущества и риски автоматизации тестирования
FL-6.2.1 (K1) Выявить преимущества и риски автоматизации тестирования

6.1 Инструменты поддержки тестирования

Инструменты тестирования поддерживают и облегчают многие действия по тестированию.

Примеры включают, но не ограничиваются приведенным ниже списком:

- Инструменты управления — повышают эффективность процесса тестирования за счет упрощения управления жизненным циклом процесса разработки, требованиями, тестами, дефектами, конфигурацией.
- Инструменты статического тестирования — помогают тестировщику в проведении рецензирования и статического анализа.
- Инструменты проектирования и реализации тестов — упрощают создание тестовых сценариев, тестовых данных и тестовых процедур.
- Инструменты исполнения тестов и измерения покрытия — выполняют автоматизированные тесты и измеряют покрытие.
- Инструменты нефункционального тестирования — позволяют тестировщику выполнять нефункциональное тестирование, которое сложно или невозможно выполнить вручную.
- Инструменты DevOps — поддерживают конвейер доставки DevOps, отслеживание рабочего процесса, автоматизированный процесс (процессы) сборки, CI/CD.
- Инструменты совместной работы – упрощают общение.
- Инструменты, поддерживающие масштабируемость и стандартизацию развертывания (например, виртуальные машины, инструменты контейнеризации).
- Любой другой инструмент, который помогает в тестировании (например, электронная таблица является инструментом тестирования в контексте тестирования).

6.2 Преимущества и риски автоматизации тестирования

Просто приобретение инструмента не гарантирует успеха. Каждый новый инструмент потребует усилий для достижения реальных и долгосрочных результатов (например, на внедрение инструмента, обслуживание и обучение). Существуют также некоторые риски, которые нуждаются в анализе и смягчении.

Потенциальные преимущества от использования автоматизации тестирования включают:

- Экономию времени за счет сокращения повторяющейся ручной работы (например, выполнения регрессионных тестов, повторного ввода одних и тех же тестовых данных, сравнения ожидаемых результатов с фактическими результатами и проверки на соответствие стандартам кодирования).
- Предотвращение простых человеческих ошибок за счет большей согласованности и повторяемости (например, тесты вытекают из требований, тестовые данные создаются систематическим образом, а тесты выполняются инструментом в одинаковом порядке и с одинаковой частотой)
- Более объективную оценку (например, покрытия) и проведение измерений, которые слишком сложны для человека

- Упрощенный доступ к информации о тестировании для поддержки управления тестированием и составления отчетов о тестировании (например, статистике, графикам и агрегированным данным о ходе тестирования, количестве дефектов и продолжительности выполнения теста)
- Сокращение времени выполнения тестов для более раннего обнаружения дефектов, более быстрой обратной связи и более быстрого вывода в промышленную эксплуатацию
- Большее время тестировщиков для разработки новых, более глубоких и эффективных тестов.

Потенциальные риски от использования автоматизации тестирования включают:

- Нереалистичные ожидания относительно преимуществ инструмента (включая функциональность и простоту использования).
- Неточную оценку времени, затрат и усилий, необходимых для внедрения инструмента, поддержания тестовых сценариев и изменения существующего процесса ручного тестирования.
- Использование инструмента тестирования, когда ручное тестирование более целесообразно.
- Чрезмерное доверие инструменту, например, игнорирование необходимости в критическом мышлении.
- Зависимость от поставщика инструмента, который может выйти из бизнеса, вывести инструмент из эксплуатации, продать инструмент другому поставщику или предоставить плохую поддержку (например, ответы на запросы, обновления и исправления дефектов).
- Использование программного обеспечения с открытым исходным кодом, которое могут перестать поддерживать, что означает, что дальнейшие обновления могут быть недоступны, или его внутренние компоненты могут потребовать довольно частых обновлений для дальнейшего развития.
- Несовместимость инструмента автоматизации с платформой разработки.
- Выбор неподходящего инструмента, который не соответствует нормативным требованиям и/или стандартам безопасности.

7. Ссылки

Стандарты

ISO/IEC/IEEE 29119-1 (2022) Software and systems engineering – Software testing – Part 1: General Concepts

ISO/IEC/IEEE 29119-2 (2021) Software and systems engineering – Software testing – Part 2: Test processes

ISO/IEC/IEEE 29119-3 (2021) Software and systems engineering – Software testing – Part 3: Test documentation

ISO/IEC/IEEE 29119-4 (2021) Software and systems engineering – Software testing – Part 4: Test techniques

ISO/IEC 25010, (2011) Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) System and software quality models

ISO/IEC 20246 (2017) Software and systems engineering – Work product reviews

ISO/IEC/IEEE 14764:2022 – Software engineering – Software life cycle processes – Maintenance

ISO 31000 (2018) Risk management – Principles and guidelines

Книги

Adzic, G. (2009) Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing, Neuri Limited

Ammann, P. and Offutt, J. (2016) Introduction to Software Testing (2e), Cambridge University Press

Andrews, M. and Whittaker, J. (2006) How to Break Web Software: Functional and Security Testing of Web Applications and Web Services, Addison-Wesley Professional

Beck, K. (2003) Test Driven Development: By Example, Addison-Wesley

Beizer, B. (1990) Software Testing Techniques (2e), Van Nostrand Reinhold: Boston MA

Boehm, B. (1981) Software Engineering Economics, Prentice Hall, Englewood Cliffs, NJ

Buxton, J.N. and Randell B., eds (1970), Software Engineering Techniques. Report on a conference sponsored by the NATO Science Committee, Rome, Italy, 27–31 October 1969, p. 16

Chelimsky, D. et al. (2010) The Rspec Book: Behaviour Driven Development with Rspec, Cucumber, and Friends, The Pragmatic Bookshelf: Raleigh, NC

Cohn, M. (2009) Succeeding with Agile: Software Development Using Scrum, Addison-Wesley

Copeland, L. (2004) A Practitioner's Guide to Software Test Design, Artech House: Norwood MA

Craig, R. and Jaskiel, S. (2002) Systematic Software Testing, Artech House: Norwood MA

Crispin, L. and Gregory, J. (2008) Agile Testing: A Practical Guide for Testers and Agile Teams, Pearson Education: Boston MA

Forgács, I., and Kovács, A. (2019) Practical Test Design: Selection of traditional and automated test design techniques, BCS, The Chartered Institute for IT

Gawande A. (2009) The Checklist Manifesto: How to Get Things Right, New York, NY: Metropolitan Books

Gärtner, M. (2011), ATDD by Example: A Practical Guide to Acceptance Test-Driven Development, Pearson Education: Boston MA

Gilb, T., Graham, D. (1993) Software Inspection, Addison Wesley

Hendrickson, E. (2013) Explore It!: Reduce Risk and Increase Confidence with Exploratory Testing, The Pragmatic Programmers

Hetzel, B. (1988) The Complete Guide to Software Testing, 2nd ed., John Wiley and Sons

Jeffries, R., Anderson, A., Hendrickson, C. (2000) Extreme Programming Installed, Addison-Wesley Professional

Jorgensen, P. (2014) Software Testing, A Craftsman's Approach (4e), CRC Press: Boca Raton FL

Kan, S. (2003) Metrics and Models in Software Quality Engineering, 2nd ed., Addison-Wesley

Kaner, C., Falk, J., and Nguyen, H.Q. (1999) Testing Computer Software, 2nd ed., Wiley

Kaner, C., Bach, J., and Pettichord, B. (2011) Lessons Learned in Software Testing: A Context-Driven Approach, 1st ed., Wiley

Kim, G., Humble, J., Debois, P. and Willis, J. (2016) The DevOps Handbook, Portland, OR

Koomen, T., van der Aalst, L., Broekman, B. and Vroon, M. (2006) TMap Next for result-driven testing, UTN Publishers, The Netherlands

Myers, G. (2011) The Art of Software Testing, (3e), John Wiley & Sons: New York NY

O'Regan, G. (2019) Concise Guide to Software Testing, Springer Nature Switzerland

Pressman, R.S. (2019) Software Engineering. A Practitioner's Approach, 9th ed., McGraw Hill

Roman, A. (2018) Thinking-Driven Testing. The Most Reasonable Approach to Quality Control, Springer Nature Switzerland

Van Veenendaal, E (ed.) (2012) Practical Risk-Based Testing, The PRISMA Approach, UTN Publishers: The Netherlands

Watson, A.H., Wallace, D.R. and McCabe, T.J. (1996) Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric, U.S. Dept. of Commerce, Technology Administration, NIST

Westfall, L. (2009) The Certified Software Quality Engineer Handbook, ASQ Quality Press

Whittaker, J. (2002) How to Break Software: A Practical Guide to Testing, Pearson

Whittaker, J. (2009) Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design, Addison Wesley

Whittaker, J. and Thompson, H. (2003) How to Break Software Security, Addison Wesley

Wiegers, K. (2001) Peer Reviews in Software: A Practical Guide, Addison-Wesley Professional

Статьи и веб-ресурсы

Brykczynski, B. (1999) "A survey of software inspection checklists," ACM SIGSOFT Software Engineering Notes, 24(1), pp. 82-89

Enders, A. (1975) “An Analysis of Errors and Their Causes in System Programs,” IEEE Transactions on Software Engineering 1(2), pp. 140-149

Manna, Z., Waldinger, R. (1978) “The logic of computer programming,” IEEE Transactions on Software Engineering 4(3), pp. 199-229

Marick, B. (2003) Exploration through Example, http://www.exampler.com/old-blog/2003/08/21.1.html#agile-testing-project-1

Nielsen, J. (1994) “Enhancing the explanatory power of usability heuristics,” Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Celebrating Interdependence, ACM Press, pp. 152–158

Salman. I. (2016) “Cognitive biases in software quality and testing,” Proceedings of the 38th International Conference on Software Engineering Companion (ICSE '16), ACM, pp. 823-826.

Wake, B. (2003) “INVEST in Good Stories, and SMART Tasks,” https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/

8. Приложение A – Цели обучения / Уровень знаний

В текущей программе обучения определены следующие цели обучения. Каждая тема в программе будет проверяться в соответствии с целями обучения, определенными для нее. Формулировки целей обучения начинаются с глагола, соответствующего уровню знаний, как перечислено ниже.

Уровень 1: Запомнить (K1) - кандидат запоминает, различает и использует термин или понятие.

Глаголы: определить, использовать, запомнить, узнать.

Примеры:

- “Определить типичные цели тестирования”
- “Запомнить концепцию пирамиды тестирования”
- “Узнать, как тестировщик повышает ценность итерации и планирования”

Уровень 2: Понять (K2) - кандидат указывает причины или поясняет понятия, относящиеся к теме, а также резюмирует, сравнивает, классифицирует, разделяет по категориям и приводит примеры.

Глаголы: классифицировать, сравнить, сопоставлять, различать, выделить, показывать, объяснять, приводить примеры, интерпретировать, обобщать.

Примеры:

- “Классифицировать различные способы по созданию критериев приемки”
- “Сравнить различные роли в тестировании” (рассмотреть сходства и/или различия)
- “Выделить риски проекта и риски продукта” (позволяет различать понятия)
- “Привести примеры цели и содержания плана тестирования”
- “Объяснить влияние контекста на процесс тестирования”
- “Обобщить мероприятия процесса рецензирования”

Уровень 3: Применить (K3) - кандидат может выполнить процедуру, встретившись со знакомой задачей, или выбрать правильную процедуру и использовать ее в заданном контексте.

Глаголы действия: применить, реализовать, подготовить, использовать.

Примеры:

- “Применить определение приоритетов к тестовым сценариям” (должно иметь отношение к процедуре, методике, процессу, алгоритму и т.д.)
- “Подготовить отчет о дефекте”
- “Использовать анализ граничных значения для получения тестовых сценариев”

Ссылки на литературу, описывающую цели обучения и их уровни**:**

Anderson, L. W. and Krathwohl, D. R. (eds) (2001) A Taxonomy for Learning, Teaching

Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Allyn & Bacon

9. Приложение B – Матрица, показывающая связь между бизнес-результатами и целями обучения

В этом приложении для программы обучения Базового уровня перечислены цели обучения, связанные с бизнес-результатами, а также представлена матрица, показывающая связь между бизнес-результатами и целями обучения.

Бизнес-результаты: Программа обучения Базового уровня		FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
BO1	Понимать, что такое тестирование и почему оно полезно	6													
BO2	Понимать фундаментальные концепции тестирования ПО		22												
BO3	Определять подход к тестированию и действия, которые необходимо реализовать в зависимости от контекста тестирования			6											
BO4	Оценивать и улучшать качество документации				9										
BO5	Повышать эффективность и результативность тестирования					20									
BO6	Согласовывать процесс тестирования с жизненным циклом разработки ПО						6								
BO7	Понимать принципы управления тестированием							6							
BO8	Формировать и передавать четкие и понятные отчеты о дефектах								1						
BO9	Понимать факторы, влияющие на приоритеты и усилия, связанные с тестированием									7					
BO10	Работать как часть кросс-функциональной команды										8				
BO11	Знать риски и преимущества, связанные с автоматизацией тестирования											1			
BO12	Определять основные навыки, необходимые для тестирования												5		
BO13	Понимать влияние риска на тестирование													4	
BO14	Эффективно сообщать о ходе и качестве тестирования														4

Глава/ раздел/ подраздел	Цель обучения	Уровень знаний	Бизнес-результат													
			FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
Глава 1	**Основы тестирования**															
1.1	**Что такое тестирование?**															
1.1.1	Определить типичные цели тестирования	K1	X													
1.1.2	Различить тестирование и отладку	K2		X												
1.2	**Почему тестирование необходимо?**															
1.2.1	Объяснить на примерах, почему тестирование необходимо	K2	X													
1.2.2	Запомнить связь между тестированием и обеспечением качества	K1		X												
1.2.3	Выделить различия между первопричиной, ошибкой, дефектом и отказом	K2		X												
1.3	**Принципы тестирования**															
1.3.1	Объяснить семь принципов тестирования	K2		X												
1.4	**Активности тестирования, тестовое обеспечение и роли в тестировании**															
1.4.1	Обобщить различные активности и задачи тестирования	K2			X											
1.4.2	Объяснить влияние контекста на процесс тестирования	K2			X			X								
1.4.3	Различить тестовое обеспечение, поддерживающее активности тестирования	K2			X											
1.4.4	Объяснить ценность обеспечения трассируемости	K2				X	X									
1.4.5	Сравнить различные роли в тестировании	K2										X				
1.5	**Основные навыки и передовой опыт в тестировании**															
1.5.1	Привести примеры основных навыков, необходимых для тестирования	K2												X		
1.5.2	Запомнить преимущества командного подхода	K1										X				
1.5.3	Выделить преимущества и недостатки независимости тестирования	K2			X											
Глава 2	**Тестирование в жизненном цикле разработки программного обеспечения**															

Глава/ раздел/ подраздел	Цель обучения	Уровень знаний	Бизнес-результат													
			FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
2.1	**Тестирование в контексте жизненного цикла разработки программного обеспечения**															
2.1.1	Объяснить влияние выбранного жизненного цикла разработки программного обеспечения на тестирование	K2						X								
2.1.2	Запомнить эффективные практики тестирования, применяющиеся во всех жизненных циклах разработки программного обеспечения	K1						X								
2.1.3	Запомнить примеры подходов опережающего проектирования тестов в разработке программного обеспечения	K1					X									
2.1.4	Обобщить как DevOps может влиять на тестирование	K2					X	X			X	X				
2.1.5	Объяснить подход сдвига влево	K2					X	X								
2.1.6	Объяснить, как ретроспективы могут быть использованы в качестве механизма для улучшения процесса	K2					X					X				
2.2	**Уровни тестирования и типы тестирования**															
2.2.1	Выделить различные уровни тестирования	K2		X	X											
2.2.2	Выделить различные типы тестирования	K2		X												
2.2.3	Различать подтверждающее тестирование и регрессионное тестирование	K2		X												
2.3	**Тестирование в период сопровождения**															
2.3.1	Обобщить тестирование в период сопровождения и его предпосылки	K2		X					X							
Глава 3	**Статическое тестирование**															
3.1	**Основы статического тестирования**															
3.1.1	Узнавать типы продуктов, которые можно проанализировать с помощью разных методов статического тестирования	K1				X	X									
3.1.2	Объяснить ценность статического тестирования	K2	X			X	X									
3.1.3	Сравнить и противопоставить статическое и динамическое тестирование	K2				X	X									

Глава/ раздел/ подраздел	Цель обучения	Уровень знаний	Бизнес-результат													
			FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
3.2	**Работа с обратной связью и процесс рецензирования**															
3.2.1	Выявить преимущества ранней и частой обратной связи от заинтересованных сторон	K1	X			X						X				
3.2.2	Обобщить мероприятия процесса рецензирования	K2			X	X										
3.2.3	Запомнить обязанности основных участников процесса рецензирования	K1				X								X		
3.2.4	Сравнить и противопоставить разные типы рецензирования	K2		X												
3.2.5	Запомнить факторы, способствующие успешному рецензированию	K1					X							X		
Глава 4	**Анализ и проектирование тестов**															
4.1	**Методы тестирования. Общие сведения**															
4.1.1	Выделить различие между тестированием методом черного ящика, тестированием методом белого ящика и тестированием на основе опыта	K2		X												
4.2	**Тестирование методом черного ящика**															
4.2.1	Использовать метод эквивалентного разбиения для создания тестовых сценариев	K3					X									
4.2.2	Использовать метод анализа граничных значений для создания тестовых сценариев	K3					X									
4.2.3	Использовать метод таблицы решений для создания тестовых сценариев	K3					X									
4.2.4	Использовать метод таблицы переходов для создания тестовых сценариев	K3					X									
4.3	**Тестирование методом белого ящика**															
4.3.1	Объяснить, в чем заключается метод тестирования операторов	K2		X												
4.3.2	Объяснить, в чем заключается метод тестирования ветвей	K2		X												
4.3.3	Объяснить важность тестирования методом белого ящика	K2	X	X												
4.4	**Тестирование на основе опыта**															

Глава/ раздел/ подраздел	Цель обучения	Уровень знаний	Бизнес-результат													
			FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
4.4.1	Объяснить, в чем заключается метод предположения об ошибках	K2		X												
4.4.2	Объяснить, в чем заключается метод исследовательского тестирования	K2		X												
4.4.3	Объяснить, в чем заключается метод тестирования на основе чек-листов	K2		X												
4.5	**Подходы к тестированию, основанные на совместной работе**															
4.5.1	Объяснить, как создавать пользовательские истории через совместную работу с разработчиками и представителями заказчика	K2				X						X				
4.5.2	Классифицировать различные способы по созданию критериев приемки	K2										X				
4.5.3	Использовать разработку через приемочное тестирование для создания тестовых сценариев	K3					X									
Глава 5	**Управление тестированием**															
5.1	**Планирование тестирования**															
5.1.1	Привести примеры цели и содержания плана тестирования	K2		X					X							
5.1.2	Узнать, как тестировщик повышает ценность итерации и планирования выпуска	K1	X									X		X		
5.1.3	Сравнить и сопоставить критерии входа и выхода	K2				X		X								X
5.1.4	Использовать методы оценки для определения необходимых трудозатрат для тестирования	K3							X		X					
5.1.5	Применять определение приоритетов тестовых сценариев	K3							X		X					
5.1.6	Запомнить концепцию пирамиды тестирования	K1		X												
5.1.7	Обобщить квадранты тестирования и их взаимосвязь с уровнями и типами тестирования	K2		X							X					
5.2	**Управление рисками**															
5.2.1	Определить уровень рисков, используя вероятность и влияние рисков	K1							X						X	
5.2.2	Выделить риски проекта и риски продукта	K2		X											X	

Глава/ раздел/ подраздел	Цель обучения	Уровень знаний	Бизнес-результат													
			FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
5.2.3	Объяснить как анализ рисков может повлиять на тщательность и объем тестирования	K2					X				X				X	
5.2.4	Объяснить, какие меры могут быть приняты по результатам анализа рисков продукта	K2		X			X								X	
5.3	**Мониторинг, контроль и завершение тестирования**															
5.3.1	Запомнить метрики, используемые для тестирования	K1									X					X
5.3.2	Обобщить цель, содержание и целевые аудитории отчетов о тестировании	K2					X				X					X
5.3.3	Привести примеры как сообщать о статусе тестирования	K2												X		X
5.4	**Управление конфигурацией**															
5.4.1	Обобщить, как управление конфигурацией поддерживает тестирование	K2					X		X							
5.5	**Управление дефектами**															
5.5.1	Подготовить отчет о дефекте	K3		X						X						
Глава 6	**Инструменты тестирования**															
6.1	**Инструменты для поддержки тестирования**															
6.1.1	Объяснить, как различные типы инструментов поддерживают тестирование	K2					X									
6.2	**Преимущества и риски автоматизации тестирования**															
6.2.1	Выявить преимущества и риски автоматизации тестирования	K1					X						X			

10. Приложение C – Описание изменений

Программа обучения ISTQB базового уровня версии 4.0 — это крупное обновление программы базового уровня версии 3.1.1 и тестировщика в сфере гибких методологий версии 2014. Именно по этой причине здесь нет подробных замечаний по каждой главе и разделу. Ниже приводится краткое содержание основных изменений. Кроме того, в отдельном документе с приложениями к выпуску ISTQB предоставляет прослеживаемость между целями обучения (LO) программы обучения базового уровня версии 3.1.1, программы обучения тестировщика в сфере гибких методологий версии 2014 и целями обучения новой программы базового уровня версии 4.0, показывая, какие цели обучения (LO) были добавлены, обновлены, или удалены.

На момент написания этой программы обучения (2022-2023 годы) более миллиона человек в более чем 100 странах сдали экзамен базового уровня, и более 800 000 являются сертифицированными тестировщиками по всему миру. Учитывая, что все они ознакомились с программой базового уровня, чтобы сдать экзамен, программа базового уровня является, вероятно, самым читаемым документом по тестированию программного обеспечения за всю историю! Это крупное обновление сделано с уважением к этому наследию и для того, чтобы улучшить мнение сотен тысяч людей об уровне качества, который ISTQB предоставляет мировому сообществу тестировщиков.

В этой версии все цели обучения (LO) были отредактированы, чтобы сделать их атомарными и создать взаимно-однозначную прослеживаемость между целями обучения (LO) и разделами программы обучения, чтобы не допустить появления контента без наличия цели обучения (LO). Цель состоит в том, чтобы сделать эту версию более легкой для чтения, понимания, изучения и перевода, уделяя особое внимание повышению практической полезности и балансу между знаниями и навыками.

В этом крупном выпуске были реализованы следующие изменения:

- Сокращение объема общей программы обучения. Программа обучения — это не учебник, а документ, который служит для описания основных элементов вводного курса по тестированию программного обеспечения, включая то, какие темы должны быть охвачены и на каком уровне. Поэтому, в частности:
  - В большинстве случаев исключены примеры из текста. Предоставление примеров, а также упражнений во время тренинга является задачей организаторов обучения.
  - Был применен “Контрольный список написания программ обучения”, который предполагает максимальный размер текста для целей обучения (LO) на каждом K-уровне (K1 = максимум 10 строк, K2 = максимум 15 строк, K3 = максимум 25 строк)
- Уменьшение количества целей обучения (LO) по сравнению с программами обучения базового уровня версии 3.1.1 и тестировщика в сфере гибких технологий версии 2014
  - 14 целей обучения (LO) уровня K1 по сравнению с 21 в программах базового уровня версии 3.1.1 (15) и тестировщика в сфере гибких технологий 2014 (6)
  - 42 цели обучения (LO) уровня K2 по сравнению с 53 в программах базового уровня версии 3.1.1 (40) и тестировщика в сфере гибких технологий 2014 (13)
  - 8 целей обучения (LO) уровня K3 по сравнению с 15 в программах базового уровня версии 3.1.1 (7) и тестировщика в сфере гибких технологий 2014 (8)

- Приведены более подробные ссылки на классические и/или хорошо зарекомендовавшие себя книги и статьи по тестированию программного обеспечения и смежным темам
- Основные изменения в главе 1 (Основы тестирования)
  - Расширен и улучшен раздел о навыках тестирования
  - Добавлен раздел, посвященный командному подходу (K1)
  - Раздел о независимости тестирования перенесен в главу 1 из главы 5
- Основные изменения в главе 2 (Тестирование в жизненном цикле разработки программного обеспечения)
  - Переписаны и улучшены разделы 2.1.1 и 2.1.2, соответствующие цели обучения (LO) изменены
  - Больше внимания уделяется таким практикам, как: подход опережающего проектирования тестов (K1), сдвиг влево (K2), ретроспективы (K2)
  - Добавлен новый раздел, посвященный тестированию в контексте DevOps (K2)
  - Уровень интеграционного тестирования разделен на два отдельных уровня тестирования: тестирование интеграции компонентов и системное интеграционное тестирование
- Основные изменения в главе 3 (Статическое тестирование)
  - Удален раздел о методах рецензирования вместе с целью обучения (LO) уровня K3 (применение методов рецензирования)
- Основные изменения в главе 4 (Анализ и проектирование тестов)
  - Удалено тестирование по сценариям использования (но все еще присутствует в программе продвинутого уровня тест-аналитик)
  - Больше внимания уделяется подходу к тестированию, основанному на совместной работе: новая цель обучения (LO) уровня K3 об использовании разработки через приемочное тестирование для создания тестовых примеров и две новые цели обучения (LO) уровня K2, посвященных пользовательским историям и критериям приемки
  - Тестирование и покрытие решений заменены на тестирование и покрытие ветвей (во-первых, на практике чаще используется покрытие ветвей; во-вторых, разные стандарты определяют решение по-разному, в отличие от “ветвей”; в-третьих, это устраняет тонкий, но серьезный недостаток старой программы базового уровня 2018, в которой утверждается, что стопроцентное покрытие решений подразумевает стопроцентное покрытие операторов - это предложение неверно в случае программ без ветвлений)
  - Улучшен раздел о ценности тестирования методом белого ящика
- Основные изменения в главе 5 (Управление тестированием)
  - Удален раздел о стратегиях/подходах тестирования
  - Новая цель обучения (LO) уровня K3 по методам оценки для оценки трудозатрат на тестирование
  - Больше внимания уделяется хорошо известным концепциям и инструментам управления тестированием, связанным с гибкими методологиями: планированию

итераций и выпусков (K1), пирамиде тестирования (K1) и квадрантам тестирования (K2).

- Раздел, посвященный управлению рисками, лучше структурирован путем описания четырех основных видов деятельности: выявление рисков, оценка рисков, снижение рисков и мониторинг рисков

- Основные изменения в главе 6 (Инструменты тестирования)
  - Содержимое по некоторым вопросам автоматизации тестирования сокращено как слишком сложное для базового уровня – удален раздел о выборе инструментов, выполнении пилотных проектов и внедрении инструментов в организацию

11. Предметный указатель