3. Методы, метдики и инструменты тестирования в проектах под управлением гибких методологий – 480 минут

Ключевые слова
Критерии приемки, исследовательское тестирование, нагрузочное тестирование, риск продукта, риск качества, регрессионное тестирование, подход к тестированию, концепция тестирования, оценка затрат на тестирование, автоматизация выполнения тестов, стратегия тестирования, разработка на основе тестов, интегрированная среда модульного тестирования.

Цели обучения методам, техникам и инструментам гибкой методологии тестирования

3.1 Методы тестирования в проектах под управлением гибких методологий

FA-3.1.1 (K1) Вспоминаем понятия и приемы разработки, основанной на тестировании, и разработки, основанной на поведении
FA-3.1.2 (K1) Вспоминаем понятия пирамиды тестов
FA-3.1.3 (K2) Рассматриваем квадранты тестирования через призму их отношения к уровням и типам тестирования
FA-3.1.4 (K3) Определяем роль тестировщика в команде проекта под управлением гибких методологий

3.2 Оценка рисков качества и объема работ по тестированию

FA-3.2.1 (K3) Оцениваем риски качества в проектах с гибкой методологией
FA-3.2.2 (K3) Оцениваем объем работ по тестированию на основе содержания итерации и рисков качества

3.3 Техники тестирования в проектах под управлением гибких методологий

FA-3.3.1 (K3) Используем доступную информацию для поддержки тестирования
FA-3.3.2 (K2) Объясняем заказчику, как установить проверяемые критерии приемки продукта
FA-3.3.3 (K3) Пишем тест-кейсы для разработки через приемочное тестирование, используя пользовательские истории
FA-3.3.4 (K3) Пишем тест-кейсы для функционального и нефункционального тестирования с применением разработки тестов методом черного ящика, используя пользовательские истории
FA-3.3.5 (K3) Проводим исследовательское тестирование для поддержания тестирования в проекте с гибкой методологией

3.4 Инструменты тестирования в проектах под управлением гибких методологий

FA-3.4.1 (K1) Вспоминаем различные инструменты, доступные для тестировщиков, в соответствии с их назначением и использованием в проектах с гибкой методологией

3.1. Методы тестирования в проектах под управлением гибких методологий

Существуют конкретные методы тестирования, которым следуют в каждом разрабатываемом проекте (под управлением как гибких, так и традиционных методологий) для получения качественной продукции. К ним относятся заранее созданные тесты, описывающие правильное поведение, причем особое внимание уделяется раннему предотвращению появления дефектов, их выявлению и удалению. Также гарантируется, что правильные типы тестов будут выполняться в заданное время и будут частью правильного уровня тестирования. В проектах с гибкой методологией стараются внедрить эту практику на ранней стадии. Тестировщики в гибких проектах играют ключевую роль во внедрении этих методов тестирования на протяжении всего жизненного цикла проекта.

3.1.1. Разработка на основе тестов, разработка через приемочное тестирование и разработка на основе поведения

Разработка, основанная на тестах, разработка через приемочное тестирование и разработка, основанная на поведении – это три взаимодополняющих метода, которые используются в группах, работающих под управлением гибких методологий, для проведения тестирования на разных уровнях. Каждый метод является примером фундаментального принципа тестирования, преимущества раннего тестирования и включения в работу команды обеспечения качества, поскольку тесты разрабатываются еще до написания кода.

Разработка, основанная на тестах

Разработка, основанная на тестах, используется для написания кода, управляемого автоматизированными тест-кейсами. Процесс разработки, основанной на тестах выстроен следующим образом:

- Добавляем тест, который фиксирует концепцию разработчика о требуемом функционировании небольшого фрагмента кода
- Запускаем тест, который должен завершится неудачно, поскольку код не существует
- Пишем код и запускаем тест по короткому циклу до тех пор, пока тест не будет пройден

- Проводим рефакторинг кода после прохождения теста, перезапускаем тест, чтобы убедиться, что тест проходит успешно после рефакторинга
- Повторяем процесс для следующего небольшого фрагмента кода, выполняя предыдущие тесты совместно с добавленными

Написанные тесты в основном относятся к конкретному функциональному блоку и ориентированы на код, хотя тесты также могут быть написаны на уровнях интеграции или всей системы. Разработка, основанная на тестах, завоевала популярность благодаря Экстремальному Программированию [Beck02], но также она используется и в других гибких методологиях, а иногда и в последовательных жизненных циклах. Это помогает разработчикам сосредоточиться на четко ожидаемых результатах. Тестирование, при этом, автоматизировано и используется в непрерывной интеграции.

Разработка через приемочное тестирование

Разработка через приемочное тестирование [Adzic09] определяет критерии приемки и тесты при создании пользовательских историй (см. Раздел 1.2.2). Разработка через приемочное тестирование – это совместный подход, позволяющий любому заинтересованному лицу понять, как должен себя вести программный компонент, и как разработчики, тестировщики и представители заказчика должны обеспечить данное поведение. Процесс разработки через приемочное тестирование объясняется в Разделе 3.3.2.

Разработка через приемочное тестирование создает многоразовые тесты для регрессионного тестирования. Специальные инструменты поддерживают создание и выполнение таких тестов, часто в процессе непрерывной интеграции. Эти инструменты могут подключаться к уровням данных и сервисов приложения, что позволяет выполнять тесты на уровне системы или приемки. Разработка через приемочное тестирование позволяет быстро устранять дефекты и проверять поведение функциональности. Это помогает определить удовлетворяются ли критерии приемки для данной функциональности.

Разработка, основанная на поведении

Разработка, основанная на поведении, [Chelimsky10] позволяет разработчику сосредоточиться на тестировании кода, написанного, исходя из ожидаемого поведения программного обеспечения. Поскольку тесты так же базируются на ожидаемом поведении программного обеспечения, их легче понять другим членам команды и заинтересованным сторонам.

Конкретные механизмы разработки, основанной на поведении, могут использоваться для определения критериев приемки на основе формата Дано / Когда / То:

Дан некоторый исходный контекст,

Когда происходит событие,

То удостоверяемся в полученных результатах.

Исходя из этих требований, среда разработки, основанной на функционировании, ге-

нерирует код, который может быть использован разработчиками для создания тест-кейсов. Такой подход помогает разработчику взаимодействовать с другими заинтересованными сторонами, в том числе с тестировщиками, для определения модульных тестов, ориентированных на потребности заказчика.

3.1.2. Пирамида тестов

Система программного обеспечения может быть протестирована на разных уровнях. Типичные уровни тестирования начинаются с основания пирамиды до вершины: модульное, интеграционное, системное, приемочное (см. [ISTQB_FL_SYL], Раздел 2.2). Пирамида тестов подразумевает наличие большого количества тестов на нижних уровнях (внизу пирамиды), и, по мере передвижения к верхним уровням, количество тестов уменьшается (верхняя часть пирамиды). Обычно модульные и интеграционные тесты автоматизированы и создаются с использованием инструментов программного интерфейса приложения. На уровне системного тестирования и приемки автотесты создаются с использованием инструментов на основе графического интерфейса пользователя. Концепция пирамиды тестов основана на принципе раннего включения в работу команды обеспечения качества и тестирования (то есть на устранении дефектов как можно раньше в жизненном цикле).

3.1.3. Квадранты тестирования, уровни тестирования, типы тестирования

Квадранты тестирования, определенные Брайаном Мэриком [Crispin08], сопоставляют уровни тестирования с соответствующими типами тестов в гибкой методологии. Модель квадрантов тестирования и ее варианты помогают гарантировать, что все важные типы тестов и уровни тестирования включены в жизненный цикл разработки. Эта модель также позволяет различать и описывать типы тестов для всех заинтересованных сторон, включая разработчиков, тестировщиков и представителей заказчика.

В квадрантах тестирования могут быть бизнес-тесты (пользовательские) или технологические тесты (для разработчика). Некоторые тесты поддерживают работу, сделанную командой, и подтверждают поведение программного обеспечения. Другие тесты могут проверить продукт. Тесты могут быть полностью мануальными, полностью автоматизированными или комбинированными (мануальными и автоматизированными), или же мануальными при поддержке инструментов. Квадранты тестирования представляют собой следующее:

- Квадрант Q1 – это модульный уровень, представленный технологией и поддерживающий разработчиков. Этот квадрант содержит модульные тесты, которые должны быть автоматизированы и включены в процесс непрерывной интеграции
- Квадрант Q2 – это системный, бизнес-ориентированный уровень, подтверждающий поведение продукта. Этот квадрант содержит функциональные тесты, примеры, тестовые истории, прототипы пользовательского опыта и моделирование. Эти тесты проверяют критерии приемки и могут быть мануальными или автоматизированными.

Они часто создаются во время разработки пользовательских историй и, таким образом, улучшают качество самих историй. Такие тесты полезны при создании автоматизированных регрессионных тестов

- Квадрант Q3 – это бизнес-ориентированный уровень приемки системы, содержащий тесты, которые проверяют продукт, используя реальные сценарии и данные. Этот квадрант включает в себя исследовательское тестирование, сценарии, потоки процессов, тестирование удобства использования, пользовательское приемочное тестирование, альфа-тестирование и бета-тестирование. Эти тесты часто являются мануальными и ориентированы на пользователей
- Квадрант Q4 – системный или операционный уровень приемки, содержащий тесты, испытывающие продукт. Этот квадрант содержит тесты производительности, тесты для нагрузочного и стресс-тестирования, тесты масштабируемости, тесты безопасности, сопровождаемости, управления памятью, совместимости и взаимодействия, миграции данных, инфраструктуры и восстановления. Эти тесты, зачастую, автоматизированы.

Во время любой итерации могут потребоваться тесты из любого или всех квадрантов. Квадранты тестирования применимы к динамическому, а не к статическому тестированию.

3.1.4. Роль тестировщика

В рамках этой программы главный упор был сделан на гибкие методы и методики тестирования, а также на роль тестировщика в различных жизненных циклах проектов под управлением гибких методологий. В этом подразделе конкретно рассматривается роль тестировщика в проекте в соответствии с жизненным циклом Скрам [Aalast13].

Командная работа

Командная работа является основополагающим принципом развития любого проекта. Краеугольным камнем гибких методологий применительно к работе проектных команд, состоящих из разработчиков, тестировщиков, представителей бизнеса и других заинтересованных сторон, является совместная работа всех их членов. Ниже приведены организационные и поведенческие рекомендации в командах, работающих по методологии Скрам:

- Кросс-функциональность: каждый член команды привносит в команду различный набор навыков. Команда работает вместе над стратегией, планированием тестирования, тестовой спецификацией, выполнением тестов, оценкой тестирования и отчетами о его результатах
- Самоорганизация: команда может состоять только из разработчиков, но, как отмечено в разделе 2.1.5, в идеале должен быть один или несколько тестировщиков
- Совместное размещение: тестировщики сидят вместе с разработчиками и владельцем продукта

- Взаимодействие: тестировщики осуществляют взаимодействие со всеми членами команды, другими командами, заинтересованными сторонами, владельцем продукта и руководителем команды
- Право на принятие решений: технические решения, касающиеся проектирования и тестирования, выполняются всей командой (разработчики, тестировщики, Скрам-мастер) в сотрудничестве с владельцем продукта и другими командами, если это необходимо
- Ориентированность на результат: тестировщик старается задавать вопросы и оценивать поведение и характеристики продукта в отношении ожиданий и потребностей клиентов и пользователей
- Прозрачность: прогресс разработки и тестирования отображается на панели задач проекта (см. Раздел 2.2.1)
- Достоверность: тестировщик должен обеспечить доверие к стратегии тестирования, ее реализации и выполнению, иначе заинтересованные стороны не будут доверять результатам тестирования. Часто это решается предоставлением информации о процессе тестирования заинтересованным сторонам
- Открытость для обратной связи: обратная связь – важный аспект успеха в любом проекте, особенно в проекте под управлением гибких методологий. Ретроспективы позволяют командам учиться на успехах и неудачах
- Гибкость: тестирование, как и любые другие активности в проекте, должно иметь возможность реагировать на изменения

Применение этих практик максимизирует вероятность успешного тестирования в проектах под управлением Скрам.

Нулевой спринт

Нулевой спринт – это первая итерация проекта, в которой происходит множество мероприятий по подготовке (см. Раздел 1.2.5). Тестировщик в течение этой итерации взаимодействует с командой посредством следующих активностей:

- Определяет сферу действий по проекту (то есть формирует список задач по продукту)
- Создает исходную системную архитектуру и прототипы высокоуровневых тестов
- Планирует, приобретает и устанавливает необходимые инструменты (например, для управления тестированием, управления дефектами, автоматизации тестирования и непрерывной интеграции)
- Разрабатывает начальную стратегию тестирования для всех уровней тестирования, исходя из (среди всего прочего) областей тестирования, технических рисков, типов тестов (см. Раздел 3.1.3) и требований, предъявляемых к покрытию функционала тестами
- Выполняет первоначальный анализ риска качества (см. Раздел 3.2.1)
- Определяет метрики тестирования для измерения процесса тестирования, прохождения тестирования в проекте и качества продукта
- Дает определение понятию «Сделано» в контексте данного проекта

- Создает панель задач (см. Раздел 2.2.1)
- Определяет, когда можно продолжить или следует остановить тестирование перед поставкой заказчику

Нулевой спринт задает направление – как достичь необходимого результата тестирования во время спринтов.

Интеграция

В проектах под управлением гибких методологий, целью является постоянное поддержание потребительской ценности продукта (желательно в каждом спринте). Для достижения этой цели, стратегия интеграции должна учитывать как проектирование, так и тестирование. Чтобы обеспечить непрерывное тестирование для поставляемых функций и характеристик, важно определить все зависимости между базовыми функциями и ключевыми функциональностями.

Планирование тестирования

Поскольку тестирование полностью интегрировано в команду, использующую гибкую методологию, планирование тестирования должно начинаться во время планирования релиза и повторяться с каждым следующим спринтом. Планирование тестирования для релиза и каждого спринта должно касаться вопросов, обсуждаемых в разделе 1.2.5.

Планирование спринта приводит к получению набора задач, размещаемого на панели задач, где каждая задача должна иметь продолжительность от одного до двух рабочих дней. Кроме того, любые проблемы с тестированием следует отслеживать, чтобы поддерживать постоянный поток задач тестирования.

Практика гибкого тестирования

Многие практики могут быть полезны для тестировщиков в команде, работающей по Скрам. Некоторые могут включать в себя следующее:

- Сопряжение: два члена команды (например, тестировщик и разработчик, два тестировщика или тестировщик и владелец продукта) сидят вместе за одной рабочей станцией для выполнения тестирования или другой задачи спринта
- Инкрементальный тест-дизайн: тест-кейсы и тестовые концепции постепенно строятся из пользовательских историй и других основ для написания тестов, начиная с простых тестов и переходя к более сложным
- Диаграммы связей: диаграммы связей – очень полезный инструмент при тестировании [Crispin08]. Например, тестировщики могут использовать их, чтобы определить, какие сессии тестирования выполнять, какие стратегии тестирования показывать и как описывать тестовые данные.

Эти практики приводятся в дополнении к другим, обсуждаемым в этой программе и в Главе 4 программы базового уровня [ISTQB_FL_SYL].

3.2. Оценка рисков и расчет трудозатрат

Типичной задачей тестирования в любых проектах, под управлением гибких или традиционных методологий, является сокращение проблем, связанных с качеством продукта, до уровня, приемлемого для выхода в релиз. Для выявления рисков качества (или рисков продукта), оценки соответствующего уровня риска, подсчета трудозатрат, необходимых для достаточного уменьшения рисков, и после, для их смягчения посредством тест-дизайна, реализации и выполнения, тестировщики в проектах под управлением гибких методологий могут использовать такие же техники, как и тестировщики традиционных проектов. Однако, учитывая короткие итерации и скорость изменений в проектах с гибкими методологиями, требуется некоторая адаптация этих методов.

3.2.1. Оценка рисков в проектах под управлением гибких методологий

Одной из сложных задач в тестировании является подходящий выбор, распределение и приоритизация тестовых условий. Это включает в себя определение соответствующей величины трудозатрат, чтобы покрыть каждое состояние тестами и упорядочить полученные тесты и предстоящие работы по тестированию максимально эффективно. Идентификация рисков, анализ и стратегия по снижению рисков могут использоваться гибкими командами тестировщиков для определения необходимого количества выполняемых тестовых сценариев, однако, следует добавить некоторые исключения в силу ограничений взаимодействия.
Риск - это событие, появляющееся с ненулевой вероятностью и оказывающее негативное влияние на проект. Уровень риска определяется путем оценки вероятности возникновения риска и степени его влияния. Когда основное влияние потенциальной проблемы связано с качеством продукта, такую проблему называют риском качества или риском продукта. Когда основное влияние потенциальной проблемы связано с успехом проекта, такую проблему называют проектным риском или риском планирования [Black07] [vanVeenendaal12].

В проектах под управлением гибких методологий анализ рисков осуществляется дважды:

- При планировании релиза: представитель бизнеса, который в курсе особенностей релиза, предоставляет высокоуровневый перечень рисков, и вся команда, включая тестировщиков, может участвовать в выявлении и оценке рисков;
- При итерационном планировании: вся команда выявляет и оценивает риски.

Примерами рисков качества являются:

- Неправильные вычисления в отчетах (функциональный риск, связанный с точностью)

- Медленное реагирование на действия пользователей (нефункциональный риск, связанный с эффективностью и временем реакции)
- Сложности, связанные с пониманием экранов и экранных объектов (нефункциональный риск, связанный с удобством использования и понятностью)

Как упоминалось ранее, итерация начинается с итерационного планирования, в результате которого мы получаем оцененные задачи на панели задач. Эти задачи могут быть ранжированы по приоритету в соответствии с их уровнем риска. Задачи с высоким приоритетом необходимо начинать раньше и задействовать больше трудозатрат на их тестирование. Задачи с низким приоритетом можно начинать позже и задействовать меньше трудозатрат.

Ниже пример того, как может быть построен процесс итерационного планирования для анализа рисков в проектах под управлением гибких методологий:

1. Соберите всех членов команды проекта вместе, включая тестировщиков
2. Составьте список всех задач для текущей итерации (например, на панели задач)
3. Определите риски для каждой задачи, учитывая все факторы рисков
4. Оцените каждый риск согласно категории, уровня его воздействия и вероятности
5. Определите масштабы тестирования согласно уровню риска
6. Выберите соответствующие методики для уменьшения каждого конкретного риска, основываясь на степени риска и соответствующих характеристиках качества.

Далее, тестировщик разрабатывает, реализует и выполняет тесты для уменьшения рисков. Этот процесс содержит в себе совокупность функций, моделей поведения, качественных характеристик и атрибутов, от которых зависит, будет ли продукт соответствовать ожиданиям клиентов, пользователей и заинтересованной стороны.

В ходе проекта команда должна быть в курсе новой дополнительной информации, которая может повлечь изменения в наборе рисков и/или уровне риска для уже известных рисков качества. Анализ рисков должен периодически корректироваться и, как следствие, корректируются и сами тесты. Корректировки включают в себя выявление новых рисков, переоценку уровня уже существующих и оценку эффективности мероприятий по снижению рисков.

Риски качества могут быть снижены и до начала выполнения тестов. Например, если при выявлении рисков обнаруживаются проблемы с пользовательскими историями, то команда проекта может, в качестве стратегии по снижению рисков, более подробно проанализировать пользовательские истории.

3.2.2. Оценка трудозатрат на тестирование, исходя из содержания и рисков

В ходе планирования релиза команда проекта под управлением гибких методологий оценивает трудозатраты, необходимые для завершения релиза. Оценка также касается и трудозатрат, необходимых для тестирования. Основной метод оценки, используемый в таких проектах, это покер-планирования – метод, основанный на поиске консенсуса. Владелец продукта или клиент читает пользовательские истории

(список требований к разрабатываемой системе) для участников обсуждения. Каждый участник имеет колоду карт, пронумерованных последовательностью чисел Фибоначчи (0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...) или любой другой последовательностью на выбор (например, значения в которых варьируются от очень маленьких до очень-очень больших). Значение на карте может быть количеством требований, дней или другой величиной, которую необходимо оценить. Последовательность чисел Фибоначчи рекомендуется использовать потому, что отражаемая в ней неопределенность растет пропорционально росту оцениваемых требований. Высокая оценка обычно означает, что требование трудно оценить или оно должно быть разбито на более мелкие требования.

Участники обсуждают каждое требование и задают вопросы владельцу продукта, если необходимо. Очень важно оценить трудозатраты на разработку и тестирование. Также, при оценке имеет значение сложность требования и область тестирования. Поэтому, еще до начала покер-планирования для задачи в дополнение к приоритету, указанному владельцем продукта, следует указать уровень риска. Когда обсуждение по требованию закончено, каждый участник выбирает одну карту из своей колоды рубашкой вверх, оценивая таким образом риск. Затем все карты одновременно переворачиваются. Если все карты одинакового достоинства, то это и будет результатом оценки. Если нет, то участники обсуждают разницу в оценках, а после раунд повторяется, пока не будет достигнуто согласие либо за счет консенсуса, либо за счет применения правил (например, использовать среднее значение или самое высокое значение) для ограничения количества раундов. Такие обсуждения гарантируют хорошую оценку трудозатрат, необходимых для завершения поставленных владельцем продукта задач, а также, улучшают понимание коллективом того, что необходимо сделать [Cohn04].

3.3. Техники тестирования в проектах под управлением гибких методологий

Многие из техник и уровней тестирования, которые применимы к традиционным проектам, также могут быть использованы в проектах под управлением гибких методологий. Однако, для таких проектов существуют некоторые специфические различия в техниках тестирования, терминологии и документации, которые следует учитывать.

3.3.1. Критерий приемки, достаточное покрытие и другая информация для тестирования

Первоначальные требования в проектах под управлением гибких методологий – это пользовательские истории, расставленные в порядке убывания их приоритета. Начальные требования краткие и обычно соответствуют определенному формату

(см. Раздел 1.2.2). Нефункциональные требования, такие как удобство использования и производительность, также важны и могут быть определены как уникальные пользовательские истории или связаны с другими функциональными пользовательскими историями. Нефункциональные требования могут соответствовать определенному формату или стандарту, например, [ISO25000] или отраслевому стандарту.

Пользовательские истории служат важной основой тестирования. Кроме них, так же существуют:

- Опыт предыдущих проектов
- Существующие функции и показатели качества системы
- Код, архитектура и дизайн
- Профили пользователей (контекст, системные конфигурации и поведение пользователя)
- Информация о дефектах существующих и предыдущих проектов
- Классификация дефектов в таксономии дефектов
- Применимые стандарты (например, [DO178B] для программного обеспечения авионики)
- Риски качества (см. Раздел 3.2.1)

Во время каждой итерации разработчики создают код, реализующий функции, описанные в пользовательских историях, с соответствующими качественными характеристиками. Этот код проверяется и подтверждается приёмочным тестированием. Чтобы код был тестируемым, критерии приемки должны касаться следующих вопросов [Wiegers13]:

- Функциональное поведение: внешне наблюдаемое поведение с действиями пользователя в качестве входных данных, работающих под определенной конфигурацией
- Характеристики качества: как система выполняет заложенную программу. Эти характеристики также могут определяться как атрибуты качества или нефункциональные требования. Общие характеристики качества – это производительность, надежность, удобство использования и т.д.
- Сценарии использования: последовательность действий между внешним агентом (часто - пользователем) и системой для достижения конкретной цели или бизнес-задачи
- Бизнес-правила: действия, которые могут быть выполнены в системе при условиях, определенных внешними процедурами и ограничениями (например, процедуры, используемые в страховых компаниях для обработки исков)
- Внешние интерфейсы: описание связей между разрабатываемой системой и внешним миром. Внешние интерфейсы можно разделить на несколько типов (пользовательский интерфейс, интерфейс с другими системами и т.д.)
- Ограничения: любое ограничение проектирования и реализации, ограничивающее возможности разработчика. Устройства со встроенным программным обеспечением часто должны соответствовать физическим ограничениям: размер, вес, интеграция с интерфейсом

- Определения данных: пользователь может описывать формат, тип данных, допустимые значения и значения по умолчанию для элемента данных в составе сложной структуры бизнес-данных (например, почтовый индекс в почтовом адресе США)

В дополнение к пользовательским историям и связанным с ними критериями приемки, существует также другая информация, актуальная для тестировщика:

- Как система должна работать и использоваться
- Системные интерфейсы, которые могут быть использованы / доступны для тестирования
- Достаточна ли текущая поддержка инструмента
- Имеет ли тестировщик достаточно знаний и навыков, чтобы выполнить необходимые тесты

Тестировщики часто обнаруживают, что нуждаются в дополнительной информации (например, покрытие кода) во время итераций. Для получения этой информации они должны работать совместно с остальными членами команды. Актуальная информация играет роль в определении того, можно ли считать конкретную деятельность выполненной. Эта концепция, дающая определение понятию «Выполнено», имеет решающее значение в проектах под управлением гибких методологий и применяется несколькими способами, описанными в следующих разделах.

Уровни тестирования

Каждый уровень тестирования имеет свое определение понятию «Выполнено». В следующем списке приводятся примеры, актуальные для разных тестовых уровней:

Модульное тестирование

- 100% покрытие решений там, где это возможно, с тщательным анализом любых невыполнимых путей
- Статический анализ, выполняемый по всему коду
- Отсутствие незакрытых дефектов высокого приоритета (ранжирование на основе приоритета и серьезности)
- Отсутствие известных недопустимых технических долгов, остающихся в дизайне и коде [Jones11]
- Весь код, модульное тестирование и результаты модульного тестирования проверяются
- Все модульные тесты автоматизированы
- Важные характеристики находятся в согласованных пределах (например, производительность)

Интеграционное тестирование

- Все функциональные требования протестированы, включая позитивные и негативные тесты, количество тестов, основано на размере, сложности и рисках
- Все интерфейсы между модулями протестированы

- Все риски качества покрываются в соответствии с согласованным уровнем тестирования
- Отсутствие незакрытых дефектов высокого приоритета (приоритет определяется рисками и важностью)
- Информирование обо всех обнаруженных дефектах
- Все регрессионное тестирование автоматизировано там, где это возможно. Автоматизированные тесты хранятся в общем репозитории

Системное тестирование

- Сквозное тестирование пользовательских историй, новой и существующей функциональности
- Покрываются все типы пользователей
- Покрываются наиболее важные характеристики качества системы (например, производительность, работоспособность, надежность)
- Тестирование проводится в окружении, близком к тому, которое будут использовать пользователи, включая все аппаратное и программное обеспечение для всех поддерживаемых конфигураций, насколько это возможно
- Все риски качества покрываются в соответствии с согласованным уровнем тестирования
- Все регрессионное тестирование автоматизировано там, где это возможно. Автоматизированные тесты хранятся в общем репозитории
- Сообщается обо всех найденных дефектах, которые, по возможности, исправляются
- Отсутствие незакрытых дефектов высокого приоритета (приоритет определяется рисками и важностью)

Пользовательская история

Определение понятию «Выполнено» для пользовательской истории может быть дано с помощью следующих критериев:

- Пользовательские истории, выбранные для итерации, должны быть полными, понятными для команды и иметь подробные, тестируемые критерии приёмки
- Все элементы пользовательской истории определены и проверены, включая тесты приёмки пользовательских историй
- Задачи, необходимые для внедрения и тестирования выбранных пользовательских историй, определены и оценены командой

Ключевая функциональность

Определение понятию «Выполнено» для функциональности может охватывать несколько пользовательских историй и наборов пользовательских историй, включающих:

- Все составляющие пользовательские истории с критериями приёмки, определенными и утвержденными заказчиком
- Дизайн завершен без известного технического долга

- Написание кода завершено без известного технического долга или незавершенного рефакторинга
- Модульное тестирование выполнено и достигло определенного уровня покрытия
- Интеграционное и системное тестирование функциональности выполнено согласно определенным критериям покрытия
- Отсутствие незакрытых дефектов высокого приоритета
- Документация по функциональности закончена и включает в себя примечания к релизу, руководства пользователя и функции интерактивной справки

Итерация

Определение понятия «Выполнено» для итерации может включать следующее:

- Все ключевые функциональности готовы и индивидуально протестированы в соответствии с критерием приемки функциональности
- Любые некритичные дефекты, которые не могут быть исправлены в рамках ограниченной по времени итерации, добавлены в список задач по продукту с выставленным приоритетом
- Интеграция всей ключевой функциональности завершена и протестирована
- Документация написана, проверена и утверждена

На этом этапе программное обеспечение потенциально готово к релизу, потому что итерация успешно завершена, однако не все итерации приводят к релизу.

Релиз

Определение понятия «Выполнено» для релиза, который может включать в себя несколько итераций, охватывает следующие области:

- Покрытие: все соответствующие элементы базы тестирования для всего содержания релиза охвачены тестированием. Адекватность покрытия определяется новым и измененным функционалом, его сложностью и размером, а также связанными с этими рисками
- Качество: интенсивность дефектов (например, сколько дефектов было найдено за день или за время транзакции), плотность дефектов (например, сколько дефектов было найдено в сравнении с количеством пользовательских историй, попыток и/или атрибутов качества), оцененное количество остающихся дефектов, находящееся в допустимых пределах, последствия незакрытых и оставшихся дефектов (например, степень серьезности и приоритетность) определены и приняты. Остаточный уровень риска, связанный с каждым конкретным риском качества, определен и принят
- Время: если установленная дата поставки достигнута, необходимо учитывать особенности бизнеса, связанные с релизом или его отсутствием
- Стоимость: предполагаемые затраты на жизненный цикл следует использовать для подсчета рентабельности разрабатываемой системы (то есть предполагаемые затраты на разработку и поддержку должны быть значительно ниже ожидаемой полной стоимости продукта). Основная часть затрат на жизненный цикл продукта проистекает из обслуживания после релиза из-за дефектов, проходящих в эксплуатацию.

3.3.2. Применение разработки, управляемой приемочным тестированием

Разработка, управляемая приемочным тестированием – это подход в стиле «сначала тестирование». Тестовые сценарии создаются до реализации пользовательской истории. Тест-кейсы создаются гибкой командой, в том числе разработчиком, тестировщиком и заинтересованными лицами со стороны бизнеса [Adzic09] и могут быть ручными или автоматизированными. Первый шаг – это работа над спецификацией, когда происходит анализ пользовательской истории, ее обсуждение и документирование разработчиками, тестировщиками и представителями бизнеса. В ходе этого процесса фиксируются любые ошибки, неоднозначности и недочеты в пользовательской истории.

Следующий шаг – это создание тестов. Это может быть реализовано всей командой или отдельно тестировщиками. В любом случае независимый эксперт, такой как представитель бизнеса, утверждает тесты. Тесты – это примеры, описывающие специфические характеристики пользовательской истории. Они помогают команде правильно интерпретировать пользовательскую историю. Поскольку примеры и тесты – это одно и то же, эти термины часто используются как синонимы. Работа начинается с основных примеров и открытых вопросов.

Как правило, первые тесты – это позитивные тесты, описывающие правильное поведение без исключений и ошибок. Позитивные тесты включают в себя последовательность выполненных действий, если все идет, как ожидалось. После того, как позитивные тесты написаны, команда должна заняться негативными тестами и охватить нефункциональные атрибуты (например, производительность, удобство использования). Тесты составлены таким образом, чтобы каждое заинтересованное лицо могло понять их смысл. Тесты должны быть написаны простым языком, содержать необходимые предусловия (если таковые имеются), входные данные и связанные с ними результаты.

Примеры должны охватывать все характеристики пользовательской истории и не должны ее дополнять. Это значит, что пример не должен описывать какой-либо аспект пользовательской истории, не задокументированный в самой истории. Кроме того, никакие два примера не должны описывать одну и ту же характеристику пользовательской истории.

3.3.3. Функциональная и нефункциональная разработка тестов методом черного ящика

В гибком тестировании многие тесты создаются тестировщиками одновременно с пишущими код разработчиками. Так же, как разработчики пишут код, основываясь на пользовательских историях и критериях приемки, так и тестировщики разрабатывают тесты, основываясь на пользовательских историях и их критериях приемки. Некоторые виды тестов, такие как исследовательские тесты и тесты, основанные на опыте использования, создаются позже, во время выполнения испытаний, как описано в

разделе 3.3.4. Тестировщики могут применять традиционный тест-дизайн методом черного ящика, например, разбиение на классы эквивалентности, анализ граничных значений, таблицы решений, тестирование таблицы переходов. Например, анализ граничных значений может использоваться для выбора тестовых данных, когда клиент ограничен в количестве позиций, которые он может выбрать для покупки.

Во многих ситуациях нефункциональные требования могут быть записаны в виде пользовательских историй. Разработка тестов методом черного ящика, например, анализом граничных значений, может также использоваться для создания тестов для нефункциональных характеристик качества. Пользовательская история может содержать требования к производительности или надежности. Например, исполнение операции не может превысить лимит времени или количество операций может быть меньше определенного числа.

Для получения более детальной информации о разработке тестов методом черного ящика, предлагаем ознакомиться с учебными программами базового [ISTQB_FL_SYL] и продвинутого уровней [ISTQB_ALTA_SYL].

3.3.4. Применение исследовательского тестирования в гибких методологиях

Исследовательское тестирование важно для проектов под управлением гибких методологий из-за ограниченного времени тестового анализа и небольшого числа деталей пользовательских историй. Для достижения наилучших результатов, исследовательское тестирование должно сочетаться с методикой на основе опыта, в рамках стратегии реактивного тестирования, смешанной с другими стратегиями, такими как аналитическое тестирование на основе рисков, тестирование, основанное на аналитических требованиях, на моделях и анти-регрессионное тестирование. Стратегии тестирования и смешивание тестовых стратегий обсуждается в учебной программе базового уровня [ISTQB_FL_SYL].

В исследовательском тестировании тест-дизайн и выполнение тестов проводятся одновременно, руководствуясь подготовленной концепцией тестирования. Концепция тестирования предоставляет условия для покрытия при проведении тестирования с ограниченным временем. Во время исследовательского тестирования результаты большинства тестов задают направление следующему тесту. Те же методы белого ящика и черного ящика могут использоваться для тест-дизайна, когда производится предварительное тестирование.

Концепция тестирования может содержать следующую информацию:

- Актор: предполагаемый пользователь системы
- Цель: тема концепции, включающая цель, которую хочет достигнуть действующее лицо, то есть тестовые условия
- Установка: что должно быть выполнено для начала испытаний
- Приоритет: относительная важность данной концепции, основанная на приоритете связанной пользовательской истории или уровня риска
- Данные: любые данные, необходимые для выполнения концепции

- Действия: список идей о том, что пользователь может захотеть сделать с системой (например, вход в систему как суперпользователь) или что было бы интересно проверить (как позитивное, так и негативное тестирование)
- Замечания тестового оракула: как оценить продукт для определения правильных результатов (например, чтобы зафиксировать, что происходит на экране, и сравнить с тем, что написано в руководстве пользователя)
- Варианты: альтернативные действия и оценки в дополнение к уже написанным идеям

Для управления исследовательским тестированием можно использовать метод управления тестовыми сеансами. Сеанс – это период непрерывного тестирования, который может продолжаться от 60 до 120 минут. Тестовый сеанс включает следующее:

- Обзорный сеанс (чтобы понять, как это работает)
- Сеанс анализа (оценка функциональности или характеристик)
- Глубокое изучение (тупиковые ситуации, сценарии, взаимодействия)

Качество тестов зависит от способности тестировщиков задавать соответствующие вопросы о том, что тестируется. Например, следующие:

- Самое важное, что нужно знать об этой системе?
- Как система может выйти из строя?
- Что произойдет, если…?
- Что должно произойти, когда…?
- Удовлетворены ли потребности, требования и ожидания клиентов?
- Можно ли установить систему (и удалить при необходимости) всеми поддерживаемыми способами?

Во время проведения тестирования, тестировщик использует свою находчивость, интуицию, знания и навыки, чтобы найти возможные проблемы продукта. Тестировщику, также, необходимы хорошие знания и понимание тестируемого программного обеспечения, области бизнеса, способа применения программного обеспечения и определения того, когда система выходит из строя.

В тестировании может применяться набор эвристик. Эвристика может служить руководством для тестировщика в том, как проводить тестирование и оценивать результаты [Hendrickson]. К примеру:

- Граничные значения
- CRUD (Создать, Прочитать, Обновить, Удалить)
- Варианты конфигурации
- Прерывание (например, выход из системы, завершение работы или перезагрузка)

Для тестировщика важно документировать процесс насколько это возможно. В противном случае очень трудно вернуться назад и посмотреть, как была обнаружена проблема в системе. В следующем списке приведены примеры информации, которые может оказаться полезным документировать:

- Тестовое покрытие: какие входные данные были использованы, сколько было покрыто и сколько еще предстоит проверить

- Оценочные примечания: наблюдения во время тестирования, кажутся ли система и ключевая функциональность стабильными, были ли обнаружены какие-либо дефекты, что планируется в качестве следующего шага в соответствии с текущими наблюдениями, и другие идеи
- Список рисков/стратегий: какие риски были покрыты, и какие остаются одними из наиболее важных, будет ли применяться первоначальная стратегия, нужны ли какие-либо изменения
- Проблемы, вопросы и аномалии: какое-либо неожиданное поведение, любые вопросы относительно эффективности подхода, любые проблемы с идеями/попытками тестирования, тестовая среда, тестовые данные, непонимание функциональности, тестовые сценарии или тестируемая система
- Актуальное поведение: запись фактического поведения системы, которое должно быть сохранено (например, видео, копии экранов, выходные файлы данных)

Записанную информацию следует собирать и/или обобщать в виде инструментов управления статусами (например, инструментов управления тестированием, инструментов управления задачами, панель управления) таким образом, который облегчит заинтересованным сторонам понять текущий статус выполненного тестирования.

3.4. Инструментарий в проектах под управлением гибких методологий

Инструментарий, описанный в программе обучения базового уровня, актуален и используется тестировщиками в гибких командах. Не все инструменты используются одинаково, более того, некоторые инструменты больше подходят именно для проектов под управлением гибких методологий, чем для традиционных проектов. Например, несмотря на то, что инструменты управления тестированием, инструменты управления требованиями и инструменты управления инцидентами (инструменты отслеживания дефектов) могут использоваться гибкими командами, некоторые из этих команд выбирают универсальный инструментарий (например, управление жизненным циклом приложения или управление задачами), который предоставляет функции, относящиеся к гибкой разработке, такие как панели задач, диаграммы сгорания и пользовательские истории. Инструменты управления конфигурацией важны для тестировщиков в гибких командах из-за большого количества автоматических тестов на всех уровнях и необходимости хранить и управлять связанными с ними автоматизированными артефактами тестирования.

Помимо инструментов, описанных в программе обучения базового уровня, тестировщики в проектах под управлением гибких методологий могут также использовать инструменты, описанные в следующих подразделах. Эти инструменты используются всей командой для обеспечения совместной работы и обмена информацией, что является самым важным для методик гибкой разработки и тестирования программного обеспечения.

3.4.1. Система управления задачами и система отслеживания

В некоторых случаях гибкие команды используют физическую доску с историями / задачами (например, классную доску, пробковую доску) для управления и отслеживания пользовательских историй, тестов и других задач в каждом спринте. Другие команды будут использовать программное обеспечение для управления жизненным циклом приложений и управления задачами, включая электронные панели задач. Эти инструменты служат для следующих целей:

- Запись истории и задачи их разработки и тестирования, чтобы гарантировать, что ничто не потеряется во время спринта
- Сбор оценок членов команды по их задачам и автоматический расчет трудозатрат, необходимых для внедрения истории и поддержки эффективных сеансов планирования итераций
- Связь задачи разработки и тестовых заданий с одной и той же историей, чтобы предоставить полную картину трудозатрат команды, необходимых для реализации истории
- Сбор в одно целое обновления разработчика и тестировщика в статус задачи по завершению их работы, автоматически предоставляя текущий снимок состояния каждой истории, итерации и общего выпуска релиза
- Обеспечение визуального представления (с помощью показателей, диаграмм и информационных панелей) текущего состояния каждой пользовательской истории, итерации и выпуска, что позволит всем заинтересованным сторонам, в том числе сотрудникам из территориально распределенных групп, быстро проверить статус
- Интеграция с инструментами управления конфигурацией, которые могут позволить автоматизировать сопоставление обновления кода в репозитории и сборку с задачами, а в некоторых случаях автоматическое обновление статуса задач.

3.4.2. Коммуникации и инструменты совместного доступа к информации

В дополнение к электронной почте, документам и вербальной коммуникации гибкие команды часто используют три дополнительных типа инструментов для поддержки общения и обмена информацией: вики, обмен мгновенными сообщениями и совместный доступ к рабочему столу.

Вики позволяют командам создавать и делиться онлайн-базой знаний по различным аспектам проекта, включая следующие:

- Диаграммы функций продукта, обсуждения функций, шаблоны диаграмм, наклейки с решениями на доске и другая информация
- Инструменты и/или методики разработки и тестирования, полезные для других членов команды
- Метрики, диаграммы и информационные панели о статусе продукта, которые особенно полезны, когда вики интегрируется с другими инструментами, такими как сервер сборки и система управления задачами, поскольку вики может автоматически обновлять статус продукта

- Разговоры между членами команды, такие как обмен мгновенными сообщениями и электронной почтой, позволяющие предоставить доступ к необходимой информации всем остальным членам команды

  Мгновенный обмен сообщениями, аудио-конференции и видеоконференции обеспечивают следующие преимущества:
- Позволяют общаться в реальном времени членам команды, особенно распределенным командам
- Вовлекать распределенные команды в регулярные собрания
- Сокращать стоимости телефонных переговоров благодаря использованию технологий передачи голоса по IP-сетям, устраняя финансовые ограничения, которые могут затруднить взаимодействие в распределенных командах

  Инструменты совместного использования и захвата рабочего стола обеспечивают следующие преимущества:
- В распределенных командах могут появляться демонстрации продуктов, обзоры кода и возможна даже совместная работа
- Демонстрации продукта в конце каждой итерации, которые могут быть размещены в вики и доступны всей команде

  Эти инструменты должны использоваться в качестве дополнения и увеличения, но не замены живого общения в гибких командах.

3.4.3. Сборка программы и инструменты ее распространения

Как обсуждалось ранее в этой учебной программе, ежедневная сборка и развертывание программного обеспечения является ключом в практике в гибких командах. Это требует использования инструментов непрерывной интеграции и создания инструментов распространения. Использование, преимущества и риски этих инструментов были описаны ранее в разделе 1.2.4.

3.4.4. Инструменты управления конфигурацией

В гибких командах инструменты управления конфигурацией могут использоваться не только для хранения исходного кода и автоматизированных тестов. Ручные тесты и другие продукты тестирования часто хранятся в том же репозитории, что и исходный код продукта. Это обеспечивает прозрачность / отслеживаемость того, какие версии программного обеспечения были протестированы с какими конкретными версиями тестов, и позволяет быстро их изменять, не теряя информацию об истории изменений. Основными типами систем управления версиями являются централизованные системы управления версиями и распределенные системы управления версиями. Размер группы, структура, местоположение и требования к интеграции с другими инструментами определяют, какая система управления версиями подходит для конкретного проекта.

3.4.5. Средства разработки, реализации и исполнения тестов

Некоторые инструменты полезны для тестировщиков, работающих с применением

гибких методик, в определенных этапах процесса тестирования программного обеспечения. Хотя большинство этих инструментов не являются новыми или специфичными для гибких методологий, они предоставляют важные возможности, учитывая быстрое изменение проектов под их управлением.

- Инструменты проектирования тестов: использование инструментов, таких как ассоциативные карты, стало более популярным для быстрой разработки и определения тестов для новой функции
- Инструменты управления тестовыми сценариями: примером инструментов управления тестовыми сценариями, используемыми в гибких методологиях, может быть часть инструмента управления жизненным циклом приложения или инструментом управления задачами всей команды
- Средства подготовки и генерации тестовых данных: инструменты, которые генерируют данные для заполнения базы данных приложения, очень полезны, когда для тестирования приложения требуется много данных и комбинаций данных. Эти инструменты также могут помочь в реорганизации структуры базы данных, поскольку продукт претерпевает изменения во время проекта и реорганизовывает сценарии для генерации данных. Это позволяет быстро обновлять тестовые данные, когда происходят изменения. Некоторые средства подготовки тестовых данных используют источники производственных данных в качестве исходного материала и используют сценарии для удаления или обезличивания конфиденциальных данных. Другие средства подготовки тестовых данных могут помочь при проверке больших входных или выходных данных
- Инструменты загрузки тестовых данных: после того, как данные были созданы для тестирования, их необходимо загрузить в приложение. Ручной ввод данных часто занимает много времени и есть вероятность совершить ошибку, но инструменты загрузки данных делают процесс надежным и эффективным. Фактически, многие генераторы данных включают интегрированный компонент загрузки данных. В других случаях возможна также массовая загрузка с использованием систем управления базами данных
- Автоматизированные инструменты выполнения тестов: существуют инструменты выполнения тестов, которые в большей степени соответствуют гибкому тестированию. Специальные средства доступны как с коммерческими, так и с открытыми исходными кодами для поддержки первых тестовых подходов, таких как разработка, управляемая поведением, разработка, управляемая тестированием и приемочная разработка, управляемая тестированием. Эти инструменты позволяют тестировщикам и представителям бизнеса выражать ожидаемое поведение системы в таблицах или на естественном языке с использованием ключевых слов
- Инструментальные средства для изучения: инструменты, которые фиксируют и регистрируют действия, выполняемые приложением во время пробной тестовой сессии, полезны для тестировщика и разработчика, поскольку они фиксируют предпринятые действия. Это полезно, когда обнаружен дефект, поскольку действия, предпринятые до возникновения сбоя, были захвачены и могут быть использованы для

сообщения о дефекте разработчикам. Шаги ведения журнала, выполненные в ходе пробной тестовой сессии, могут оказаться полезными, если в конечном итоге тест включен в набор автоматических регрессионных тестов

3.4.6. Инструменты для облачных вычислений и виртуализации

Виртуализация позволяет одному физическому ресурсу (серверу) работать сопоставимо с большим количеством отдельных машин, но при этом потреблять меньше ресурсов. Когда используются виртуальные машины или облачные средства, команды имеют большее количество серверов, доступных для разработки и тестирования. Это может помочь избежать задержек, связанных с ожиданием физических серверов. Предоставление нового сервера или восстановление сервера более эффективно с помощью возможностей моментальных снимков, встроенных в большинство средств виртуализации. Некоторые средства управления тестированием теперь используют технологии виртуализации для серверов моментальных снимков в момент обнаружения ошибки, позволяя тестировщикам использовать моментальный снимок совместно с разработчиками, изучающими ошибку.

4. Ссылки

4.1. Стандарты

- [DO-178B] RTCA/FAA DO-178B, Software Considerations in Airborne Systems and Equipment Certification, 1992.
- [ISO25000] ISO/IEC 25000:2005, Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE), 2005.

4.2. Документы ISTQB

- [ISTQB_ALTA_SYL] ISTQB Advanced Level Test Analyst Syllabus, Version 2012
- [ISTQB_ALTM_SYL] ISTQB Advanced Level Test Manager Syllabus, Version 2012
- [ISTQB_FA_OVIEW] ISTQB Foundation Level Agile Tester Overview, Version 1.0
- [ISTQB_FL_SYL] ISTQB Foundation Level Syllabus, Version 2011 s

4.3. Книги

[Aalst13] Leo van der Aalst and Cecile Davis, “TMap NEXT® in Scrum,” ICT-Books.com, 2013.
[Adzic09] Gojko Adzic, “Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing,” Neuri Limited, 2009.
[Anderson13] David Anderson, “Kanban: Successful Evolutionary Change for Your Technology Business,” Blue Hole Press, 2010.
[Beck02] Kent Beck, “Test-driven Development: By Example,” Addison-Wesley Professional, 2002.
[Beck04] Kent Beck and Cynthia Andres, “Extreme Programming Explained: Embrace Change, 2e” Addison-Wesley Professional, 2004.
[Black07] Rex Black, “Pragmatic Software Testing,” John Wiley and Sons, 2007.
[Black09] Rex Black, “Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing, 3e,” Wiley, 2009.
[Chelimsky10] David Chelimsky et al, “The RSpec Book: Behavior Driven Development with Rspec, Cucumber, and Friends,” Pragmatic Bookshelf, 2010.
[Cohn04] Mike Cohn, “User Stories Applied: For Agile Software Development,” Addison-Wesley Professional, 2004.
[Crispin08] Lisa Crispin and Janet Gregory, “Agile Testing: A Practical Guide for Testers and Agile Teams,” Addison-Wesley Professional, 2008.
[Goucher09] Adam Goucher and Tim Reilly, editors, “Beautiful Testing: Leading Professionals Reveal How They Improve Software,” O'Reilly Media, 2009.

[Jeffries00] Ron Jeffries, Ann Anderson, and Chet Hendrickson, “Extreme Programming Installed,” Addison-Wesley Professional, 2000.
[Jones11] Capers Jones and Olivier Bonsignour, “The Economics of Software Quality,” Addison-Wesley Professional, 2011.
[Linz14] Tilo Linz, “Testing in Scrum: A Guide for Software Quality Assurance in the Agile World,” Rocky Nook, 2014.
[Schwaber01] Ken Schwaber and Mike Beedle, “Agile Software Development with Scrum,” Prentice Hall, 2001.
[vanVeenendaal12] Erik van Veenendaal, “The PRISMA approach”, Uitgeverij Tutein Nolthenius, 2012.
[Wiegers13] Karl Wiegers and Joy Beatty, “Software Requirements, 3e,” Microsoft Press, 2013.