Russian Software Testing Qualifications Board

Сертифицированный тестировщик в сфере гибких методологий

Программа обучения

Версия 2014

International Software Testing Qualifications Board

ISTQB® International Software Testing Qualifications Board

История изменений

Версия	**Дата**	**Замечания**
Syllabus v0.1	26.07.2013	Создание исходной версии документа
Syllabus v0.2	16.09.2013	Добавление комментариев рабочей группы к версии v0.1
Syllabus v0.3	20.10.2013	Добавление комментариев рабочей группы к версии v0.2
Syllabus v0.7	16.12.2013	Добавление комментариев к предварительной версии v0.3
Syllabus v0.71	20.12.2013	Обновления рабочей группы v0.7
Syllabus v0.9	30.01.2014	Бета версия
Syllabus 2014	31.05.2014	Общедоступная версия
Syllabus 2014	30.09.2014	Исправление незначительных опечаток
RSTQB 2014	30.03.2018	Перевод на русский язык

Оглавление

Благодарности

Документ подготовлен рабочей группой International Software Testing Qualifications Board базового уровня.
Благодарим команду рецензентов и Национальные коллегии за их предложения и вклад.
На момент завершения подготовки расширенной программы обучения гибким методологиям базового уровня в рабочей группе состояли: Rex Black (председатель), Bertrand Cornanguer (заместитель председателя), Gerry Coleman (ответственный за цели обучения), Debra Friedenberg (ответственный за экзамен), Alon Linetzki (бизнес и маркетинг), Tauhida Parveen (редактор) и Leo van der Aalst (руководитель разработки).

Авторы: Rex Black, Anders Claesson, Gerry Coleman, Bertrand Cornanguer, Istvan Forgacs, Alon Linetzki, Tilo Linz, Leo van der Aalst, Marie Walsh, and Stephan Weber.
Внутренние рецензенты: Mette Bruhn-Pedersen, Christopher Clements, Alessandro Collino, Debra Friedenberg, Kari Kakkonen, Beata Karpinska, Sammy Kolluru, Jennifer Leger, Thomas Mueller, Tuula Pääkkönen, Meile Posthuma, Gabor Puhalla, Lloyd Roden, Marko Rytkönen, Monika Stoecklein-Olsen, Robert Treffny, Chris Van Bael, and Erik van Veenendaa.

Авторы перевода: Маргарита Трофимова (руководитель рабочей группы), Александр Александров (редактор), Екатерина Акулова, Екатерина Белая, Андрей Конушин, Елена Костина, Александр Куцан, Анастасия Мордвинцева, Антон Романов, Александра Титова.

Команда также выражает благодарность членам Национальных коллегий и Экспертного сообщества в области гибких методологий, которые приняли участие в комментировании, голосовании и рецензировании расширенной программы обучения гибким методологиям базового уровня: Dani Almog, Richard Berns, Stephen Bird, Monika Bögge, Afeng Chai, Josephine Crawford, Tibor Csöndes, Huba Demeter, Arnaud Foucal, Cyril Fumery, Kobi Halperin, Inga Hansen, Hanne Hinz, Jidong Hu, Phill Isles, Shirley Itah, Martin Klonk, Kjell Lauren, Igal Levi, Rik Marselis, Johan Meivert, Armin Metzger, Peter Morgan, Ninna Morin, Ingvar Nordstrom, Chris O'Dea, Klaus Olsen, Ismo Paukamainen, Nathalie Phung, Helmut Pichler, Salvatore Reale, Stuart Reid, Hans Rombouts, Petri Säilynoja, Soile Sainio, Lars-Erik Sandberg, Dakar Shalom, Jian Shen, Marco Sogliani, Lucjan Stapp, Yaron Tsubery, Sabine Uhde, Stephanie Ulrich, Tommi Välimäki, Jurian Van de Laar, Marnix Van den Ent, António Vieira Melo, Wenye Xu, Ester Zabar, Wenqiang Zheng, Peter Zimmerer, Stevan Zivanovic, and Terry Zuo.

Документ был официально одобрен к публикации Генеральной Ассамблеей International Software Testing Qualifications Board 31 мая 2014.

0. Введение

0.1 Цели документа

Программа обучения является основой для подготовки к международной сертификации базового уровня для тестировщиков, работающих в области гибких методологий. International Software Testing Qualifications Board предоставляет эту программу:

- Национальным коллегиям для перевода и проведения аккредитации учебных курсов на местном языке. Национальные коллегии могут адаптировать программу согласно языковым потребностям и изменять ссылки, адаптируя под местные публикации.
- Экзаменационным коллегиям для составления экзаменационных вопросов на местном языке, адаптируя цели обучения для каждой программы.
- Поставщикам учебных курсов для подготовки программы обучения и определения соответствующих методов обучения.
- Кандидатам на сертификацию для подготовки к экзамену (как часть учебного курса или независимо)
- Международным обществам разработки систем и программного обеспечения для повышения квалификации в тестировании ПО и систем, или как основу для книг и статей.

ISTQB оставляет за собой право предоставлять эту программу и в других целях по предварительной просьбе и письменному разрешению.

0.2 Краткое содержание

Документ включает в себя следующую информацию:

- Бизнес цели программы
- Резюме программы
- Взаимосвязь между программами
- Описание уровней понимания (К-уровни)
- Приложения

0.3 Цели обучения

Обучение по программе предназначено для достижения бизнес целей, а также для подготовки к сертификационному экзамену базового уровня – сертифицированный тестировщик в области гибких методологий (Certified Tester Foundation Level — Agile Tester Certification). В целом все части данной программы проверяются на уровне К1. Это значит, что кандидат должен идентифицировать, запоминать и воспроизводить термин или концепцию. Конкретные цели обучения на уровнях К1, К2 и К3 показаны в начале каждой главы.

1. Гибкая методология разработки программного обеспечения – 150 минут

Ключевые слова

Манифест гибкой методологии разработки программного обеспечения, гибкая методология разработки программного обеспечения, инкрементная модель разработки, итеративная модель разработки, жизненный цикл программного обеспечения, автоматизация тестирования, базис тестирования, разработка на основе тестов, тестовый предсказатель, пользовательская история

Цели обучения

1.1. Основы гибкой методологии разработки программного обеспечения

FA-1.1.1 (K1) Вспомнить основную концепцию гибкой методологии разработки программного обеспечения, основанной на манифесте гибкой разработки
FA-1.1.2 (K2) Понять преимущества командного подхода
FA-1.1.3 (K2) Понять преимущества ранней и частой обратной связи

1.2. Аспекты подходов гибких методологий

FA-1.2.1 (K1) Вспомнить подходы гибких методологий разработки программного обеспечения
FA-1.2.2 (K3) Написать тестируемые пользовательские истории в сотрудничестве с разработчиками и представителями бизнеса
FA-1.2.3 (K2) Понять, как ретроспективы могут быть использованы в качестве механизма улучшения процесса в проектах с гибкой методологией
FA-1.2.4 (K2) Понять применение и цель непрерывной интеграции
FA-1.2.5 (K1) Знать различия между планированием итерации и выпуска, и как тестировщик добавляет ценность для каждой из этих активностей.

1.1. Основы гибкой методологии разработки программного обеспечения

Тестировщик в проектах с гибкой методологией будет работать иначе, чем работающий в традиционных проектах. Тестировщик должен понимать ценности и принципы, лежащие в основе проектов с гибкой методологией, и как тестировщики являются неотъемлемой частью командного подхода вместе с разработчиками и представителями бизнеса. Участники проектов с гибкой методологией взаимодействуют друг с другом рано и часто, это помогает устранять дефекты как можно раньше и разрабатывать качественный продукт.

1.1.1. Гибкая методология разработки программного обеспечения и ее Манифест

В 2001 году группа людей, представляющих наиболее широко используемые облегченные методологии разработки программного обеспечения, договорились об общем наборе ценностей и принципов, которые стали известны как Манифест гибкой разработки программного обеспечения [Agilemanifesto]. Он включает заявления четырех ценностей:

- Люди и взаимодействия важнее процессов и инструментов
- Работающий продукт важнее исчерпывающей документации
- Сотрудничество с заказчиком важнее согласования условий контракта
- Готовность к изменениям важнее следования плану

Манифест гибкой разработки утверждает, что, хотя понятия справа имеют ценность, понятия слева имеют большую ценность.

Люди и взаимодействия
Гибкая разработка сильно ориентирована на людей. Команды создают программное обеспечение, и команды могут работать более эффективно благодаря непрерывному взаимодействию вне зависимости от используемых инструментов и процессов.

Работающее программное обеспечение
С точки зрения клиента, работающее программное обеспечение гораздо более полезно и ценно, чем слишком подробная документация, и это обеспечивает возможность дать группе разработчиков быструю обратную связь. Кроме того, поскольку работающее программное обеспечение, пусть даже с урезанной функциональностью, доступно гораздо раньше в жизненном цикле разработки, гибкая разработка может принести значительное преимущество от времени выхода на рынок. Поэтому гибкая разработка особенно полезна в быстро меняющихся бизнес-средах, где про-

блемы и/или решения не ясны или где бизнес хочет внедрять инновации в новых проблемных сферах деятельности.

Сотрудничество с заказчиком

Пользователи часто сталкиваются с большими трудностями при определении системы, которая им требуется. Непосредственное сотрудничество с пользователем повышает вероятность понимания того, что требуется клиенту. В то время, как наличие контрактов с клиентами может быть важным, работа в постоянном и тесном сотрудничестве с ними, вероятно, принесет больше успеха проекту.

Готовность к изменениям

Изменения в проектах программного обеспечения неизбежны. Окружение, в котором работает бизнес, законодательство, конкуренция, продвижение технологий и другие факторы могут иметь большое влияние на проект и его цели. Эти факторы должны учитываться в процессе разработки. Таким образом, наличие гибкости в методах принятия изменений важнее, чем просто строго придерживаться плана.

Принципы

Суть ценностей Манифеста гибкой разработки отражены в 12 принципах:

- Наивысшим приоритетом является удовлетворение потребностей заказчика, благодаря ранней и непрерывной поставке ценного программного обеспечения.
- Изменение требований приветствуется даже на поздних стадиях разработки. Гибкие процессы используют изменения для конкурентного преимущества заказчика.
- Поставляйте работоспособное программное обеспечение часто: с периодичностью от нескольких недель до нескольких месяцев, предпочитая более короткие интервалы.
- В течении всего проекта представители бизнеса и разработчики должны ежедневно работать вместе.
- Создавайте проекты в окружении мотивированных личностей. Предоставьте им окружение и поддержку, в которой они нуждаются, и доверьте им сделать работу.
- Наиболее эффективный и результативный способ передачи информации в команду и внутри нее - это непосредственное общение.
- Работающее программное обеспечение - основной показатель прогресса.
- Гибкие процессы способствуют устойчивому процессу разработки. Спонсоры, разработчики и пользователи должны быть в состоянии поддерживать постоянный темп бесконечно.
- Постоянное внимание к техническому совершенству и хорошему дизайну повышает гибкость.

- Простота - искусство максимизации объема невыполненной работы - необходима.
- Лучшая архитектура, требования и дизайн появляются в самоорганизующихся командах.
- На регулярной основе команда размышляет, как стать более эффективной, затем соответственно корректирует поведение.

Различные гибкие методологии предлагают предписывающие практики, чтобы привести эти ценности и принципы в действие.

1.1.2. Командный подход

Командный подход означает привлечение каждого, кто обладает знаниями и навыками, необходимыми для обеспечения успеха проекта. Команда включает представителей клиента и других заинтересованных лиц бизнеса, которые определяют функции продукта. Команда должна быть относительно небольшой, успешные команды наблюдались немногочисленные - от 3 до 9 человек. В идеале, вся команда разделяет одно и тоже рабочее пространство, поскольку совместное размещение сильно облегчает общение и взаимодействие. Командный подход поддерживается на ежедневных митингах (см. Раздел 2.2.1), в которых участвуют все члены команды, где сообщается о ходе работы и отмечаются любые препятствия на пути прогресса. Командный подход способствует более эффективной и результативной командной динамике.
Использование командного подхода в разработке продукта является одним из основных преимуществ гибкой разработки. Эти преимущества включают:

- Повышение коммуникации и сотрудничества внутри команды
- Предоставление различных наборов навыков в команде для использования в интересах проекта
- Общая ответственность за качество

Вся команда отвечает за качество в гибких проектах. Суть командного подхода заключается в том, что тестировщики, разработчики и представители бизнеса работают вместе не каждом этапе процесса разработки. Тестировщики будут тесно сотрудничать и с разработчиками, и с представителями бизнеса, чтобы обеспечить достижение желаемых уровней качества. Это включает поддержку и сотрудничество с представителями бизнеса, чтобы помочь им создать подходящие приемочные тесты, работу с разработчиками для согласования стратегии тестирования и принятии решения о подходах автоматизации тестирования. Таким образом, тестировщики могут передавать и распространять знания о тестировании другим членам команды и влиять на разработку продукта.
Вся команда участвует в любых консультациях или встречах, на которых функционал продукта представляется, анализируется или оценивается. Концепция привле-

чения тестировщиков, разработчиков и представителей бизнеса во всех обсуждениях функционала называется силой трех [Crispin08].

1.1.3. Ранняя и частая обратная связь

Гибкие проекты имеют короткие итерации, позволяющие проектной команде получать раннюю и постоянную обратную связь о качестве продукта на протяжении всего жизненного цикла разработки. одним из способов обеспечения быстрой обратной связи является непрерывная интеграция (см. раздел 1.2.4).
Когда используются подходы последовательной разработки, клиент часто не видит продукт до тех пор, пока проект не будет завершен. В этот момент для команды разработчиков зачастую слишком поздно эффективно решать любые проблемы, которые могут возникнуть у клиента. Получая часто обратную связь от клиента в процессе проекта, гибкие команды могу включить больше новых изменений в процессе разработки продукта. Ранняя и частая обратная связь помогает команде сосредоточится на функциях с наивысшим бизнес приоритетом или связанном с ними риском, и они предоставляются клиенту в первую очередь. Это также помогает лучше управлять командой, поскольку способность команды прозрачна для всех. Например, сколько работы мы можем сделать в спринте или итерации? Что может помочь нам двигаться быстрее? Что мешает нам сделать это?
Преимущества ранней и частой обратной связи включает:

- Избежание недоразумений в требованиях, которые возможно, не были обнаружены до тех пор в цикле разработки, пока они не стали более дорогостоящими для исправления.
- Уточнение требований пользователей к функционалу, делая его доступным для использования пользователем на ранней стадии. Таким образом, продукт лучше отражает то, что хочет клиент.
- Обнаружение (через непрерывную интеграцию), изоляция и решение проблем качества на ранней стадии.
- Предоставление информации гибкой команде относительно производительности и возможности поставки.
- Последовательная поддержка темпа проекта.

1.2. Аспекты подходов гибких методологий

Существует много подходов гибких методологий, используемых в организациях. Общепринятые практики большинства гибких методологий в организациях включают совместное создание пользовательских историй, ретроспективы, непрерывной интеграции и планирования каждой итерации, как и релиза. В этом подразделе описаны

некоторые подходы гибких методологий.

1.2.1. Подходы гибкой методологии разработки программного обеспечения

Существует несколько подходов гибкой методологии, каждый из которых по-разному реализует ценности и принципы Манифеста. В этой программе рассмотрены три представленных подхода гибких методологий: Экстремальное программирование (XP), Скрам, Канбан.

Экстремальное программирование

Экстремальное программирование (XP), первоначально представленное Кентом Бек [Beck04], является подходом гибкой методологии разработки программного обеспечения, характеризуемым определенными ценностями, принципами и практиками разработки.

XP включает в себя пять ценностей для руководства разработкой:

- коммуникация,
- простота,
- обратная связь,
- смелость
- уважение.

XP описывает набор принципов как дополнительные рекомендации:

- человечность,
- экономичность,
- взаимная выгода,
- сходство,
- улучшение,
- разнообразие,
- обдумывание,
- течение,
- возможность,
- избыточность,
- неудача,
- качество,
- маленькими шажками,
- принятая ответственность.

XP описывает тринадцать основных практик:

- сидеть вместе,
- команда как одно целое,
- информативное рабочее пространство,
- напряженная работа,

- парное программирование,
- истории,
- недельный цикл,
- ежеквартальный цикл,
- слабость,
- десятиминутная сборка,
- непрерывная интеграция,
- сначала тесты,
- инкрементное проектирование.

Многие из подходов гибких методологий разработки программного обеспечения, использующиеся сегодня, зависят от XP его ценностей и принципов. Например, команды в гибких методологиях, придерживающиеся Срам, часто используют методы XP.

Скрам

Скрам – это система управления гибкой методологией, которая содержит следующие составные инструменты и практики [Schwaber01]:

- Спринт: Скрам делит проект на итерации (называемые спринтами) фиксированной длины (обычно от двух до четырех недель)
- Приращение продукта: каждый спринт приводит к потенциально выпускаемому/поставляемому продукту (называемому приращением).
- Набор задач продукта: владелец продукта управляет приоритезированным списком запланированных функций продукта (называемым набором задач продукта)
- Набор задач спринта: в начале каждого спринта команда Скрам выбирает набор функций с наивысшим приоритетом (называемый набором задач спринта) из набора задач продукта. Поскольку команда Скрам, а не владелец продукта, выбирает функции, которые будут реализованы в рамках спринта, этот выбор считается принципом вытягивания, а не принципом проталкивания.
- Критерии готовности: чтобы убедиться, что в конце каждого спринта есть потенциальный релиз продукта, команда Скрам обсуждает и определяет критерии завершения спринта. Дискуссия расширяет понимание командой набор задач и требований продукта.
- Временная шкала: только те задачи, требования или функции, которые команда ожидает завершить за спринт, являются набором задач спринта. Если команда разработчиков не может завершить задачу за спринт, связанные функции продукта удаляются из спринта и задача возвращается в набор задач продукта. Временная шкала применяется не только к задачам, но и в других ситуациях (например, соблюдение времени начала и окончания встречи).
- Прозрачность: команда разработчиков ежедневно сообщает и обновляет статус спринта на встрече, называемой ежедневным скрам-митингом. Это делает содержание и прогресс текущего спринта, включая результаты тестирования, видимыми

для команды, руководства и всех заинтересованных сторон. Например, команда разработчиков может отобразить статус спринта на доске.

Скрам определяет три роли:

- Скрам мастер: обеспечивает выполнение и соблюдение практик и правил Скрам, а также устраняет любые нарушения, проблемы с ресурсами или другие препятствия, которые могут помешать команде следовать практикам и правилам. Этот человек не руководитель команды, а тренер.
- Владелец продукта: представляет клиента, а также создает, поддерживает и приоритезирует набор задач продукта. Он не руководитель команды.
- Команда разработки: разрабатывает и тестирует продукт. Команда самоорганизована: нет руководителя команды, поэтому команда сама принимает решения. Команда также является кросс-функциональной (см. Раздел 2.3.2 и Раздел 3.1.4).

Скрам (в отличие от XP) не предписывает конкретные методы разработки программного обеспечения (например, сначала тесты). Кроме того, Скрам не дает рекомендаций как должно проводиться тестирование в проекте Скрам.

Канбан

Канбан [Anderson13] это подход к управлению, который иногда используется в проектах с гибкой методологией. Основная цель - визуализация и оптимизация потока работ производственной цепи. Канбан использует три инструмента [Linz14]:

- Канбан доска: Цепочка создания ценности, которую нужно контролировать, Скрам визуализируется Канбан доской. Каждый столбец отображает состояние, которое представляет набор связанных действий, например, разработку или тестирование. Элементы или задачи, которые должны быть обработаны, символически изображаются карточками, перемещающимися слева направо по доске путем изменения состояний.
- Лимит незавершенных работ: Количество параллельных активных задач строго ограничено. Это контролируется максимальным количеством карточек, разрешенных для состояния и/или глобально для доски. Всякий раз, когда на состоянии есть свободное место, сотрудник перетягивает карточку от состояния-предшественника.
- Время выполнения: Канбан используется для оптимизации непрерывного потока задач для минимизации (среднего) времени выполнения задач.

Канбан имеет сходство со Скрамом. В обоих подходах визуализация активных задач (например, на общей доске) обеспечивает прозрачность содержания и прогресс задач. Задачи, которые еще не запланированы, ждут в наборе задач и переходят на Канбан доску, как только появится новое место (производственные мощности).
Итерации или спринты необязательны в Канбан. Процесс Канбан позволяет выпускать свои продукты функция за функцией, а не как часть релиза продукта. Таким об-

разом, временная шкала как механизм синхронизации необязателен, в отличии от Скрам, который синхронизирует все задачи в спринте.

1.2.2. Совместное создание пользовательской истории

Плохие спецификации часто являются основной причиной провала проекта. Проблемы со спецификацией могут возникнуть в результате отсутствия у пользователей понимания их истинных потребностей, отсутствия глобального видения системы, избыточного или противоречивого функционала и других недопониманий. В разработке по гибким методологиям пользовательские истории написаны для сбора требований с точки зрения разработчиков, тестировщиков и представителей бизнеса. При последовательной разработке это общее видение функции осуществляется посредством формальных рецензирований после написания требований; в разработке по гибким методологиям это общее видение достигается посредством частых неформальных рецензирований, пока пишутся требования.

Пользовательские истории должны учитывать, как функциональные, так и нефункциональные свойства. Каждая история содержит критерии приемки этих свойств. Эти критерии должны определяться в сотрудничестве с представителями бизнеса, разработчиками и тестировщиками. Они предоставляют разработчикам и тестировщикам расширенное видение функции, которую будут подтверждать представители бизнеса. Команда в гибкой методологии считает, что задача завершена, когда выполнен список критериев приемки.

Как правило, уникальное видение тестировщика улучшит пользовательскую историю, указав потерянные детали или нефункциональные требования. Тестировщик может внести свой вклад, обратившись к представителям бизнеса по открытым вопросам пользовательской истории, предложив способы проверки пользовательской истории и подтвердив критерии приемки.

При совместном написании пользовательской истории можно использовать такие методы, как «мозговой штурм» и «составление карт памяти». Тестировщик может использовать технику INVEST [INVEST]:

- Независимая (Independent)
- Обсуждаемая (Negotiable)
- Полезная (Valuable)
- Поддающаяся оценке (Estimable)
- Небольшая (Small)
- Тестируемая (Testable)

Согласно концепции «трех С» [Jeffries00], пользовательская история - это сочетание трех элементов:

- Карточка (Card): Карточка представляет собой физический носитель, описывающий пользовательскую историю. Она определяет требование, его критичность, ожидаемую продолжительность разработки и тестирования, а также критерии приемки этой истории. Описание должно быть точным, поскольку оно будет использоваться в наборе задач продукта.
- Диалог (Conversation): Посредством диалога разъясняется, как будет использоваться программное обеспечение. Диалог может быть задокументирован или оставаться устным. Тестировщики, имеющие другую точку зрения от разработчиков и представителей бизнеса [ISTQB_FL_SYL], привносят ценный вклад в обмен мыслями, мнениями и опытом. Диалог начинается во время этапа планирования релиза и продолжается даже после того, как история запланирована.
- Подтверждение (Confirmation): Критерии приемки, обсуждаемые в диалоге, используются для подтверждения того, что история выполнена. Эти критерии приемки могут охватывать несколько пользовательских историй. Для оценки критериев следует использовать как позитивные, так и негативные тесты. Во время подтверждения различные участники играют роль тестировщика. К ним могут относиться разработчики, а также специалисты по производительности, безопасности, совместимости и другим характеристикам качества. Чтобы подтвердить, что история выполнена, конкретные критерии приемки должны быть протестированы и должно быть продемонстрировано, что они достигнуты.

Команды в гибких методологиях различаются тем, как они документируют пользовательские истории. Независимо от подхода к документированию пользовательских историй, документация должна быть краткой, достаточной и необходимой.

1.2.3. Ретроспектива

В разработке по гибкой методологии ретроспектива - это встреча в конце каждой итерации для обсуждения того, что было успешным, что можно было бы улучшить, и как включить улучшения и сохранить успехи в будущих итерациях. Ретроспективы охватывают такие темы, как процесс, люди, организации, отношения и инструменты. Регулярно проводимые итоговые встречи, после которых следуют соответствующие действия, имеют решающее значение для самоорганизации и постоянного совершенствования разработки и тестирования.

Результатом ретроспективы могут стать решения по улучшению, связанные с тестированием, ориентированные на эффективность, производительность тестирования, на качество тестовых сценариев и удовлетворенность команды. Они также могут учитывать возможности тестирования приложений, пользовательских историй, функций или системных интерфейсов. Анализ первопричин дефектов может привести к совершенствованию тестирования и разработки. В целом команды должны реализовать только несколько улучшений за итерацию. Это позволяет непрерывно проводить улучшения с постоянным темпом.

Расписание и организация ретроспективы зависят от конкретного метода гибких методологий. Представители бизнеса и команда присутствуют на каждой ретроспективе в качестве участников, а координатор организует и проводит встречу. В некоторых случаях команды могут приглашать на встречу других участников.
Тестировщики должны играть важную роль в ретроспективе. Тестировщики являются частью команды и привносят свой уникальный взгляд [ISTQB_FL_SYL, раздел 1.5]. Тестирование есть в каждом спринте и жизненно важно для успеха. Все члены команды, тестировщики и не-тестировщики, могут вносить вклад как в тестирование, так и в активности, не связанные с тестированием.
Ретроспектива должна происходить в профессиональной среде, характеризующейся взаимным доверием. Атрибуты успешной ретроспективы такие же, как и для любого другого рецензирования, описанного в учебном плане Базового Уровня [ISTQB_FL_SYL, раздел 3.2].

1.2.4. Непрерывная интеграция

Осуществление приращения продукта требует надежного, рабочего, интегрированного программного обеспечения в конце каждого спринта. Непрерывная интеграция решает эту задачу, объединяя все изменения, внесенные в программное обеспечение, и регулярно интегрируя все измененные компоненты по крайней мере один раз в день. Управление конфигурацией, компиляция, сборка программного обеспечения, развертывание и тестирование - все в едином, автоматическом, повторяемом процессе. Поскольку разработчики постоянно интегрируют свою работу, постоянно собирают и постоянно тестируют, дефекты в коде обнаруживаются быстрее.

После кодирования, отладки и фиксации кода разработчиков в общем хранилище исходного кода процесс непрерывной интеграции состоит из следующих автоматизированных действий:

- Статический анализ кода: выполнение статического анализа кода и создание отчетов о результатах
- Компиляция: компиляция и линковка кода, генерация исполняемых файлов
- Модульное тестирование: выполнение модульных тестов, проверка покрытия кода и создание отчетов о результатах тестирования
- Развертывание: установка сборки в тестовом окружении
- Интеграционное тестирование: выполнение интеграционных тестов и создание отчетов о результатах
- Отчет (сводная таблица): публикация статуса всех этих активностей в общедоступном местоположении или отправка статуса команде по электронной почте

Автоматизированный процесс сборки и тестирования происходит ежедневно, рано и быстро обнаруживает интеграционные ошибки. Непрерывная интеграция позволяет тестировщикам в гибких методологиях регулярно запускать автоматические тесты, в

некоторых случаях, как часть процесса непрерывной интеграции, и предавать быструю обратную связь команде о качестве кода.

Эти результаты тестирования видны всем членам команды, особенно когда автоматизированные отчеты интегрируются в процесс. Автоматическое регрессионное тестирование может быть непрерывным на протяжении всей итерации. Хорошие автоматические регрессионные тесты охватывают как можно больше функциональности, включая пользовательские истории, реализованные в предыдущих итерациях. Хорошее покрытие автоматическими регрессионными тестами помогает поддерживать создание (и тестирование) больших интеграционных систем. Когда регрессионное тестирование автоматизировано, тестировщики в гибких методологиях освобождаются, чтобы сконцентрировать ручное тестирование на новых функциях, внесенных изменениях и подтверждающем тестировании исправленных дефектов.

Организации, использующие непрерывную интеграцию, в дополнение к автоматическим тестам обычно используют инструменты сборки для непрерывного контроля качества. В дополнение к запуску модульных и интеграционных тестов, такие инструменты могут запускать дополнительные статические и динамические тесты, измерять и профилировать производительность, извлекать и оформлять документацию на основании исходного кода и облегчать проведение процессов обеспечения качества. Это постоянное применение контроля качества нацелено на улучшение качества продукта, а также сокращение времени, затрачиваемого на его поставку, взамен традиционной практики применения контроля качества после завершения всей разработки.

Инструменты сборки могут быть связаны с инструментами автоматического развертывания, которые могут доставать соответствующую сборку с сервера непрерывной интеграции или с сервера сборки и развертывать его в одной или нескольких средах: разработки, тестирования, промежуточной или даже производственной среды. Это уменьшает ошибки и задержки, связанные с использованием специализированного персонала или программистов для установки релизов в этих средах.

Непрерывная интеграция может обеспечить следующие преимущества:

- Позволяет более раннее обнаружение и более простой анализ первопричины проблем интеграции и конфликтующих изменений
- Дает команде разработчиков регулярную информацию о работоспособности кода
- Сохраняет версию программного обеспечения, протестированного в течение дня разработанной версии
- Снижает риск регрессии, связанный с рефакторингом кода, благодаря быстрому повторному тестированию основного кода после каждого небольшого набора изменений

- Обеспечивает уверенность в том, что ежедневная работа разработки основывается на прочном фундаменте
- Создает прогресс видимого движения продукта к завершению, поощряя разработчиков и тестировщиков
- Устраняет риски расписания, связанные с интеграционным «большим взрывом»
- Обеспечивает постоянную доступность исполняемого программного обеспечения на протяжении всего спринта для тестирования, демонстрации или образовательных целей
- Уменьшает повторяющиеся действия ручного тестирования
- Обеспечивает быструю обратную связь по решениям, принятым для улучшения качества и тестов

Однако, непрерывная интеграция не лишена своих рисков и проблем:

- Необходимо внедрять и поддерживать инструменты непрерывной интеграции
- Процесс непрерывной интеграции должен быть определен и создан
- Автоматизация тестирования требует дополнительных ресурсов и может быть сложной для создания
- Полное тестовое покрытие имеет важное значение для достижения преимуществ автоматизированного тестирования
- Иногда команды излишни полагаются на модульные тесты и выполняют слишком мало системного и приемочного тестирования

Непрерывная интеграция требует использования инструментов, включая инструменты тестирования, инструменты автоматизации процесса сборки и инструменты контроля версий.

1.2.5. Планирование релиза и итерации

Как упоминалось в программе Базового Уровня [ISTQB_FL_SYL], планирование – это постоянный вид деятельности, и это также относится и к жизненным циклам гибких методологий. Для жизненных циклов гибких методологий выполняются два вида планирования – планирование релиза и планирование итерации.

Планирование релиза рассматривает перспективы релиза продукта, часто за несколько месяцев до начала проекта. Планирование релиза определяет и переопределяет набор задач продукта и может включать в себя детализацию больших пользовательских историй в коллекцию небольших историй. Планирование релиза обеспечивает основу для подхода к тестированию и планирования тестирования, охватывающего все итерации. Планы релиза – высокоуровневые.

При планировании релиза представители бизнеса совместно с командой устанавливают и расставляют приоритеты для пользовательских историй релиза. (см. Раздел 1.2.2). Основываясь на этих пользовательских историях, определяются риски проекта и качества и выполняется высокоуровневая оценка объема работ (см. Раздел

3.2).
Тестировщики участвуют в планировании релиза и особенно повышают ценность следующих активностей:

- Определение тестируемости пользовательских историй, включая критерии приемки
- Участие в анализе рисков проекта и качества
- Оценка объема работ по тестированию, связанных с пользовательскими историями
- Определение необходимых уровней тестирования
- Планирование тестирования релиза

После завершения планирования релиза начинается планирование первой итерации. Планирование итерации рассматривает одну итерацию до конца и связано с набором задач итерации.

При планировании итерации команда выбирает пользовательские истории из приоритезированного набора задач релиза, прорабатывает пользовательские истории, проводит для пользовательских историй анализ рисков и оценивает объем работы, необходимый для каждой пользовательской истории. Если пользовательская история слишком расплывчата и предпринятые попытки уточнить ее неудачны, команда может отказаться принять ее и использовать следующую пользовательскую историю, основываясь на приоритете. Представители бизнеса должны ответить на вопросы команды о каждой истории, чтобы команда могла понять, что они должны реализовать и как тестировать каждую историю.
Количество выбранных историй основано на установленной скорости команды и предполагаемом размере выбранных пользовательских историй. После того, как содержимое итерации согласовано, пользовательские истории разбиваются на задачи, которые будут выполняться соответствующими членами команды.
Тестировщики участвуют в планировании итерации и особенно повышают ценность следующих активностей:

- Участие в детальном анализе рисков пользовательских историй
- Определение тестируемости пользовательских историй
- Создание приемочных тестов для пользовательских историй
- Разбивка пользовательских историй на задачи (в частности, задачи по тестированию)
- Оценка объема работа по тестированию для всех задач тестирования
- Определение функциональных и нефункциональных аспектов тестируемой системы
- Поддержка и участие в автоматизации тестирования на нескольких уровнях тестирования

Планы релиза могут меняться в течении проекта, включая изменения пользовательских историй в наборе задач продукта. Эти изменения могут быть вызваны внутренними или внешними факторами. Внутренние факторы включают возможности поставки, быстроту поставки и технические проблемы. Внешние факторы включают

открытие новых рынков и возможностей, новых конкурентов или бизнес-угроз, которые могут изменить цели релиза и / или целевые даты. Кроме того, планы итераций могут меняться во время итерации. Например, конкретная пользовательская история, которая считалась относительно простой во время оценки, может оказаться более сложной, чем ожидалось.

Эти изменения могут быть сложными для тестировщиков. Тестировщики должны понимать общую картину релиза для планирования целей тестирования, и они должны иметь достаточный базис тестирования и тестовый предсказатель для каждой итерации для разработки целей тестирования, как описано в программе обучения Базового Уровня [ISTQB_FL_SYL, раздел 1.4]. Необходимая информация должна быть доступна тестировщику раньше, но все же изменение должно быть принято в соответствии с принципами гибких методологий. Эта дилемма требует тщательного принятия решений относительно стратегии тестирования и документации по тестированию. Подробнее о задачах тестирования в гибких методологиях см. [Black09, глава 12].

Планирование релиза и итераций должно касаться планирования тестирования, так же, как и планирования активностей в области разработки. К конкретным вопросам, связанным с тестированием, относятся:

- Объем тестирования, глубина тестирования областей из этого объема, цели тестирования и обоснования этих решений.
- Члены команд, которые будут проводить работы по тестированию.
- Необходимые тестовое окружение и тестовые данные, когда они нужны, и произойдут ли какие-либо изменения в тестовом окружении и / или в данных до или во время проекта
- Сроки, последовательность, зависимости и предварительные условия для функционального и нефункционального тестирования (например, как часто запускать регрессионные тесты, какие функции зависят от других функций или тестовых данных и т.д.), в том числе о том, как работы по тестированию связаны и зависят от работ по разработке.
- Риски проекта и качества, подлежащие рассмотрению (см. Раздел 3.2.1).

Дополнительно, большие команды оценивая объем работ должны включать учет необходимого времени и объема работ для завершение требуемого тестирования.

2. Ключевые принципы, методики и процессы в гибком тестировании – 105 минут

Ключевые слова
Верификационный тестовый сценарий, элемент конфигурации, управление конфигурацией

Цели обучения фундаментальным принципам, практикам и процессам гибкого тестирования

2.1. Различия процессов тестирования при традиционных и гибких подходах

FA-2.1.1 (K2) Описать различия тестовых активностей в проектах под управлением гибких методологий и в проектах не под их управлением
FA-2.1.2 (K2) Описать взаимосвязь активностей по разработке и тестированию в проектах под управлением гибких методологий
FA-2.1.3 (K2) Описать роль независимого тестирования в проектах под управлением гибких методологий

2.2. Статус тестирования в проектах под управлением гибких методологий

FA-2.2.1 (K2) Описать инструменты и техники, используемые обмена для информацией по статусу тестирования, включая прогресс и качество продукта
FA-2.2.2 (K2) Описать процесс развития тестов в течение множества итераций и объяснить почему в проектах под управлением гибких методологий для управления регрессионными рисками важна автоматизация тестирования

2.3. Роль и навыки тестировщика в команде, работающей по гибким методологиям

FA-2.3.1 (K1) Понять, какие навыки (в сфере общения с людьми, в профессиональной сфере и в тестировании) необходимы тестировщику в гибкой команде
FA-2.3.2 (K3) Понять роль тестировщика в гибкой команде

2.1. Различия процессов тестирования при традиционных и гибких подходах

Как описано в учебном плане базового уровня [ISTQB_FL_SYL] и в [Black09], тестовые активности связаны с активностями по разработке, и, как следствие, процесс тестирования строится по-разному в разных жизненных циклах. Тестировщики должны понимать различия между тестированием в традиционных моделях (например, в последовательных, таких как V-модель или итеративных, таких как RUP) и тестированием при гибком подходе, чтобы работать эффективно и продуктивно. Гибкие модели отличаются тем, как интегрируются процессы тестирования и разработки, тем как выстраивается работа над проектом, наименованиями, критериями входа и выхода, использующихся для различных уровней тестирования, использованием инструментов и тем, как можно эффективно использовать независимое тестирование.
Тестировщики должны помнить, что организации значительно различаются реализацией жизненных циклов ПО. Отклонение от идеалов гибких подходов (см. Раздел 1.1) может представлять собой разумную настройку и адаптацию разных практик. Способность адаптироваться к особенностям конкретного проекта, включая фактически применяемые методы разработки программного обеспечения, является ключевым фактором успеха для тестировщиков.

2.1.1. Активности тестирования и разработки

Одним из ключевых отличий между традиционными и гибкими подходами в разработке программного обеспечения является концепция сравнительно коротких итераций, каждая из которых приводит к созданию рабочего программного обеспечения, которое предоставляет функции, имеющие ценность для заинтересованных сторон бизнеса. В начале проекта осуществляется планирование выпуска релиза. За этим следует последовательность итераций. В начале каждой итерации осуществляется ее планирование. После определения перечня задач на итерацию осуществляется разработка выбранных пользовательский историй, интеграция их в общий продукт, а затем тестирование. Из-за высокой динамики протекания итерации процессы разработки, внедрения и тестирования происходят как параллельно, так и одновременно. Процесс тестирования происходит на протяжении всей итерации, а не в конце.
Тестировщики, разработчики и представители бизнеса - все они играют определенную роль в тестировании, как и в традиционных жизненных циклах. Разработчики выполняют модульные тесты, поскольку они разрабатывают функции из пользовательских историй. Тестировщики затем проверяют этот функционал. Представители бизнеса также проверяют истории во время реализации. Все заинтересованные стороны могут использовать как написанные тестовые сценарии, так и просто экспериментировать с использованием функционала, чтобы обеспечить быструю обратную связь с командой разработчиков.

В некоторых случаях могут возникать укрепляющие или стабилизирующие итерации для исправления давно найденных дефектов и других форм так называемого технического долга. Однако наилучшей практикой является то, что никакой функционал не считается выполненным до тех пор, пока он не будет интегрирован и системно протестирован [Goucher09]. Другая хорошая практика заключается в исправлении дефектов, оставшихся от предыдущей итерации, в начале следующей итерации, называемая «Сначала исправить ошибки». Иногда возникают жалобы, что эта практика приводит к ситуации, когда суммарная работа, которая должна быть выполнена в процессе итерации, неизвестна, и становится сложнее оценить, когда может быть реализован остальной функционал. В конце последовательности из нескольких итераций может проводиться ряд действий по подготовке релиза, однако в некоторых случаях релиз выпускается в конце каждой итерации.

Вследствие того, что тестирование на основе риска используется в качестве одной из стратегий тестирования, во время планирования релиза проводится анализ рисков высокого уровня, при этом тестировщики часто используют этот анализ. Однако конкретные риски качества, связанные с каждой итерацией, определяются и оцениваются при планировании итераций. Этот анализ рисков может влиять на последовательность разработки, а также на приоритет и глубину тестирования функционала. Это также влияет на оценку усилий, необходимых для тестирования каждой функции (см. Раздел 3.2).

В некоторых гибких методах (например, в экстремальной разработке) используется парная работа. Под ней подразумевается объединение тестировщиков, работающих вместе на проекте, в пары для проверки функционала. Парная работа может также объединять тестировщика с разработчиком для разработки и тестирования функционала. Парная работа может быть затруднена при распределенной работе тестовой группы, но существующие процессы и инструменты могут помочь обеспечить ее выполнение. Дополнительные сведения о распределенной работе см. в разделе [ISTQB_ALTM_SYL, раздел 2.8].

Тестировщики могут также выступать в качестве тренеров по тестированию и повышению качества в команде, делиться опытом тестирования и поддерживать работу по обеспечению качества. Это способствует осознанию коллективной ответственности за качество продукта.

Автоматизация тестирования на всех уровнях тестирования происходит во многих командах, работающих с применением гибких методологий, и это означает, что тестировщики тратят время на создание, выполнение, мониторинг и поддержку автоматизированных тестов и результатов. Из-за интенсивного использования автоматизации тестирования, большое количество ручного тестирования на проектах, как правило, осуществляется с использованием методов, основанных на опыте и основанных на найденных дефектах, таких как программные атаки, исследовательское тестирование и предположение об ошибках [ISTQB_ALTA_SYL, разделы 3.3 и 3.4] и [ISTQB_FL_SYL, раздел 4.5]).

В то время как разработчики будут сосредоточены на создании модульных тестов,

тестировщикам следует сосредоточиться на создании автоматизированных систем интеграции, системных и системных интеграционных тестов. Это приводит к тому, что команды, работающие по гибким методологиям, предпочитают тестировщиков с большим техническим опытом и опытом автоматизации тестирования.

Один из основных принципов гибких методологий заключается в том, что изменения могут возникать на протяжении всего проекта. Поэтому в проектах, выстроенных с применением таких методологий, работа с документацией проекта ведется в облегченном режиме. Изменения существующих функций влияют на тестирование, особенно на регрессионное тестирование. Использование автоматизированного тестирования является одним из способов управления трудозатрат по тестированию, связанных с изменением функционала. Однако важно, чтобы скорость изменения не превышала способность проектной команды справляться с рисками, связанными с этими изменениями.

2.1.2. Результаты проектной работы

Результаты работы проектной команды и тестировщиков в частности в гибких подходах обычно делят на три категории:

1. Бизнес-ориентированные результаты, описывающие предназначение функционала (например, спецификации требований) и способы его использования (например, пользовательская документация)
2. Результаты разработки, которые описывают, как система построена (например, диаграммы сущности объекта), как фактически реализована (например, код) или как устроены отдельные фрагменты кода (например, автоматические модульные тесты)
3. Результаты тестирования, которые описывают, как строился процесс тестирования системы (например, стратегии и планы тестирования), как фактически тестировалась система (например, ручные и автоматические тесты) и результаты тестирования.

В типичном проекте, выстроенном на основе гибких методологий, обычной практикой является избегание большого количества документации. Вместо этого основное внимание уделяется созданию рабочего программного обеспечения вместе с автоматическими тестами, которые демонстрируют соответствие требованиям. Это поощрение к сокращению документации относится только к документации, которая не представляет ценности для заказчика. В успешном проекте наблюдается баланс между повышением эффективности за счет сокращения документации и предоставления достаточной документации для поддержки бизнеса, тестирования, разработки и сопровождению. На этапе планирования релиза, проектная команда должна принять решение о необходимых результатах своей работы к завершению проекта и определить, какой уровень проектной документации необходим.

Типичные бизнес-ориентированные рабочие продукты на таких проектах включают пользовательские истории и критерии приемки при работе над проектом. Пользовательские истории - это гибкая форма спецификаций требований, целью которых является объяснение, как система должна вести себя в отношении единой, когерентной функции или функционала. Пользовательская история должна определять достаточно маленький функционал, чтобы его можно было реализовать за одну итера-

цию. Большие коллекции связанных функций или набор вспомогательных функций, составляющих один сложный функционал, принято называть «Бизнес-потребностями». Бизнес-потребности могут включать пользовательские истории из разных групп разработчиков. Например, одна пользовательская история может описывать, что требуется на уровне API (промежуточное ПО), в то время как другая история описывает, что необходимо на уровне интерфейса (приложение). Эти сочетания могут быть разработаны в рамках серии итераций. Каждая бизнес-потребность и ее пользовательские истории должны соответствовать критериям приемки.

Типичные продукты для разработчиков на проектах, построенных с применением гибких методологий, включают код. Эти же разработчики часто создают автоматизированные модульные тесты. Такие тесты могут быть созданы после разработки кода. Однако в некоторых случаях разработчики создают тесты постепенно, до того, как весь код будет написан, чтобы обеспечить проверку, как только часть кода будет написана, работает ли она так, как ожидалось.

Хотя этот подход и называется «сначала тестируем» или «разработка, управляемая тестированием», в действительности, тесты являются скорее формой исполняемых низкоуровневых проектных спецификаций, чем тестов [Beck02].

Типичные результаты работы тестировщика на проектах, построенных с применением гибких методологий, включают автоматические тесты, а также такие документы, как планы тестирования, каталоги рисков качества, ручные тесты, отчеты о дефектах и журналы результатов тестирования. Документы о тестировании должны быть написаны максимально простым языком и иметь максимально простую структуру, что также бывает характерно для традиционных подходов. Тестировщики также собирают метрики на основе результатов проведенной работы по тестированию и протоколов результатов тестов, где делается упор на упрощенный подход к этим действиям.

В некоторых реализациях гибких методологий, особенно в регламентированных, критически важных, распределенных или очень сложных проектах и продуктах, необходима дальнейшая формализация этих результатов. Например, некоторые команды преобразовывают пользовательские истории и критерии приемки в более формальные спецификации требований. Отчеты по вертикальной и горизонтальной возможности отслеживания могут быть подготовлены для анализа аудиторам, соответствия нормативам и других требований.

2.1.3. Уровни тестирования

Уровни тестирования - это активности тестирования, которые логически связаны друг с другом, зачастую по готовности или полноте проверяемого функционала.

В последовательных моделях жизненного цикла уровни тестирования часто определяются так, что критерии завершения одного уровня являются частью критериев начала для следующего уровня. В некоторых итеративных моделях это правило не применяется. Уровни тестирования перекрываются. Спецификация требований, проектная спецификация и действия по разработке могут перекрываться уровнями тестирования.

В некоторых жизненных циклах разработки с применением гибких методологий перекрытие происходит, потому что изменения в требованиях, дизайне и коде могут происходить в любой момент итерации. Хотя Скрам теоретически не позволяет изменять пользовательские истории после планирования итерации, на практике такие изменения иногда происходят. Во время итерации для любой пользовательской истории обычно выполняются последовательно следующие тестовые действия:

- Модульное тестирование, обычно выполняемое разработчиком
- Приемочное тестирование функционала, которое иногда разбивается на два вида деятельности:
  - Верификационное тестирование компонентов, которое часто автоматизировано, может выполняться разработчиками или тестировщиками и включает тестирование по критериям приемки пользовательской истории
  - Валидационное тестирование функционала, которое обычно является ручным и может выполняться силами разработчиков, тестировщиков и бизнес-партнеров, работающих совместно, чтобы определить, подходит ли эта функция для использования, улучшить наглядность достигнутого прогресса и получить реальную обратную связь от заказчика

Кроме того, часто наблюдается параллельный процесс регрессионного тестирования, происходящий на протяжении всей итерации. Он включает повторное выполнение автоматических модульных тестов и проверок функционала, реализованного в текущей итерации и в предыдущих итерациях, как правило, с помощью системы непрерывной интеграции.

В некоторых проектах может существовать уровень системного тестирования, который начинается, когда первая пользовательская история готова к такому тестированию. Это может включать выполнение функциональных тестов, а также нефункциональные тесты производительности, надежности, удобства использования и других соответствующих типов тестов.

Гибкие команды могут использовать различные формы приемочного тестирования (используя термин, описанный в учебном плане базового уровня [ISTQB_FL_SYL]). Внутренние альфа-тесты и внешние бета-тесты могут проводиться либо по завершении каждой итерации, либо после серии итераций. Приемочные испытания для пользователей, эксплуатационные приемочные испытания, нормативные приемочные испытания и контрактные приемочные испытания также могут проводиться либо по завершении каждой итерации, либо после серии итераций.

2.1.4. Управление тестированием и взаимодействием

Гибкие проекты часто связаны с использованием автоматизированных инструментов разработки, тестирования и управления разработкой программного обеспечения. Разработчики используют инструменты статического анализа, модульного тестирования и покрытия кода. Разработчики постоянно проверяют код и модульные тесты в системе управления конфигурацией, используя автоматизированные схемы сборки и тестирования. Эти интегрированные среды позволяют непрерывно добавлять новое

программное обеспечение в систему, при проверке которого повторяется и статический анализ, и модульные тесты.
Такие автоматизированные тесты могут также включать функциональные тесты на уровне интеграционного и системного тестирования. Эти функциональные автоматические тесты могут быть созданы с использованием функциональных испытательных заглушек, функциональных средств тестирования пользовательского интерфейса с открытым исходным кодом или коммерческих инструментов и могут быть интегрированы с автоматизированными сценариями, которые выполняются как часть непрерывной интеграции. В некоторых случаях из-за продолжительности функциональных тестов они отделяются от модульных и выполняются реже. Например, модульные тесты могут запускаться каждый раз, когда проверяется новое программное обеспечение, а более длинные функциональные тесты запускаются только каждые несколько дней.
Одна из целей автоматизированных тестов - подтвердить, что сборка работает и устанавливается. Если какой-либо автоматизированный тест завершается неуспешно, команда должна своевременно устранить обнаруженный дефект и сохранить изменения в коде. Для оперативного доступа к результатам тестирования требуется умение создавать отчеты о тестировании в режиме реального времени. Этот подход помогает сократить дорогостоящие и неэффективные циклы «разработка-установка-ошибка-исправление-переустановка», которые могут возникать во многих традиционных проектах, поскольку изменения, которые нарушают сборку или являются причиной сбоя программного обеспечения, обнаруживаются быстро.
Автоматизированные инструменты тестирования и сборки помогают управлять риском регрессии, связанным с частыми изменениями, которые часто возникают в проектах, построенных с применением гибких методологий. Однако чрезмерная зависимость от автоматизированного модульного тестирования для управления этими рисками может быть проблемой, поскольку модульное тестирование часто имеет ограниченную эффективность обнаружения дефектов [Jones11]. Также требуются автоматизированные тесты на уровне системного и интеграционного тестирования.

2.1.5. Варианты организации независимого тестирования

Как обсуждалось в программе базового уровня [ISTQB_FL_SYL], независимые тестировщики часто более эффективны при поиске дефектов. В некоторых гибких командах разработчики создают множество автоматизированных тестов. Один или несколько тестировщиков могут быть встроены в команду, выполняя множество задач тестирования. Однако, учитывая положение этих тестировщиков в команде, существует риск потери независимости и объективной оценки.
Другие гибкие команды сохраняют полностью независимые, отдельные тестовые группы и назначают тестировщиков по требованию в течение последних дней каждого спринта. Это может сохранить независимость, и такие тестировщики могут обеспечить объективную, независимую оценку программного обеспечения. Однако, временные рамки, непонимание новых функций продукта и проблемы взаимоотношений

с заинтересованными сторонами и разработчиками, зачастую, приводят к проблемам при таком подходе.

Третий вариант состоит в том, чтобы иметь независимую отдельную группу тестирования, в которой тестировщики назначаются в группы на долгосрочной основе в начале проекта, что позволяет им сохранять свою независимость, получая хорошее представление о продукте и крепкие отношениях с другими членами команды. Кроме того, независимая команда тестирования может иметь специализированных тестировщиков вне таких групп для работы в долгосрочных и / или итерационно-независимых мероприятиях, таких как разработка автоматизированных тестовых инструментов, проведение нефункционального тестирования, создание и поддержка тестовых сред и выполнения разно уровневого тестирования, которые не могут быть хорошо выполнены в рамках спринта (например, системное интеграционное тестирование).

2.2. Статус тестирования в проектах под управлением гибких методологий

Изменения в проектах под управлением гибких методологий происходят очень быстро. Это означает, что статус тестирования, ход тестирования и качество продукта должны постоянно контролироваться. Тестировщики должны искать способы доводить до команды информацию, позволяющую принимать правильные решения на каждой итерации. К тому же изменения в коде могут повлиять на функциональность из предыдущих итераций. Таким образом, ручные и автоматические тесты должны модифицироваться для предотвращения регрессионных рисков.

2.2.1. Обмен информацией по статусу тестирования, прогрессу и качеству продукта

Команды, работающие по гибким методологиям, должны выпускать работоспособное ПО в конце каждой итерации. Чтобы определить, когда у команды будет готово работоспособное ПО, необходимо отслеживать прогресс всех рабочих составляющих для каждой итерации и каждом релизе. Тестировщики в командах используют различные методы учета прогресса и статуса тестирования. Например, написание отчетов по результатам автоматического тестирования, представление последовательности тестовых задач и пользовательских историй на доске задач, изображение диаграммы «сгорания» задач и графиков, отражающих статус команды. Это может быть доступно другим членам команды с использованием базы знаний, электронных писем в стиле информационной доски, а также устно в рамках ежедневных митингов, команды могут использовать различные инструменты для автоматической генерации статусных отчетов, основывающихся на результатах и прогрессе тестирования. Эти ин-

струменты автоматически обновляют wiki доски, посылают письма, а также собирают метрики, которые могут быть использованы для улучшения процесса. Такой метод предоставления статуса тестирования экономит время тестировщиков, позволяя сосредоточиться на разработке и выполнении большего количества тестов.

Команды могут использовать диаграмму сгорания задач для отслеживания прогресса на протяжении всего релиза и в течении каждой итерации. Диаграмма сгорания задач показывает количество оставшихся работ, которые необходимо выполнить за время, выделенное на итерацию.

Для предоставления текущего детального статуса всей команды, включая статус тестирования, можно использовать доску задач. Пользовательские истории, задачи разработчиков, задачи тестировщиков, и другие задачи, созданные во время планирования итерации (см. раздел 1.2.5) отражаются на доске задач, обычно используя цветные карточки для определения типа задачи. Во время итерации ход разработки управляется движением этих задач по доске по колонкам «сделать», «в процессе», «проверить» и «сделано». Команды, работающие по гибким методологиям, могут использовать инструменты для автоматизации поддержки своих карточек с историями и досок задач.

Задачи тестирования на доске задач относятся к критериям приемки, определенным для пользовательских историй. Как только автоматические тесты, ручные тесты или исследовательское тестирование оканчиваются, они передвигаются в колонку «сделано» на доске. Вся команда регулярно следит за статусом работ на доске задач, обычно во время ежедневных митингов, что обеспечивает своевременное перемещение задач по доске. Если какие-либо задачи (включая тестовые) не двигаются или двигаются очень медленно, команда выявляет проблемы, которые могут блокировать выполнение этих задач.

Ежедневные встречи должны посещаться всей командой, включая тестировщиков. На этих встречах каждый сообщает статус своей работы. План отчета для каждого члена команды [согласно Agile Alliance Guide] выглядит так:

- Что было сделано с момента последней встречи
- Что планируется сделать до следующей встречи
- Какие есть проблемы

Так как любые проблемы, блокирующие работу, обсуждаются на ежедневных встречах, то вся команда о них знает и может помочь с их решением.

Для улучшения качества продукта многие команды проводят опросы удовлетворенности заказчиков, с целью получения их отзывов о продукте и выяснения соответ-

ствует ли продукт ожиданиям клиентов. Для работы над качеством продукта могут использоваться дополнительные показатели, аналогичные описанным в традиционных методологиях разработки. Например, соотношение количества прошедших / не прошедших тестов, количество обнаруженных дефектов, результаты приемочного и исследовательского тестирования, плотность дефектов, количество найденных и исправленных дефектов, покрытие требований тестами, покрытие рисков, покрытие кода, объем измененного кода.

Как и в любом жизненном цикле, собранные и предоставленные метрики должны быть актуальны и помогать в принятии решений. Метрики не должны использоваться для поощрения, наказания, или изолирования кого-либо из членов команды.

2.2.2. Управление регрессионными рисками, используя ручное и автоматическое тестирование

В проектах под управлением гибких методологий продукт увеличивается в размерах по окончанию каждой итерации. Таким образом, объем тестирования тоже увеличивается. Помимо тестирования изменений в коде, тестировщики должны проверить, не появилась ли необходимость регрессионного тестирования функциональности, разработанной и проверенной в предыдущих итерациях. Риск появления такого тестирования в гибкой разработке велик из-за экстенсивного изменения кода (добавленных, измененных или удаленных строк кода между двумя версиями). Поскольку повышенная ответственность за изменения является основным принципом гибких методологий, модификации могут затрагивать и завершенную ранее функциональность. Для поддержания скорости крайне важно, чтобы команды вкладывались в автоматизацию тестирования на всех уровнях тестирования как можно раньше. Также важно, чтобы все тестовые артефакты - автоматизированные тесты, ручные тестовые сценарии и др. - поддерживались в актуальном состоянии на каждой итерации. Рекомендуется, чтобы все тестовые артефакты содержались и поддерживались в специальном инструменте управления конфигурациями. Это необходимо для осуществления контроля версий, для обеспечения доступа к артефактам всеми членами команды, для управления изменениями, сохраняя историческую информацию о тестовых артефактах по мере изменения функциональности.

Поскольку выполнение всех тестов не всегда возможно, особенно в ограниченных по времени проектах, тестировщики должны выделять время в каждой итерации для оценки ручных и автоматических тестовых сценариев из предыдущего и текущего циклов. Это позволит выбрать тесты-кандидаты на регрессионное тестирование, а также те тесты, которые больше не нужны. Тесты, написанные на ранних итерациях для проверки конкретной функциональности, имеют меньшую ценность на поздних итерациях, поскольку функциональности изменены или написаны новые, которые влияют на поведение предшествующих.

Во время анализа тестов тестировщики должны оценивать их пригодность для авто-

матизации. Команда должна автоматизировать как можно большее количество тестов из предыдущей и текущей итераций. Это позволяет провести автоматическое регрессионное тестирование меньшими ресурсами, нежели этого требует ручное тестирование. Эти освобожденные ресурсы направляются на более тщательную проверку новой функциональности и особенностей текущей итерации.

Важно чтобы тестировщик мог быстро определить и обновить тестовые сценарии из предыдущей итерации, на которые повлияли изменения, внесенные в текущей версии. Определение, как команда будет проектировать, писать и хранить тесты, происходит на этапе планирования релиза. Хорошей привычкой является проектирование и реализация тестов на ранних стадиях и последовательное их применение. Более короткие сроки тестирования и постоянные итерации на поздних этапах увеличат негативное влияние некачественного проектирования и реализации тестов.

Использование автоматизации тестирования на всех уровнях тестирования позволяет гибкой команде быстро реагировать на любое изменение качества продукта. Хорошо написанные автоматические тесты являются описанием функциональности системы [Crispin08]. Проверяя автоматические тесты и их результаты в системе управления конфигурациями в соответствии с разными версиями продукта, команды могут следить за объемом проверенной функциональности и результатами для различных сборок в каждый момент времени.

Автоматические модульные тесты выполняются перед тем, как код выкладывается в основную ветку системы управления конфигурациями. Это делается для проверки того, что изменения не сломали сборку. Для уменьшения ошибок, которые могут замедлить прогресс всей команды, код не выкладывается до того, как он проверен автоматическими модульными тестами. Результаты автоматических модульных тестов предоставляют информацию о качестве сборки, но не о качестве продукта в целом.

Автоматические приемочные тесты выполняются периодически в рамках непрерывной интеграции сборки всей системы. Эти тесты выполняются над всей системой, по крайней мере, ежедневно, но обычно не проводятся при каждом внесении изменения в код, так как они выполняются дольше, чем модульные тесты и могут сильно замедлить выкладывание кода. Результаты автоматического приемочного тестирования говорят о качестве продукта, учитывая регрессию с момента последней сборки, но не предоставляют информацию о качестве продукта в целом.

Автоматические тесты могут выполняться постоянно над всей системой. Первоначальное подмножество автоматических тестов для покрытия критической функциональности системы должно быть создано сразу после поступления новой сборки на тестирование. Эти тесты называются проверочными тестами. Их результаты говорят о качестве сборки в целом, таким образом, тестировщики не тратят время на тестирование нестабильных сборок.

Автоматические тесты, являющиеся частью регрессионного тестирования, обычно выполняются ежедневно в рамках непрерывной разработки ПО, и каждый раз, когда новая сборка поступает на тестирование. Как только автоматический регрессионный тест завершается неудачей, команда прекращает текущую работу и исследует причину провала теста. Тест может провалиться из-за легитимного изменения функциональности в текущей итерации, в этом случае тестовый сценарий или пользовательская история должны быть изменены, чтобы отразить новый приемочный критерий. Другой вариант - тест может быть удален из цикла и заменен новым тестом, покрывающим изменения. Однако если тест провалился из-за дефекта, то хорошей практикой считается его исправление до начала работы с новой функциональностью.

Помимо автоматизации тестов, следующие тестовые задачи могут быть также автоматизированы:

- Генерирование тестовых данных
- Загрузка данных в систему
- Развёртывание сборки в тестовой среде
- Восстановление тестовой среды (например, файлов базы данных сайта) до исходной
- Сравнение выходных данных

2.3. Роль и навыки тестировщика в команде, работающей по гибким методологиям

В команде, работающей по гибким методологиям, тестировщик обязан тесно сотрудничать с другими членами команды и заинтересованными сторонами. Это накладывает определенные требования к навыкам, которыми должен обладать тестировщик, а также к деятельности, которую он должен осуществлять в рамках команды.

2.3.1. Навыки тестировщика, работающего по гибким методологиям

Тестировщик, работающий по гибким методологиям, должен обладать всеми навыками, описанными в Программе Обучения Базового уровня ISTQB. Помимо этого, он должен обладать знаниями в автоматизации тестирования, разработке на основе приемочного тестирования, тестировании методом белого и черного ящика, тестировании на основе имеющегося опыта.

Так как гибкие методологии сильно зависят от коммуникации и взаимодействия между членами команды и заинтересованными лицами вне команды, тестировщики

должны обладать хорошими коммуникативными навыками. Тестировщик в команде, под управлением гибких методологий, должен:

- Быть позитивным и настроенным на решения задач при общении с членами команды и заинтересованными лицами
- Демонстрировать критическое, нацеленное на качество продукта, скептическое мышление
- Уметь получать информацию от заинтересованных сторон (а не полагаться исключительно на письменные спецификации)
- Точно оценивать и сообщать результаты и статус тестирования, и информацию о качестве продукта
- Эффективно уметь определять тестопригодность пользовательских историй, особенно критерии приемки с представителем клиента и заинтересованными сторонами
- Работать совместно с командой, работать в паре с программистом и другими членами команды
- Быстро реагировать на изменения, в том числе внося изменения в тесты, добавляя или исправляя тестовые сценарии
- Планировать и организовывать свою работу

Непрерывное развитие навыков, в том числе навыков общения, имеет важное значение для всех тестировщиков, особенно тех, кто работает в командах, под управлением гибких методологий.

2.3.2. Роль тестировщика в команде

Роль тестировщика в команде, работающей по гибким методологиям, включает в себя действия, которые организовывают и обеспечивают обратную связь не только касательно статуса и качества продукта, но и качества процесса. В дополнение к деятельности, описанной в другой главе этой программы, эти действия включают в себя:

- Понимание, внедрение и обновление стратегии тестирования
- Измерение тестового покрытия и отчетности по всем текущим покрытиям
- Обеспечение надлежащего использования тестовых инструментов
- Настройка, использование и управление тестовой средой и тестовыми данными
- Отчетность по дефектам и умение работать с командой во время их решения

- Консультирование других членов команды в соответствующих аспектах тестирования
- Обеспечение соответствующего расписания для текущих задач тестирования и планирование итерации
- Активное сотрудничество с разработчиками и заинтересованными сторонами с целью уточнения требований, особенно тестопригодности, непротиворечивости и полноты
- Активное участие в командных ретроспективах; внесение предложений и внедрение усовершенствований

В рамках гибкой команды каждый член ответственен за качество продукта и играет роль в формировании задач тестирования.

Компании, работающие с применением гибких методологий, могут столкнуться с некоторыми организационными рисками:

- Тестировщики так тесно работают с разработчиками, что теряют соответствующее мышление тестировщика
- Тестировщики становятся терпимы или молчат о неэффективных или низкокачественных практиках в команде
- Тестировщику не удается следить за приходящими изменениями при ограниченном времени итерации

Для снижения этих рисков, организации могут рассмотреть варианты для сохранения независимости части команда, отвечающей за тестирование, описанные в разделе 2.1.5.

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.