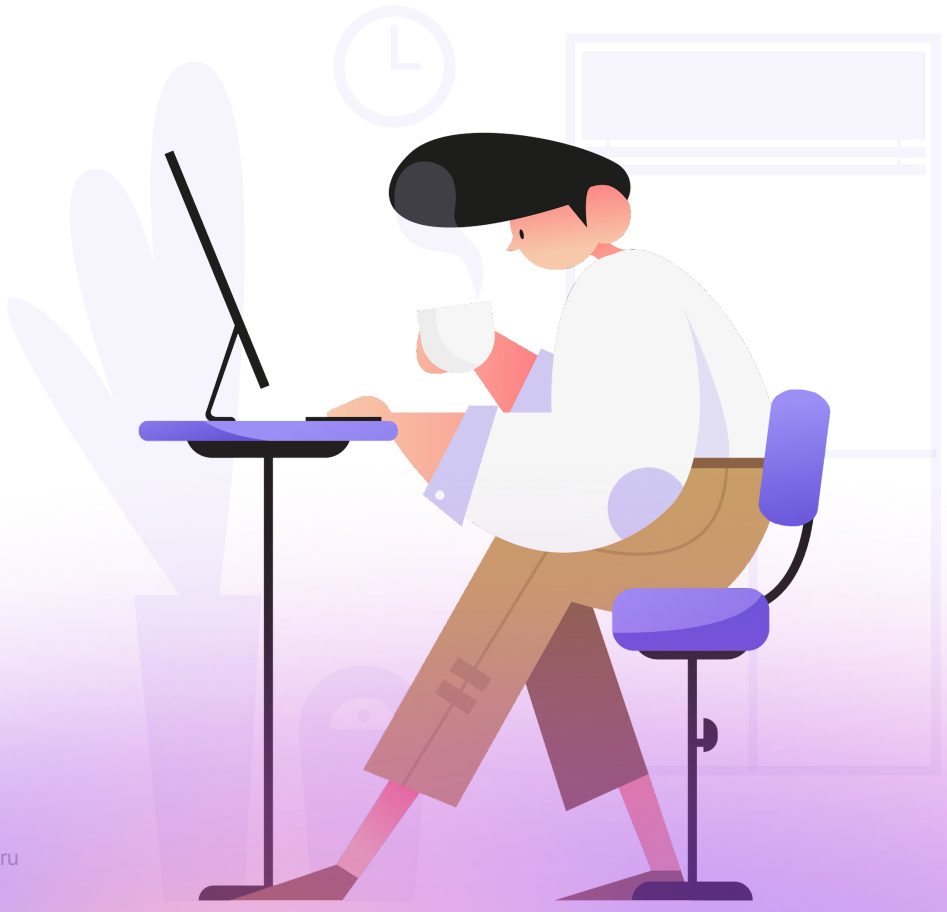
GeekBrains

Основы ручного тестирования

Жизненный цикл программного обеспечения



На этом уроке

1. Рассмотрим основные модели разработки ПО.
2. Познакомимся с гибкими методологиями разработки.
3. Изучим инструменты управления задачами.
4. Поговорим о роли тестирования в процессе разработки ПО.

Оглавление

Жизненный цикл ПО

Жизненный цикл в широком смысле — это ряд этапов, которые проходит живой организм, предмет или система в процессе развития, от момента появления до гибели. То же относится и к жизненному циклу ПО — ряд событий, происходящих с системой в процессе её создания и дальнейшего использования.

Подготовка. Этап от возникновения идеи о создании ПО до описания того его вида и функций. На этом этапе проводится сбор и обработка требований к ПО, планирование этапов работ, сроков, ресурсов и стоимости.

Проектирование. Получение технических заданий, разработка спецификаций. Заказчик получает документальное изложение своих требований и планы проведения работ.

Создание:

- дизайн — получение графических макетов, визуальных форм, разработка интерфейсов, создание индивидуального стиля;

- кодирование — получение исходного кода;
- тестирование — проверка программы на соответствие всем предъявляемым к ней требованиям;
- документирование (опционально) — передача накопленных знаний другим участникам команды разработки.

Поддержка:

- внедрение — установка ПО, обучение пользователей при необходимости;
- сопровождение — исправление выявленных ошибок, поддержка пользователей.

Иногда отдельно в жизненном цикле выделяется такой этап, как прекращение использования ПО.

Модели жизненного цикла ПО

Модель жизненного цикла ПО — описание, как проходит процесс разработки. В модели жизненного цикла ПО описывается, какие этапы проходит ПО, от рождения идеи до завершения использования, и что происходит с ПО на этих этапах. Модель — это, прежде всего, этапы разработки и их последовательность.

Каскадная модель, или «водопад»

Основная особенность — этапы жизненного цикла выполняются последовательно, и есть определённые условия для перехода с одного этапа на другой. Невозможно начать следующий этап, если предыдущий не завершился. После перехода на следующий этап либо невозможно вернуться, либо такой возврат очень сложен. Тестирование в этой модели начинается поздно.

Водопадная модель зарекомендовала себя при создании ПО, для которого в самом начале разработки точно и полно формулируются требования.

Преимущества

- стабильные требования перед началом разработки;
- фиксированный порядок этапов, что позволяет провести качественное планирование разработки;
- у каждой стадии есть чёткий проверяемый результат;
- в каждый момент времени команда выполняет один вид работы;
- простота в изучении;
- легко контролировать;

- обычно используется технически слабо подготовленными командами или неопытным персоналом;
- раннее прогнозирование стоимости проекта.

Недостатки

Недостаток каскадной модели в современной разработке — позднее получение обратной связи от тестировщиков на этапе проверки и от пользователей на этапе эксплуатации. Это существенно повышает стоимость ошибки в процессе разработки.

- позднее получение результата;
- иногда документация избыточна;
- позднее обнаружение конструктивных ошибок;
- позднее представление результатов заказчику;
- сложность возврата к предыдущим этапам;
- отсутствие обратной связи от пользователей;
- сложность изменений и сильное влияние этих изменений на сроки и стоимость проекта;
- неравномерная загрузка участников проекта.

Использование

- крупные проекты со стабильными требованиями — космическая отрасль, медицинское ПО;
- недорогие, несложные, средние проекты;
- позитивный опыт разработки аналогичных систем;
- создание новой версии ранее разработанного продукта, когда вносимые изменения определены и управляемы;
- перенос уже существующего продукта на новую платформу.

V-модель

Усовершенствованная водопадная модель. Тестирование появляется в начале, и каждому этапу модели соответствует свой этап тестирования.

На этапе составления бизнес-требований формируется набор тестов для приёмочного тестирования.

1. Этапу написания функциональных требований соответствует этап написания функциональных тестов.
2. Этапу написания архитектуры системы соответствует этап написания интеграционных тестов.
3. Этапу написания архитектуры компонентов соответствует этап написания модульных тестов.

Преимущества

- простота в использовании;
- планирование и проектирование тестирования на ранних этапах разработки;
- верификация и аттестация всех внешних и внутренних полученных данных, а не только конечного продукта.

Недостатки

- сложно вносить изменения;
- нет действий, направленных на анализ рисков;
- другие недостатки «водопада».

Использование

- управление разработкой ПО в немецкой администрации;
- стандарт для немецких правительственных и оборонных проектов;
- системы, в которых требуется высокая надёжность, например, прикладные программы для наблюдения за пациентами в клиниках;
- встроенное ПО для устройств управления аварийными подушками безопасности в автомобилях.

Инкрементная модель

Основная особенность модели — требования заранее известны.

Разработка идёт «по частям» — инкрементам.

Инкремент — это постоянно увеличивающаяся величина. То есть продукт в процессе разработки постоянно увеличивается, «наращивается».

Первое, что создаётся в этой модели, — минимально работающий продукт, который можно быстро выпустить на рынок, получить обратную связь от пользователей, а затем продолжить его развивать, постепенно расширяя и улучшая инструментарий приложения.

Продукт в инкрементной модели проходит всё те же стадии (этапы) разработки ПО, что и в предыдущих моделях. При этом они многократно повторяются для каждого нового инкремента (мини-водопады).

Преимущества

- позволяет уменьшить затраты на первоначальных этапах разработки — до достижения уровня исходной производительности;
- ускоряет процесс создания функционирующей системы;
- позволяет получать отзывы заказчика в процессе разработки;
- снижает риск неудачи при изменении требований.

Недостатки

- не всегда определяется полная функциональность ПО в начале разработки;
- поскольку модули разрабатываются в разное время, требуется чётко определить интерфейсы;
- сложность приоритизации инкрементов;
- возникает тенденция к откладыванию сложных решений на будущее.

Использование

Проекты, в которых требования сформулированы заранее, и требуется быстрая поставка на рынок.

Итеративная модель

Разработка идёт циклами — итерациями. Каждая итерация представляет собой мини-водопад. Результат каждой итерации — законченная часть инструментария.

Итерация — временной интервал:

- в течение которого проходят все фазы разработки, например, анализ, проектирование, разработка, тестирование;
- который многократно повторяется.

На начальном этапе в этой модели требования могут быть весьма условными или вовсе отсутствовать. В процессе разработки требования создаются, уточняются, изменяются в зависимости от потребностей заказчика, ситуации на рынке.

Преимущества

- быстрый выпуск минимального продукта;
- не требуется полной спецификации требований заранее;
- снижение рисков на ранних стадиях;
- эффективная обратная связь с заказчиком;
- фокус на наиболее важных направлениях проекта;
- непрерывное тестирование, которое позволяет оценить успешность всего проекта;
- раннее обнаружение ошибок в требованиях;
- равномерная загрузка участников проекта.

Недостатки

- могут возникнуть проблемы с архитектурой;
- отсутствие фиксированного бюджета и сроков;
- отсутствие конечной цели, может привести к затягиванию проекта.

Использование

- большие проекты с конкретными требованиями;
- проекты по разработке ПО, которое носит инновационный характер и основано на бизнес-гипотезах, требующих проверки.

Итеративно-инкрементная модель

Чаще всего в современной разработке применяется сочетание двух моделей — инкрементной и итеративной. Это позволяет сделать процесс разработки гибким и менее рискованным.

Если рассматривать её как одну модель, то налицо двойственность:

- с точки зрения жизненного цикла, модель представляется итерационной, так как подразумевает многократное повторение одних и тех же стадий;
- с точки зрения развития продукта, а именно, приращения его полезных функций, модель считается инкрементальной.

Ключевая особенность этой модели — разбиение проекта на небольшие промежутки (итерации), каждый из которых в общем случае может включать в себя все классические стадии, присущие

водопадной и V-образной моделям. Итог итерации — приращение (инкремент) функциональности продукта, выраженное в новой версии.

Спиральная модель

Спиральная модель представляет собой частный случай итерационно-инкрементальной модели, где делается акцент на управлении рисками, в особенности влияющими на организацию процесса разработки проекта и контрольные точки.

Спиральная модель делится на четыре ключевые фазы:

1. Проработка целей, альтернатив и ограничений.
2. Анализ рисков и прототипирование.
3. Разработка продукта (промежуточной версии).
4. Планирование следующего цикла.

Главная задача — как можно быстрее показать пользователям системы работоспособный продукт, активизируя тем самым процесс уточнения и дополнения требований. То есть создать так называемый минимально жизнеспособный продукт (MVP — minimum viable product).

С точки зрения тестирования и управления качеством, повышенное внимание рискам — ощутимое преимущество при использовании спиральной модели для разработки концептуальных проектов, в которых требования естественным образом будут сложными и нестабильными. То есть требования могут многократно меняться по ходу выполнения проекта.

Эта модель не подойдёт для малых проектов, но уместна для сложных и дорогих. Например, разработка системы документооборота для банка, когда каждый следующий шаг требует большего анализа для оценки последствий, чем программирование само по себе. Или проекты с высоким риском, исследовательские проекты, либо проекты, в которых изначально неизвестно, возможна ли их реализация в текущих условиях.

Преимущества

- высокий уровень анализа рисков;
- активное участие заказчиков, особенно на стадии планирования;
- усовершенствование процесса на каждом витке.

Недостатки

- высокая стоимость разработки;
- сложно определить момент перехода на следующий виток спирали;

- сложная структура, трудная для понимания;
- спираль может продолжаться бесконечно.

Использование

- проекты с высоким риском;
- исследовательские проекты;
- проекты, в которых изначально неизвестно, возможна ли их реализация в текущих условиях.

Методология разработки

Методология разработки — это набор методов по управлению разработкой ПО, подборка практических правил и техник разработки. Или по-другому, методология — это детализированный набор правил, практик и принципов, применяемый при реализации той или иной модели.

Фреймворк процессов — это методология, содержащая большое число правил. Необязательно использовать их все. Можно выбрать только те, что нужны сейчас, и построить на их основе процесс разработки.

Гибкие методологии разработки

Из-за разной информации и недопонимания часто встречается понятие Agile среди моделей или методологий разработки. Но это ошибочно: Agile — не модель и не методология. Это философия разработки, на которую опираются гибкие методологии. Agile объединяет в себе разные гибкие методологии разработки, такие как:

- SCRUM;
- Kanban;
- Lean;
- XP — экстремальное программирование.

Гибкие методологии разработки основываются на итеративно-инкрементной модели.

Гибкие методологии разработки — обобщающий термин для нескольких подходов и практик, основанных на ценностях манифеста гибкой разработки программного обеспечения и 12 принципах, лежащих в его основе.

Agile-манифест — главные идеи, на основании которых выстраиваются процессы в гибких методологиях разработки:

- люди и их взаимодействие важнее процессов и инструментов;

- готовый продукт важнее документации по нему;
- сотрудничество с заказчиком важнее жёстких контрактных ограничений;
- реакция на изменения важнее следования плану.

Agile не отрицает важности того, что процессы, инструменты, документация и прочее также важны для разработки качественного ПО, но приоритет всё же отдаётся идеям Agile-манифеста.

Кроме манифеста, есть некоторые принципы, лежащие в основе гибкой разработки. Основные из них:

1. Наивысший приоритет — удовлетворение потребностей заказчика благодаря регулярной и ранней поставке нового инструментария. Чем раньше заказчик увидит новую функциональность, тем быстрее команда проекта получит обратную связь и скорректирует свою работу.
2. Изменение требований приветствуется даже на поздних стадиях разработки. Agile-процессы позволяют использовать изменения для обеспечения заказчику конкурентного преимущества.
3. Работающий продукт следует выпускать как можно чаще с периодичностью от двух недель до нескольких месяцев.

На протяжении всего проекта разработчики и представители бизнеса должны ежедневно работать вместе. Цель — получать обратную связь, корректировать действия команды и разрешать возникшие трудности и недопонимания в процессе разработки.

Над проектом должны работать мотивированные профессионалы. Этот пункт важен для гибких методологий, так как процессы в них менее формализованы, нет чётких критериев оценки работы, отчётов о рабочем времени.

Прямое общение — наиболее важный и эффективный способ обмена информацией:

- участников команды;
- команды;
- заказчика.

Работающий продукт — основной показатель прогресса.

Инвесторы, разработчики и другие члены команды должны иметь возможность поддерживать постоянный ритм работы. Работа над проектом обычно занимает много времени, от нескольких месяцев до нескольких лет. Поэтому требуется создавать условия работы и организовывать процессы таким образом:

- чтобы команда работала без сбоев, авралов;

- чтобы не происходило профессионального выгорания членов команды и всем было комфортно работать.

Постоянное внимание к техническому совершенству и качеству проектирования повышает гибкость проекта — для долгосрочной работы нужно, чтобы приложение было качественно и технически грамотно разработано.

Простота — искусство минимизации лишней работы — очень важна для проектов с гибкой разработкой. Не нужно выполнять ту работу, которую можно не делать, например, составлять детальный план тестирования.

Самые лучшие требования, архитектурные и технические решения рождаются у самоорганизующихся команд.

Задача команды — систематически анализировать возможные способы улучшения эффективности и корректировать процессы и стиль работы.

Особенности Agile-команд

- при планировании работ учитываются интересы всех членов команды;
- работа над завершением задач более приоритетного уровня идёт в первую очередь;
- признание проблем;
- члены команды помогают друг другу;
- тестировщик — полноценный член команды.

Agile — подходы к созданию ПО путём непрерывной, быстрой поставки ценного рабочего инструментария от самоорганизующейся команды специалистов в сотрудничестве с заказчиком.

Преимущества

- быстрая установка рабочего инструментария;
- взаимодействие с заказчиком;
- возможность получения быстрой обратной связи от конечных пользователей;
- адаптируемость к изменению внешних условий;
- в процессе разработки есть возможность корректировать требования.

Недостатки

- сложно применять в непростых проектах;

- опасность возникновения неучтённых рисков;
- может быть недостаточно документации.

SCRUM

Самой используемой и гибкой методологией разработки в 2018 году был SCRUM, а также «гибридные» методологии на его основе.

SCRUM — набор принципов, ценностей, политик, ритуалов, артефактов, на которых строится процесс SCRUM-разработки. Процесс его разработки позволяет в жёстко фиксированные и небольшие по времени итерации, называемые спринтами, предоставлять конечному пользователю работающий продукт с новыми бизнес-возможностями, для которых определён наибольший приоритет.

Основные роли в SCRUM

1. Скрам-мастер (SCRUM Master) следит, чтобы команда работала по SCRUM, и помогает решать возникающие проблемы. Скрам-мастером может выступать любой член команды, который хорошо понимает, как организуются процессы в SCRUM, и способен ими управлять.
2. Владелец продукта (Product Owner) — заказчик или человек, который определяет основные цели в реализации проекта. Эти задачи формируют бэклог продукта.
3. Команда разработки (Development Team).

Из чего состоит SCRUM

- итерации или спринты;
- планирование итерации (sprint planning meeting);
- ежедневный стендап (standup);
- демо (demo);
- ретроспектива (retro);
- доска задач (SCRUM Board).

Итерация или спринт в SCRUM — это промежуток времени от недели до месяца, в течение которого происходит разработка работоспособной части продукта или новой функциональности. В конце каждой итерации команда предоставляет заказчику либо другому заинтересованному лицу готовый инкремент продукта. Принятая на проекте длина итерации должна быть постоянной, чаще всего она составляет две недели.

Перед стартом каждой новой итерации происходит планирование, то есть определение набора задач, который планируется реализовать в процессе итерации. Обычно это называется бэклогом спринта.

В итерацию (спринт) берётся часть функций, которые требуется реализовать в этом спринте. Эти функции уже формируют бэклог спринта.

Чтобы сформировать бэклог спринта, в начале каждой итерации проводится встреча всех членов команды, а также скрам-мастера, владельца продукта и других заинтересованных лиц. Эта встреча называется планированием спринта. На ней обсуждаются все задачи и цели команды и определяется, какой объём работы может быть выполнен командой в процессе предстоящего спринта.

Все члены команды должны быть согласны с объёмом работы, который может предварительно оцениваться в человеко-часах, стори-поинтах или любых других попугаях, предусмотренных на проекте.

Есть разные варианты оценки задач, которые формируют бэклог спринта, подробнее о них — в руководстве по SCRUM: ссылка на него — в дополнительных материалах.

Все задачи, запланированные для спринта, требуется приоритизировать и выполнить в том спринте, в бэклог которого они включаются.

Во время прохождения спринта важным организационным мероприятием становится ежедневный скрам-митинг. Это встреча всей команды для обсуждения текущей ситуации на проекте. Её основная цель — оценка прогресса команды и оперативное выявление возникающих проблем. Она длится 15 минут.

После спринта проводится демонстрация полученных результатов — демо. На демонстрацию приглашаются все заинтересованные лица и владелец продукта. На встрече команда показывает владельцу продукта результат и получает рекомендации для дальнейшей работы. На основании проведённого демо у владельца продукта появляется возможность скорректировать весь бэклог продукта.

Отдельно для команды проекта в SCRUM предусматривается проведение ретроспективы по итогам спринта и демо. На встрече подводятся итоги спринта, обсуждаются возникшие проблемы или успехи.

Для визуализации задач и контроля прогресса в SCRUM используется доска, на которой размещаются все задачи, взятые в бэклог спринта.

Kanban

Kanban — методология гибкой разработки, в основе которой лежит быстрота поставки готового продукта на рынок, а также равномерное распределение нагрузки между членами команды. Процесс разработки прозрачен для всех участников.

Основное преимущество разработки по методологии Kanban — скорость поставки законченной функциональности пользователю. Здесь нет спринтов.

Например, заказчик хочет изменить процедуру оплаты товара, приобретённого пользователем в интернет-магазине. Он сообщает об этом аналитику команды, аналитик создаёт задачу для разработчика и назначает её наивысший приоритет. Разработчик реализует новую функциональность и сразу передаёт её на тестирование.

Если во время тестирования не выявится критических или средних дефектов, то новая функциональность сразу же установится на продуктовые серверы, и у пользователей появится возможность с ней работать.

Основной инструмент в Kanban — доска. Её главная особенность — ограничение задач, которые могут быть одновременно в разработке или тестировании. Поэтому важно следить, чтобы в процессе работы не возникало «бутылочных горлышек», и большое число задач не скапливалось на одном этапе разработки.

Жизненный цикл тестирования ПО

Жизненный цикл тестирования связан с моделью разработки. Если разработка ведётся по каскадной модели, тестирование проводится один раз на соответствующем этапе, и все фазы его жизненного цикла повторяются только один раз.

В итерационно-инкрементной модели этапы циклично повторяются на протяжении проекта. В этом случае длина жизненного цикла тестирования обычно зависит от длины итерации либо от фазы разработки, на которой проект находится в конкретный момент.

Например, на начальном этапе разработки больше внимания уделяется тестированию требований, ближе к окончанию проекта — составлению и анализу отчётов по результатам тестирования, а также приёмочным тестам.

Этапы жизненного цикла тестирования

Общее планирование и анализ требований. На этом этапе требуется определить:

- что конкретно предстоит тестировать: продукт, приложение, предметную область, набор технологий;
- ожидаемый объём работы;
- возможные риски.

Чтобы оценить перечисленное выше, надо провести анализ требований (если они есть) либо планирование другим возможным способом.

Уточнение критериев приёмки. Перед началом тестирования важно определиться с основными критериями оценок (метрик). Надо точно указать:

- метрики и признаки, руководствуясь которыми можно начинать тестирование;
- метрики, на основании которых можно приостановить тестирование, например, количество дефектов к объёму пройденных тестов, и затем его возобновить;
- критерии завершения тестирования.

Подробнее о метриках поговорим в следующих лекциях.

Уточнение стратегии тестирования. В зависимости от стадии или этапа проекта уточняются стратегии тестирования, чтобы они были актуальны для текущей итерации.

Разработка тест-кейсов. Разработка, написание, корректировка и прочие действия с тест-кейсами, их наборами и иными тестовыми артефактами, которые будут использоваться в процессе тестирования.

Выполнение тест-кейсов. Этап прямого тестирования. Тесно связан со следующим этапом.

Фиксация найденных дефектов. Дефекты, обнаруженные в процессе прохождения тест-кейсов либо иным образом, требуют фиксации в системе хранения дефектов (баг-трекере).

Анализ результатов тестирования. При проведении анализа результатов важно опираться на метрики и критерии, сформированные на этапах 1, 2 и 3, а также на данные тест-планов. Полученные выводы оформляются на следующей стадии.

Отчётность. Подведение итогов проведённого тестирования. В итерационно-инкрементной модели выводы, сформулированные на этом этапе, служат основой для стадий 1, 2 и 3 в следующей итерации тестирования.

Подходы к тестированию

Тестирование на основе моделей

Тестовые модели — абстрактные наглядные схемы, описывающие состояние, взаимодействия и связи системных компонентов. Это схемы, таблицы, диаграммы переходов состояний и интеллект-карты. Такие модели создаются на этапе проектирования системы и демонстрируют влияние одной части ПО на другую.

Тестовые модели собираются до старта разработки, их можно обновлять и переиспользовать при изменении системы. Таким образом, тестовая модель даёт представление о системе всем участникам разработки и упрощает поддержку тестовой документации.

При тестировании на основе модели тестировщик создаёт тест-кейсы и чек-листы, опираясь на информацию об устройстве продукта. На этапе исполнения проверяется соответствие реализации и спецификации системы.

Тестирование на основе данных

Тестирование на основе данных (Data Driven testing, DDT) — это инфраструктура автоматизации тестирования, которая хранит тестовые данные в виде таблицы. Входные значения считываются из файлов данных и сохраняются в переменной тестовых сценариев. DDT объединяет положительные и отрицательные сценарии в один тест. Входные данные хранятся в файлах Excel, XML, CSV, базах данных.

Тестирование на основе риска

Тестирование на основе рисков (Risk Based Testing, RBT) — это тип тестирования, основанный на вероятности риска. Он включает в себя оценку риска на основе сложности, критичности бизнеса, частоты использования, видимых областей, областей, подверженных дефектам и т. д. Это тестирование включает определение приоритетов тестирования модулей и функций тестируемого приложения на основе влияния и вероятности отказов.

Риск — возникновение неопределённого события. Оно положительно или отрицательно влияет на измеримые критерии успеха проекта. Это могут быть события, которые произошли в прошлом, а также текущие или будущие события.

Неопределённые события часто влияют на стоимость, бизнес, технические и качественные цели проекта. Позитивные риски упоминаются как возможности и помощь в устойчивости бизнеса.

Например, инвестирование в новый проект, изменение бизнес-процессов, разработка новых продуктов.

Отрицательные риски называются угрозами, и для успеха проекта должны быть реализованы рекомендации по их минимизации или устранению.

Примерный чек-лист для тестирования на основе рисков:

1. Важные функциональные возможности.
2. Видимая пользователю функциональность.
3. Функциональность, оказывающая наибольшее влияние на безопасность.
4. Функциональные возможности, которые оказывают наибольшее финансовое влияние на пользователей.
5. Сложные области исходного кода и кодов, подверженных ошибкам.
6. Особенности или функциональные возможности, добавленные в дизайн продукта в последнюю минуту.
7. Факторы подобных проектов, вызвавшие проблемы.

8. Факторы или проблемы аналогичных проектов, увеличившие эксплуатационные расходы.
9. Плохие требования.

Контрольные вопросы

1. В чём отличие модели жизненного цикла ПО от методологии разработки?
2. Какие модели жизненного цикла ПО вам известны?
3. Какие этапы проходит жизненный цикл тестирования?
4. Что такое Agile?
5. Какие подходы к тестированию вам известны?

Практическое задание

Ответьте на вопросы:

1. Изменилось ли восприятие деятельности тестировщика по сравнению с началом курса? Посмотрите, как вы отвечали на вопросы в практическом задании к уроку 1.
2. Назовите 3 плюса и 3 минуса в сфере тестирования лично для вас.
3. Как вы видите развитие в сфере тестирования?
4. Представьте, что вы работаете уже полгода. Опишите свой рабочий день.

---

*Тест для самопроверки - https://coreapp.ai/app/player/lesson/614344ea40a70c676714c75c (сдавать не нужно)

Глоссарий

Гибкие методологии разработки — обобщающий термин для нескольких подходов и практик, основанных на ценностях манифеста гибкой разработки программного обеспечения и 12 принципах, лежащих в его основе.

Методология разработки — это набор методов по управлению разработкой ПО, подборка практических правил и техник разработки.

Модель жизненного цикла ПО — описание, как проходит процесс разработки. В модели жизненного цикла ПО описывается, какие этапы проходит ПО, от рождения идеи до завершения использования, и что происходит с ПО на этих этапах.

Kanban — методология гибкой разработки, в основе которой лежит быстрота поставки готового продукта на рынок, а также равномерное распределение нагрузки между членами команды.

SCRUM — набор принципов, ценностей, политик, ритуалов, артефактов, на которых строится процесс SCRUM-разработки. Последний позволяет в жёстко фиксированные и небольшие по времени итерации, называемые спринтами, предоставлять конечному пользователю работающий продукт с новыми бизнес-возможностями, для которых определён наибольший приоритет.

Дополнительные материалы

1. Agile-манифест.
2. Статья «Кто такой скрам-мастер, и что входит в его обязанности».
3. Статья «Скрам — это эффективное управление проектами».
4. Статья «Scrum vs Kanban: в чём разница и что выбрать?».
5. Статья «Методология Kanban: введение».
6. Статья «Как объяснить дедушке, что такое Agile».
7. Статья «Опыт использования Scrumban в тестировании».
8. Статья «Скрам-мастер: что это за специалист и как им стать?».
9. Канбан — метод управления разработкой.
10. Статья «Основные методологии разработки программного обеспечения» (не только Agile).