**https://vladislaveremeev.gitbook.io/qa\_bible/sdlc-i-stlc/agile**

**Agile**

Agile - это способность создавать и реагировать на изменения. Это способ справиться с неопределенной и неспокойной средой и в конечном итоге преуспеть в ней. Авторы Agile Manifesto выбрали «Agile» в качестве названия всей этой идеи, потому что это слово олицетворяет адаптивность и реакцию на изменения, которые так важны для их подхода. На самом деле речь идет об осмыслении того, как вы можете понять, что происходит в среде, в которой вы находитесь сегодня, определить, с какой неопределенностью вы сталкиваетесь, и выяснить, как вы можете адаптироваться к этому по мере продвижения.

Гибкая разработка программного обеспечения - это больше, чем такие фреймворки, как Scrum, Extreme Programming или Feature-Driven Development (FDD). Гибкая разработка программного обеспечения - это больше, чем такие практики, как парное программирование, разработка на основе тестирования, стендапы, сессии планирования и спринты.

Гибкая разработка программного обеспечения - это общий термин для набора структур и практик, основанных на ценностях и принципах, изложенных в Манифесте гибкой разработки программного обеспечения и 12 принципах, лежащих в его основе. Когда вы подходите к разработке программного обеспечения особым образом, обычно хорошо жить в соответствии с этими ценностями и принципами и использовать их, чтобы помочь понять, что делать в вашем конкретном контексте.

Одна вещь, которая отличает Agile от других подходов к разработке программного обеспечения, - это сосредоточение внимания на людях, выполняющих работу, и на том, как они работают вместе. Решения развиваются в результате сотрудничества между самоорганизующимися кросс-функциональными командами, использующими соответствующие методы для своего контекста. Сообщество Agile-разработчиков программного обеспечения уделяет большое внимание совместной работе и самоорганизующейся команде. Это не значит, что менеджеров нет. Это означает, что команды могут самостоятельно определять, как они собираются подходить к делу. Это означает, что эти команды кросс-функциональны. Этим командам не обязательно должны быть задействованы определенные роли, просто когда вы собираете команду вместе, вы убедитесь, что у вас есть все необходимые навыки в команде. Еще есть место для менеджеров. Менеджеры следят за тем, чтобы члены команды имели или приобрели правильный набор навыков. Менеджеры создают среду, которая позволяет команде быть успешной. Менеджеры в основном отступают и позволяют своей команде выяснить, как они собираются выпускать продукты, но они вмешиваются, когда команды пытаются, но не могут решить проблемы. Когда большинство команд и организаций начинают заниматься гибкой разработкой, они сосредотачиваются на практиках, которые помогают в совместной работе и организации работы, и это здорово. Однако другой ключевой набор практик, которым не так часто следуют, но которые должны соблюдаться, - это конкретные технические практики, которые напрямую связаны с разработкой программного обеспечения таким образом, чтобы помочь вашей команде справиться с неопределенностью. Эти технические приемы очень важны, и их нельзя упускать из виду.

В конечном итоге Agile - это образ мышления, основанный на ценностях и принципах Agile Manifesto. Эти ценности и принципы содержат указания о том, как создавать изменения и реагировать на них, а также как справляться с неопределенностью. Можно сказать, что первое предложение Agile Manifesto заключает в себе всю идею: «Мы открываем лучшие способы разработки программного обеспечения, делая это и помогая другим делать это». Когда вы сталкиваетесь с неуверенностью, попробуйте что-то, что, по вашему мнению, может сработать, получите обратную связь и внесите соответствующие коррективы. При этом помните о ценностях и принципах. Позвольте вашему контексту определять, какие рамки, практики и методы вы используете для сотрудничества со своей командой и предоставления ценности вашим клиентам.

Если Agile - это образ мышления, то что это говорит об идее Agile-методологий? Чтобы ответить на этот вопрос, вам может быть полезно иметь четкое определение методологии. Алистер Кокберн предположил, что методология - это набор условностей, которым команда соглашается следовать. Это означает, что у каждой команды будет своя собственная методология, которая будет в малой или большой степени отличаться от методологии любой другой команды. Таким образом, Agile-методологии - это условности, которым команда решает следовать в соответствии с ценностями и принципами Agile. Вы, наверное, скажете: «Подождите, - я думал, что Scrum и XP - это Agile-методологии». Алистер применил термин “framework” к этим концепциям. Они, безусловно, родились на основе методологии одной команды, но они стали фреймворками, когда были обобщены для использования другими командами. Эти фреймворки помогают понять, где команда начинает свою методологию, но они не должны быть ее методологией. Команде всегда необходимо адаптировать использование фреймворка, чтобы оно соответствовало его контексту.

**Ключевые концепции Agile**

* Пользовательские истории (**User Stories**): после консультации с заказчиком или владельцем продукта команда делит работу, которую необходимо выполнить, на функциональные этапы, называемые «пользовательскими историями». Ожидается, что каждая пользовательская история внесет свой вклад в ценность всего продукта;
* Ежедневные собрания (**Daily Meeting**): каждый день в одно и то же время группа собирается, чтобы ознакомить всех с информацией, которая имеет жизненно важное значение для координации: каждый член команды кратко описывает все «завершенные» вклады и любые препятствия, стоящие на их пути;
* Персонажи (**Personas**): когда этого требует проект - например, когда пользовательский опыт является основным фактором результатов проекта - команда создает подробные синтетические биографии фиктивных пользователей будущего продукта: они называются personas;
* Команда (**Team**): «Команда» в Agile понимании - это небольшая группа людей, назначенных на один и тот же проект или effort, почти все из них на постоянной основе. Незначительное меньшинство членов команды может работать неполный рабочий день или иметь конкурирующие обязанности;
* Инкрементальная разработка (**Incremental Development**): почти все Agile-команды отдают предпочтение стратегии инкрементального развития; в контексте Agile это означает, что можно использовать каждую последующую версию продукта, и каждая основывается на предыдущей версии, добавляя видимые для пользователя функциональные возможности;
* Итеративная разработка (**Iterative Development**): Agile-проекты являются итеративными, поскольку они намеренно позволяют «повторять» действия по разработке программного обеспечения и потенциально «пересматривать» одни и те же рабочие продукты;
* Ретроспектива (**Milestone Retrospective**): после того, как проект был запущен в течение некоторого времени или в конце проекта, все постоянные члены команды (не только разработчики) вкладывают от одного до трех дней в подробный анализ значимых событий проекта.

**The Agile Manifesto**:

* люди и взаимодействие важнее процессов и инструментов;
* работающий продукт важнее исчерпывающей документации;
* сотрудничество с заказчиком важнее согласования условий контракта;
* готовность к изменениям важнее следования первоначальному плану.

Основополагающие принципы Agile Manifesto:

* наивысшим приоритетом признается удовлетворение заказчика за счет ранней и бесперебойной поставки ценного программного обеспечения;
* изменение требований приветствуется даже в конце разработки (это может повысить конкурентоспособность полученного продукта);
* частая поставка работающего программного обеспечения (каждые пару недель или пару месяцев с предпочтением меньшего периода);
* общение представителей бизнеса с разработчиками должно быть ежедневным на протяжении всего проекта;
* проекты следует строить вокруг заинтересованных людей, которых следует обеспечить нужными условиями работы, поддержкой и доверием;
* самый эффективный метод обмена информацией в команде - личная встреча;
* работающее программное обеспечение - лучший измеритель прогресса;
* спонсоры, разработчики и пользователи должны иметь возможность поддерживать постоянный темп на неопределённый срок;
* постоянное внимание к техническому совершенству и хорошему проектированию увеличивают гибкость;
* простота как искусство не делать лишней работы очень важна;
* лучшие требования, архитектура и проектные решения получаются у самоорганизующихся команд;
* команда регулярно обдумывает способы повышения своей эффективности и соответственно корректирует рабочий процесс.

**Существуют методологии**, которые придерживаются ценностей и принципов заявленных в Agile Manifesto, некоторые из них:

* Agile Modeling - набор понятий, принципов и приемов (практик), позволяющих быстро и просто выполнять моделирование и документирование в проектах разработки программного обеспечения. Не включает в себя детальную инструкцию по проектированию, не содержит описаний, как строить диаграммы на UML. Основная цель: эффективное моделирование и документирование; но не охватывает программирование и тестирование, не включает вопросы управления проектом, развертывания и сопровождения системы. Однако включает в себя проверку модели кодом.
* Agile Unified Process (AUP) упрощенная версия IBM Rational Unified Process (RUP), разработанная Скоттом Амблером, которая описывает простое и понятное приближение (модель) для создания программного обеспечения для бизнес-приложений.
* Agile Data Method- группа итеративных методов разработки программного обеспечения, в которых требования и решения достигаются в рамках сотрудничества разных кросс-функциональных команд.
* DSDM основан на концепции быстрой разработки приложений (Rapid Application Development, RAD). Представляет собой итеративный и инкрементный подход, который придаёт особое значение продолжительному участию в процессе пользователя/потребителя.
* Essential Unified Process (EssUP).
* Экстремальное программирование (Extreme programming, XP).
* Feature driven development (FDD) - функционально-ориентированная разработка. Используемое в FDD понятие функции или свойства (англ. feature) системы достаточно близко к понятию прецедента использования, используемому в RUP, существенное отличие - это дополнительное ограничение: «каждая функция должна допускать реализацию не более, чем за две недели». То есть если сценарий использования достаточно мал, его можно считать функцией. Если же велик, то его надо разбить на несколько относительно независимых функций.
* Getting Real - итеративный подход без функциональных спецификаций, использующийся для веб-приложений. В данном методе сперва разрабатывается интерфейс программы, а потом её функциональная часть.
* OpenUP - это итеративно-инкрементальный метод разработки программного обеспечения. Позиционируется как легкий и гибкий вариант RUP. OpenUP делит жизненный цикл проекта на четыре фазы: начальная фаза, фазы уточнения, конструирования и передачи. Жизненный цикл проекта обеспечивает предоставление заинтересованным лицам и членам коллектива точек ознакомления и принятия решений на протяжении всего проекта. Это позволяет эффективно контролировать ситуацию и вовремя принимать решения о приемлемости результатов. План проекта определяет жизненный цикл, а конечным результатом является окончательное приложение.
* Scrum устанавливает правила управления процессом разработки и позволяет использовать уже существующие практики кодирования, корректируя требования или внося тактические изменения. Использование этой методологии дает возможность выявлять и устранять отклонения от желаемого результата на более ранних этапах разработки программного продукта.
* Бережливая разработка программного обеспечения (lean software development) использует подходы из концепции бережливого производства.

**Манифест тестирования в Agile**:

* постоянное тестирование, а не только в конце разработки;
* предотвращение багов более значимо, чем их поиск;
* понимание тестируемого продукта выше проверки функционала;
* построение лучшей системы в связке с командой выше поиска методов ее сломать;
* вся команда отвечает за качество, а не только тестировщик.

**Особенности тестирования в Agile**

В Agile, тестирование - это ответственность каждого. Критерий качества должен соблюдаться на протяжении всего цикла. В то время как бизнес-аналитики сосредотачиваются на создании подробных пользовательских историй, разработчики сосредотачиваются на разработке качественного кода, QA несет ответственность за уточнение критериев приемлемости для каждой пользовательской истории, тестирование завершенной функциональности в каждом спринте с точки зрения клиента и проверка всей ранее выполненной функциональности. Роли и ответственность QA не ограничиваются только тестированием в Agile, но также включают в себя следующее:

* QA работают в тесном сотрудничестве с владельцами продуктов, бизнес-аналитиками и командой разработчиков, чтобы понять разрабатываемый продукт, для кого он предназначен и каковы будут критерии успеха продукта.
* Они углубляются в критерии приемки, созданные бизнес-аналитиками, для написания тестовых примеров, визуализации рабочего процесса, тестирования стандартных элементов и проведения негативного тестирования.
* Опытный QA также хорошо осведомлен об объеме выпуска и соответственно устанавливает границы своего тестирования.
* Тестировщики должны общаться с командой и задавать вопросы во время тестирования, что позволяет им выявлять пробелы в требованиях или получать ответы на нерешенные вопросы. Коммуникация и сотрудничество с командой имеют решающее значение, поскольку они помогают сделать тестирование более точным и надежным. Это также помогает достичь необходимого темпа, чтобы двигаться быстрее, с ранними отзывами о тестировании и повышенным качеством.
* Как член Agile команды, тестировщик всегда должен быть синхронизирован с командой проекта, посещая сессии планирования спринтов для выявления возможных проблемных областей и ежедневных митингов для содействия сотрудничеству. Посещение ретроспективы спринта дает возможность выявить слабые места и определить решения внутри команды. Посещение митингов по обзору спринта или демонстрации продукта позволяет тестировщику увидеть, как работает новая функция, и дает им возможность задать важные вопросы разработчикам.
* Документирование сценариев тестирования и выполнения тестов с доказательствами важно для тестировщиков, но оно должно быть минимальным и кратким.

Какие стратегии использует QA для проведения Agile-тестирования?

В каждой организации есть разные подходы и стратегии, которые они используют для тестирования приложений. Методология Agile предполагает подготовку документации, достаточной только для удовлетворения насущных потребностей команды. Таким образом, QA подготовят достаточно высокоуровневой документации для стратегий тестирования и планов для руководства командой. Ниже приведены несколько стратегий, которым я следую при подготовке к тестированию в Agile:

* Чрезвычайно важно иметь четкий план до начала тестирования. Перед началом тестирования спланируйте свое время и тест-кейсы, и это позволит вам сразу же погрузиться в тестирование после развертывания приложения.
* Я использую сочетание ручного и автоматизированного тестирования. Автоматизированное тестирование помогает мне ускорить мои регрессионные тесты с помощью наборов тестов перед сборкой, а ручное тестирование помогает мне, когда мне нужно проводить более сфокусированное тестирование.
* Как тестировщик, я всегда верю в проведение смоук-тестов основных или базовых функций сразу после развертывания приложения, что помогает мне выявлять любые ошибки критические раньше.
* Выполнение приемочных испытаний, основанных на критериях приемки, чтобы обеспечить лучшее покрытие тестами. Каждый критерий приемки может иметь одно или несколько приемочных испытаний и ориентирован на заданные условия.
* Тестирование эффективности потока помогает определить, могут ли пользователи беспрепятственно перемещаться по продукту. Это помогает определить, сбивает ли вас с толку какая-либо часть потока, и помогает определить, нужно ли вам добавлять или удалять какие-либо шаги.
* Проверка бизнес-правил и определений данных - важная часть тестирования.
* Проведение исследовательского тестирования позволяет тестировщикам идти по неопределенному пути и находить скрытые риски и дефекты в приложении.
* Проведение регрессионного тестирования - важная часть гибкого тестирования. Регрессионное тестирование гарантирует, что проверенные функции работают должным образом после введения новых функций.
* Использование кроссбраузерного тестирования чрезвычайно важно для гибкого тестирования, так как тестировщик может быстро протестировать несколько устройств и браузеров за ограниченное время.
* Наличие приложений для обмена сообщениями в реальном времени, таких как Slack и Zoom, позволяет всем в команде быть на связи и быстро отвечать на важные вопросы.