Лекция 1.2. Понимание SDLC и STLC

- SDLC — это аббревиатура от термина Software Development Life Cycle «Жизненный цикл разработки программного обеспечения».
- Это также называется процессом разработки программного обеспечения.
- SDLC — это структура, определяющая задачи, выполняемые на каждом этапе процесса разработки программного обеспечения.
- ISO/IEC 12207 — это международный стандарт для процессов жизненного цикла программного обеспечения. Он стремится быть стандартом, определяющим все задачи, необходимые для разработки и сопровождения программного обеспечения.
- Он состоит из подробного плана, описывающего, как разрабатывать, поддерживать, заменять и изменять или улучшать конкретное программное обеспечение. Жизненный цикл определяет методологию улучшения качества программного обеспечения и всего процесса разработки.

На рисунке 1 представлено графическое представление различных этапов типичного SDLC. Он состоит из подробного плана, описывающего, как разрабатывать, поддерживать, заменять и изменять или улучшать конкретное программное обеспечение. Жизненный цикл определяет методологию улучшения качества программного обеспечения и всего процесса разработки.

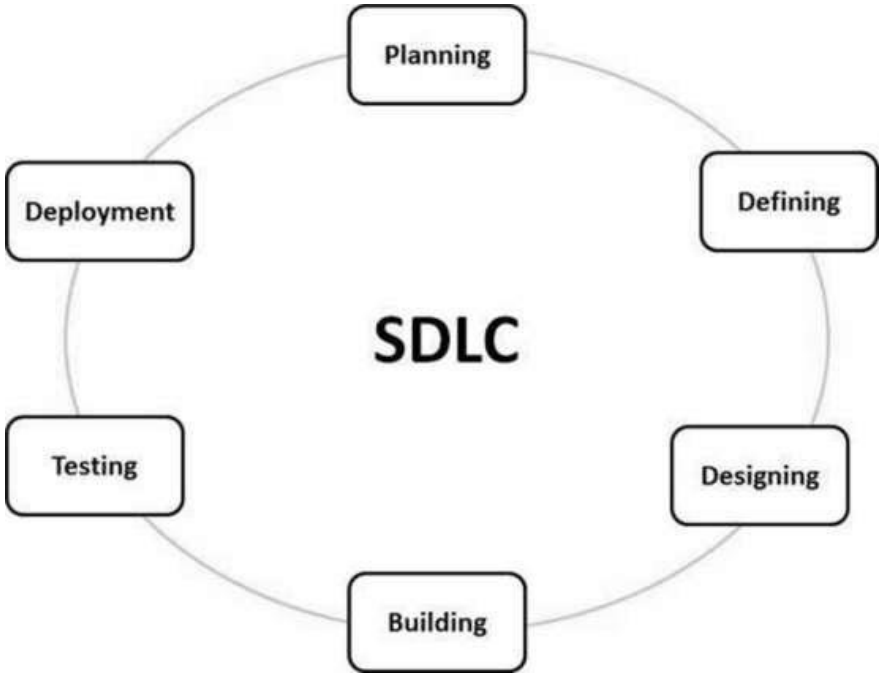

Рисунок 1 – Типичный жизненный цикл разработки программного обеспечения состоит из следующих этапов

Этап 1: Планирование и анализ требований

Анализ требований - самый важный и фундаментальный этап в SDLC. Он выполняется старшими членами команды при участии клиента, отдела продаж, маркетинговых исследований и отраслевых экспертов. Эта информация затем используется для планирования базового подхода к проекту и проведения технико-экономического обоснования продукта в экономической, эксплуатационной и технической областях.

Планирование требований к обеспечению качества и выявление рисков, связанных с проектом, также выполняется на стадии планирования. Результатом технико-экономического обоснования является определение различных технических подходов, которым можно следовать для успешной реализации проекта с минимальными рисками.

Этап 2: Определение требований/Проектирование и дизайн

Следующим шагом после проведения анализа требований является четкое определение и документирование требований к продукту и их утверждение от клиента или рыночных аналитиков. Это делается с помощью документа SRS (Спецификация требований к программному обеспечению), который состоит из всех требований к продукту, которые должны быть спроектированы и разработаны в течение жизненного цикла проекта.

Необходимо задать следующий вопрос: «Как мы добьемся наших целей?» Иначе говоря, вам необходимо понять, что именно вы оптимизируете и проектировать соответствующе.

После фазы дизайна вы наконец-то сможете засесть за клавиатуры, и внесение изменений в отношении времени и потраченных ресурсов будет неуклонно расти, а также буду постепенно накапливаться всевозможные малые факторы. В этой фазе для принятия окончательных решений по вопросам дизайна я рекомендую учитывать несколько основных его элементов: операционное превосходство, безопасность, надежность, эффективность производительности, и оптимизация затрат.

Этап 3: Разработка архитектуры продукта

SRS — это ориентир для архитекторов продукта, который может предложить лучшую архитектуру продукта, который будет разработан. На основе требований, указанных в SRS, обычно предлагается более одного подхода к проектированию для архитектуры продукта и документируется в DDS — Design Document Specification.

Этот DDS рассматривается всеми важными заинтересованными сторонами и на основе различных параметров, таких как оценка рисков, надежность продукта, модульность конструкции, бюджет и временные ограничения, выбирается лучший подход к дизайну для продукта.

Подход к проектированию четко определяет все архитектурные модули продукта вместе с его взаимодействием и представлением потока данных с внешними и сторонними модулями (если таковые имеются). Внутренний дизайн всех модулей предлагаемой архитектуры должен быть четко определен с мельчайшими деталями в DDS.

Этап 4: Создание или разработка продукта

На этом этапе SDLC начинается фактическая разработка и создается продукт. На этом этапе создается программный код согласно DDS. Если проектирование выполняется детально и организованно, генерация кода может выполняться без особых хлопот.

Разработчики должны следовать руководящим принципам кодирования, определенным их организацией, и инструменты программирования, такие как компиляторы, интерпретаторы, отладчики и т.д., используются для генерации кода. Для кодирования используются различные языки программирования высокого уровня, такие как C, C++, Pascal, Java и PHP. Язык программирования выбирается в зависимости от типа разрабатываемого программного обеспечения.

Этап 5: Тестирование продукта

Этот этап обычно является подмножеством всех этапов, так как в современных моделях SDLC действия по тестированию в основном задействованы на всех этапах SDLC. Однако этот этап относится только к этапу тестирования продукта, когда дефекты продукта сообщаются, отслеживаются, исправляются и повторно тестируются, пока продукт не достигнет стандартов качества, определенных в SRS.

Этап 6: Размещение на рынке и обслуживание

После того, как продукт протестирован и готов к развертыванию, он официально выпускается на соответствующий рынок. Иногда развертывание продукта происходит поэтапно в соответствии с бизнес-стратегией этой организации. Сначала продукт может быть выпущен в ограниченном сегменте и протестирован в реальной бизнес-среде (UAT — пользовательское приемочное тестирование).

Затем, основываясь на отзывах, продукт может быть выпущен как есть или с предлагаемыми улучшениями в целевом сегменте рынка. После того, как продукт выпущен на рынок, его обслуживание выполняется для существующей клиентской базы.

Модели SDLC

Определены и разработаны различные модели жизненного цикла разработки программного обеспечения, которым следуют в процессе разработки программного обеспечения. Эти модели также называются моделями процессов разработки программного обеспечения. Каждая модель процесса следует серии шагов, уникальных для своего типа, чтобы обеспечить успех в процессе разработки программного обеспечения.

Ниже приведены наиболее важные и популярные модели SDLC, используемые в отрасли.

- Модель водопада.
- Итерационная модель.
- Спиральная модель.

- V-модель.
- Модель большого взрыва.

К другим связанным методологиям относятся Agile Model, RAD Model, Rapid Application Development и Prototyping Models.

1. Модель водопада

Модель водопада была первой моделью процесса, которая была представлена. Он также называется линейно-последовательной моделью жизненного цикла. Это очень просто понять и использовать. В модели водопада каждая фаза должна быть завершена до того, как может начаться следующая фаза, и в фазах нет совпадений.

Модель Waterfall — самый ранний подход SDLC, который использовался для разработки программного обеспечения.

Модель водопада — дизайн

Модель водопада иллюстрирует процесс разработки программного обеспечения в линейном последовательном потоке. Это означает, что любой этап в процессе разработки начинается, только если предыдущий этап завершен. В этой модели водопада фазы не перекрываются.

Водопадный подход был первой моделью SDLC, которая широко использовалась в программной инженерии для обеспечения успеха проекта. В подходе «Водопад» весь процесс разработки программного обеспечения делится на отдельные фазы. В этой модели водопада, как правило, результат одной фазы действует как вход для следующей фазы последовательно.

Следующая иллюстрация (рисунок 2) представляет различные фазы модели водопада.

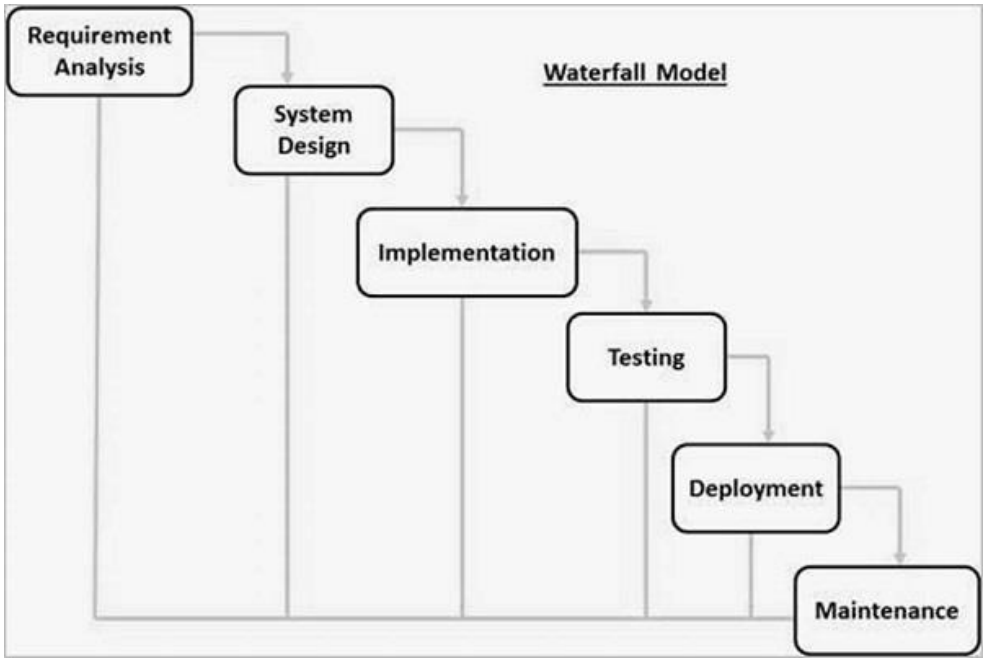

Рисунок 2 – Модель водопада

Последовательные фазы в модели Waterfall:

- Сбор и анализ требований — Все возможные требования к разрабатываемой системе фиксируются на этом этапе и документируются в документе спецификации требований.
- Проектирование системы. На этом этапе изучаются требования к требованиям первого этапа и готовится проектирование системы. Такая конструкция системы помогает определить требования к оборудованию и системе и помогает определить общую архитектуру системы.
- Внедрение. С учетом исходных данных, полученных при проектировании системы, система сначала разрабатывается в виде небольших программ, называемых модулями, которые интегрируются на следующем этапе. Каждое устройство разработано и проверено на его функциональность, которая называется модульным тестированием.
- Интеграция и тестирование — все модули, разработанные на этапе внедрения, интегрируются в систему после тестирования каждого модуля. После интеграции вся система проверяется на наличие ошибок и сбоев.
- Развертывание системы — после завершения функционального и нефункционального тестирования; продукт развернут в среде клиента или выпущен на рынок.
- Обслуживание — Есть некоторые проблемы, которые возникают в клиентской среде. Чтобы исправить эти проблемы, исправления выпущены. Также для улучшения продукта выпущены лучшие версии. Техническое обслуживание проводится для предоставления этих изменений в среде клиента.

Сбор и анализ требований — все возможные требования к разрабатываемой системе фиксируются на этом этапе и документируются в документе спецификации требований.

Проектирование системы. На этом этапе изучаются требования к требованиям первого этапа и готовится проектирование системы. Такая конструкция системы помогает определить требования к оборудованию и системе и помогает определить общую архитектуру системы.

Внедрение. С учетом исходных данных, полученных при проектировании системы, система сначала разрабатывается в виде небольших программ, называемых модулями, которые интегрируются на следующем этапе. Каждое устройство разработано и проверено на его функциональность, которая называется модульным тестированием.

Интеграция и тестирование — все модули, разработанные на этапе внедрения, интегрируются в систему после тестирования каждого модуля. После интеграции вся система проверяется на наличие ошибок и сбоев.

Развертывание системы — после завершения функционального и нефункционального тестирования; продукт развернут в среде клиента или выпущен на рынок.

Обслуживание — Есть некоторые проблемы, которые возникают в клиентской среде. Чтобы исправить эти проблемы, исправления выпущены. Также для улучшения продукта выпущены лучшие версии. Техническое обслуживание проводится для предоставления этих изменений в среде клиента.

Все эти фазы каскадно связаны друг с другом, в которых прогресс рассматривается как непрерывно нисходящий (как водопад) через фазы. Следующий этап начинается только после того, как определенный набор целей достигнут для предыдущего этапа, и он подписан, поэтому называется «Модель водопада». В этой модели фазы не перекрываются.

Модель водопада — применение

Каждое разработанное программное обеспечение отличается и требует соответствующего подхода SDLC, основанного на внутренних и внешних факторах. Некоторые ситуации, в которых использование модели Водопад является наиболее подходящим:

- Требования очень хорошо задокументированы, понятны и зафиксированы.
- Определение продукта является стабильным.
- Технология понятна и не динамична.
- Здесь нет двусмысленных требований.
- Достаточные ресурсы с необходимым опытом доступны для поддержки продукта.
- Проект короткий.

Модель водопада — преимущества

Преимущества развития водопада состоят в том, что он позволяет отделить и контролировать. График может быть установлен со сроками для каждого этапа разработки, и продукт может проходить этапы модели процесса разработки один за другим.

Разработка переходит от концепции к проектированию, внедрению, тестированию, установке, устранению неполадок и заканчивается эксплуатацией и обслуживанием. Каждый этап развития протекает в строгом порядке.

Некоторые из основных преимуществ модели водопада:

- Просто и легко понять и использовать
- Простота в управлении благодаря жесткости модели. Каждый этап имеет конкретные результаты и процесс обзора.
- Фазы обрабатываются и завершаются по одному.
- Хорошо работает для небольших проектов, где требования очень хорошо поняты.
- Четко определенные этапы.

- Хорошо понятные вехи.
- Легко организовать задачи.
- Процесс и результаты хорошо документированы.

Модель водопада — недостатки

Недостатком развития водопада является то, что он не позволяет много размышлений или пересмотра. Когда приложение находится на стадии тестирования, очень трудно вернуться назад и изменить что-то, что не было хорошо документировано или продумано на стадии разработки.

Основные недостатки модели водопада:

- Никакое рабочее программное обеспечение не производится до конца жизненного цикла.
- Большое количество риска и неопределенности.
- Не очень хорошая модель для сложных и объектно-ориентированных проектов.
- Плохая модель для длительных и текущих проектов.
- Не подходит для проектов, где требования изменяются от умеренного до высокого риска. Таким образом, риск и неопределенность высоки с этой моделью процесса.
- Трудно измерить прогресс в несколько этапов.
- Невозможно учесть меняющиеся требования
- Регулировка объема в течение жизненного цикла может завершить проект.

Интеграция осуществляется как «большой взрыв» в самом конце, что не позволяет выявить какие-либо технологические или бизнес-узкие места или проблемы на ранних этапах.

2. SDLC — Итерационная модель

В итеративной модели итерационный процесс начинается с простой реализации небольшого набора требований к программному обеспечению и итеративно улучшает развивающиеся версии, пока вся система не будет реализована и не будет готова к развертыванию.

Итеративная модель жизненного цикла не пытается начать с полной спецификации требований. Вместо этого разработка начинается с определения и реализации только части программного обеспечения, которая затем проверяется для определения дальнейших требований. Затем этот процесс повторяется, создавая новую версию программного обеспечения в конце каждой итерации модели.

Итерационная модель —дизайн

Итерационный процесс начинается с простой реализации подмножества требований к программному обеспечению и итеративно улучшает развивающиеся версии, пока не будет реализована вся система. На каждой итерации вносятся изменения в дизайн и добавляются новые функциональные возможности. Основная идея, лежащая в основе этого метода, заключается в разработке системы

посредством повторяющихся циклов (итеративно) и небольшими частями за раз (инкрементально).

Следующая иллюстрация (рисунок 3) представляет собой итеративную и инкрементную модель.

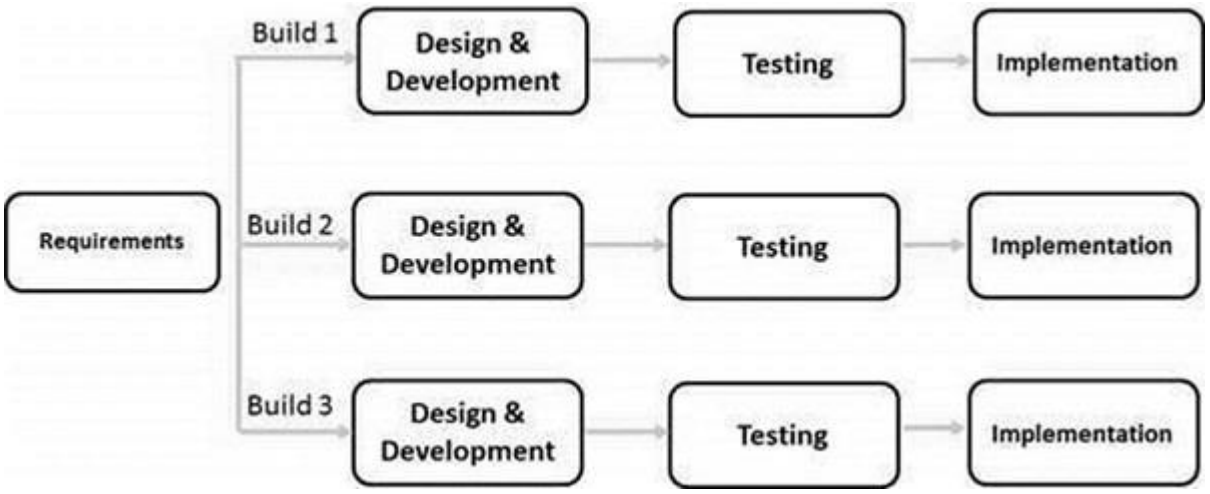

Рисунок 3 – Итеративная модель

Итеративная и инкрементная разработка — это комбинация итеративного проектирования или итеративного метода и модели инкрементальной сборки для разработки. «Во время разработки программного обеспечения может выполняться более одной итерации цикла разработки программного обеспечения одновременно». Этот процесс можно описать как подход «эволюционного приобретения» или «постепенного наращивания».

В этой инкрементальной модели все требования разделены на различные сборки. Во время каждой итерации модуль разработки проходит этапы требований, проектирования, реализации и тестирования. Каждый последующий выпуск модуля добавляет функции к предыдущему выпуску. Процесс продолжается до тех пор, пока вся система не будет готова в соответствии с требованиями.

Ключом к успешному использованию итеративного жизненного цикла разработки программного обеспечения является тщательная проверка требований, а также проверка и тестирование каждой версии программного обеспечения на соответствие этим требованиям в рамках каждого цикла модели. По мере того, как программное обеспечение развивается через последовательные циклы, тесты необходимо повторять и расширять для проверки каждой версии программного обеспечения.

Итерационная модель — приложение

Как и другие модели SDLC, итеративная и инкрементная разработка имеет некоторые специфические приложения в индустрии программного обеспечения. Эта модель чаще всего используется в следующих сценариях:

- Требования ко всей системе четко определены и понятны.
- Должны быть определены основные требования; однако некоторые функции или требуемые улучшения могут со временем развиваться.
- Пришло время рыночным ограничениям.
- Новая технология используется и изучается командой разработчиков во время работы над проектом.
- Ресурсы с необходимыми наборами навыков недоступны и планируется использовать на контрактной основе для конкретных итераций.
- Есть некоторые функции и цели с высоким риском, которые могут измениться в будущем.

Итерационная модель — плюсы и минусы

Преимущество этой модели в том, что на самом раннем этапе разработки имеется рабочая модель системы, что упрощает поиск функциональных или конструктивных недостатков. Обнаружение проблем на ранней стадии разработки позволяет принимать корректирующие меры в ограниченном бюджете.

Недостатком этой модели SDLC является то, что она применима только к большим и громоздким проектам разработки программного обеспечения. Это связано с тем, что небольшую программную систему трудно разбить на дополнительные небольшие обслуживаемые приращения / модули.

Преимущества итеративной и инкрементной модели SDLC заключаются в следующем:

- Некоторые рабочие функции можно разработать быстро и на ранних этапах жизненного цикла.
- Результаты получаются рано и периодически.
- Возможна параллельная разработка.
- Прогресс можно измерить.
- Менее затратно изменить объем / требования.
- Тестировать и отлаживать во время небольшой итерации легко.
- Риски выявляются и устраняются во время итерации и каждая итерация — это легко управляемая веха.
- Легче управлять риском - сначала выполняется часть с высоким риском.
- С каждым приращением доставляется рабочий продукт.
- Проблемы, проблемы и риски, выявленные на каждом этапе, могут быть использованы / применены к следующему этапу.
- Анализ рисков лучше.
- Он поддерживает меняющиеся требования.
- Начальное время работы меньше.
- Лучше подходит для крупных и критически важных проектов.
- В течение жизненного цикла программное обеспечение создается на ранней стадии, что облегчает оценку и обратную связь с клиентами.

Недостатки итеративной и инкрементной модели SDLC следующие:

- Могут потребоваться дополнительные ресурсы.

- Хотя стоимость изменения меньше, но это не очень подходит для меняющихся требований.
- Требуется больше внимания со стороны руководства.
- Проблемы системной архитектуры или проектирования могут возникнуть из-за того, что не все требования собраны в начале всего жизненного цикла.
- Для определения приращений может потребоваться определение всей системы.
- Не подходит для небольших проектов.
- Сложность управления больше.
- Конец проекта может быть неизвестен, что является риском.
- Для анализа рисков требуются высококвалифицированные ресурсы.
- Прогресс проектов во многом зависит от фазы анализа рисков.

3. SDLC — спиральная модель

Спиральная модель сочетает в себе идею итеративного развития с систематическими, управляемыми аспектами модели водопада. Эта спиральная модель представляет собой комбинацию модели итеративного процесса разработки и модели последовательной линейной разработки, то есть каскадной модели с очень большим упором на анализ рисков. Это позволяет производить инкрементные выпуски продукта или инкрементные доработки на каждой итерации по спирали.

Модель спирали — дизайн

Спиральная модель состоит из четырех фаз. Программный проект многократно проходит эти фазы в итерациях, называемых спиралями.

Идентификация

Этот этап начинается со сбора бизнес-требований по базовой спирали. В последующих спиралях по мере развития продукта на этом этапе выполняется идентификация системных требований, требований подсистем и требований к устройствам.

Этот этап также включает понимание требований к системе путем непрерывного взаимодействия между заказчиком и системным аналитиком. В конце спирали продукт внедряется на идентифицированный рынок.

Дизайн

Этап проектирования начинается с концептуального проектирования в базовой спирали и включает архитектурный дизайн, логический дизайн модулей, физический дизайн продукта и окончательный дизайн в последующих спиралях.

Построить или построить

Фаза построения относится к производству фактического программного продукта на каждой спирали. В базовой спирали, когда продукт только обдумывается, а дизайн разрабатывается, на этом этапе разрабатывается POC (Proof of Concept), чтобы получить отзывы клиентов.

Затем в последующих спиралях с большей ясностью требований и деталей дизайна создается рабочая модель программного обеспечения, называемая сборкой, с номером версии. Эти сборки отправляются заказчику для обратной связи.

Оценка и анализ рисков

Анализ рисков включает выявление, оценку и мониторинг технической осуществимости и рисков управления, таких как отставание от графика и перерасход средств. После тестирования сборки в конце первой итерации заказчик оценивает программное обеспечение и предоставляет отзывы.

На следующем рисунке (рисунок 4) представлена спиральная модель, в которой перечислены действия на каждой фазе.

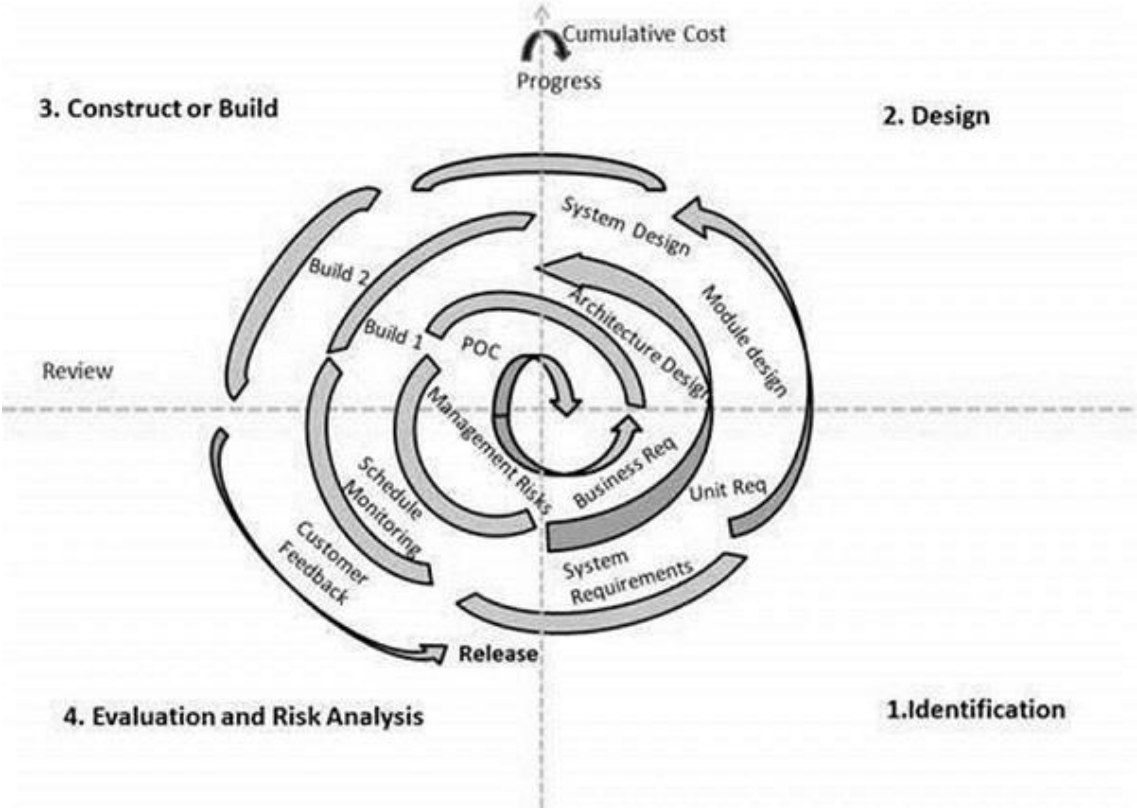

Рисунок 4 – Спиральная модель

На основе оценки клиента процесс разработки программного обеспечения переходит на следующую итерацию и впоследствии следует линейному подходу для реализации обратной связи, предложенной клиентом. Процесс итераций по спирали продолжается на протяжении всего срока службы программного обеспечения.

Применение спиральной модели

Спиральная модель широко используется в индустрии программного обеспечения, поскольку она синхронизируется с естественным процессом разработки любого продукта, то есть с обучением со зрелостью, которое предполагает минимальный риск как для заказчика, так и для фирм-разработчиков.

Следующие указатели объясняют типичное использование спиральной модели.

- Когда есть бюджетные ограничения и оценка рисков важна.
- Для проектов со средним и высоким уровнем риска.
- Долгосрочные обязательства по проекту из-за возможных изменений экономических приоритетов по мере изменения требований со временем.
- Заказчик не уверен в своих требованиях, что обычно бывает.
- Требования сложны и требуют оценки для ясности.
- Новая линейка продуктов, которая должна выпускаться поэтапно, чтобы получить достаточную обратную связь от клиентов.
- В процессе разработки продукта ожидаются существенные изменения.

Модель спирали — плюсы и минусы

Преимущество спиральной модели жизненного цикла состоит в том, что она позволяет добавлять элементы продукта, когда они становятся доступными или известными. Это гарантирует отсутствие противоречий с предыдущими требованиями и дизайном.

Этот метод согласуется с подходами, предусматривающими несколько сборок и выпусков программного обеспечения, что позволяет упорядоченно переходить к обслуживанию. Еще одним положительным аспектом этого метода является то, что спиральная модель вынуждает пользователей на раннем этапе участвовать в разработке системы.

С другой стороны, для завершения таких продуктов требуется очень строгое управление, и есть риск запустить спираль в неопределенный цикл. Таким образом, дисциплина изменений и степень принятия запросов на изменение очень важны для успешной разработки и развертывания продукта.

Преимущества модели Spiral SDLC следующие:

- Изменяющиеся требования могут быть учтены.
- Позволяет широко использовать прототипы.
- Требования можно фиксировать более точно.
- Пользователи рано видят систему.
- Разработка может быть разделена на более мелкие части, а рискованные части могут быть разработаны раньше, что помогает в более эффективном управлении рисками.

Недостатки модели Spiral SDLC следующие:

- Управление более сложное.
- О завершении проекта может быть не известно рано.
- Не подходит для небольших проектов или проектов с низким уровнем риска и может быть дорогостоящим для небольших проектов.
- Процесс сложный
- Спираль может продолжаться бесконечно.

- Большое количество промежуточных этапов требует излишней документации.

4. SDLC — V-модель

V-модель — это модель SDLC, в которой выполнение процессов происходит последовательно в V-образной форме. Она также известна как модель проверки и проверки.

V-модель является расширением модели водопада и основана на объединении фазы тестирования для каждой соответствующей стадии разработки. Это означает, что для каждой фазы в цикле разработки есть непосредственно связанная фаза тестирования. Это очень дисциплинированная модель, и следующий этап начинается только после завершения предыдущего.

V-Model — дизайн

В рамках V-модели параллельно планируется соответствующая фаза тестирования на этапе разработки. Таким образом, есть этапы проверки с одной стороны от фазы «V» и фазы проверки с другой стороны. Фаза кодирования объединяет две стороны V-модели.

На следующем рисунке 5 показаны различные фазы V-модели SDLC.

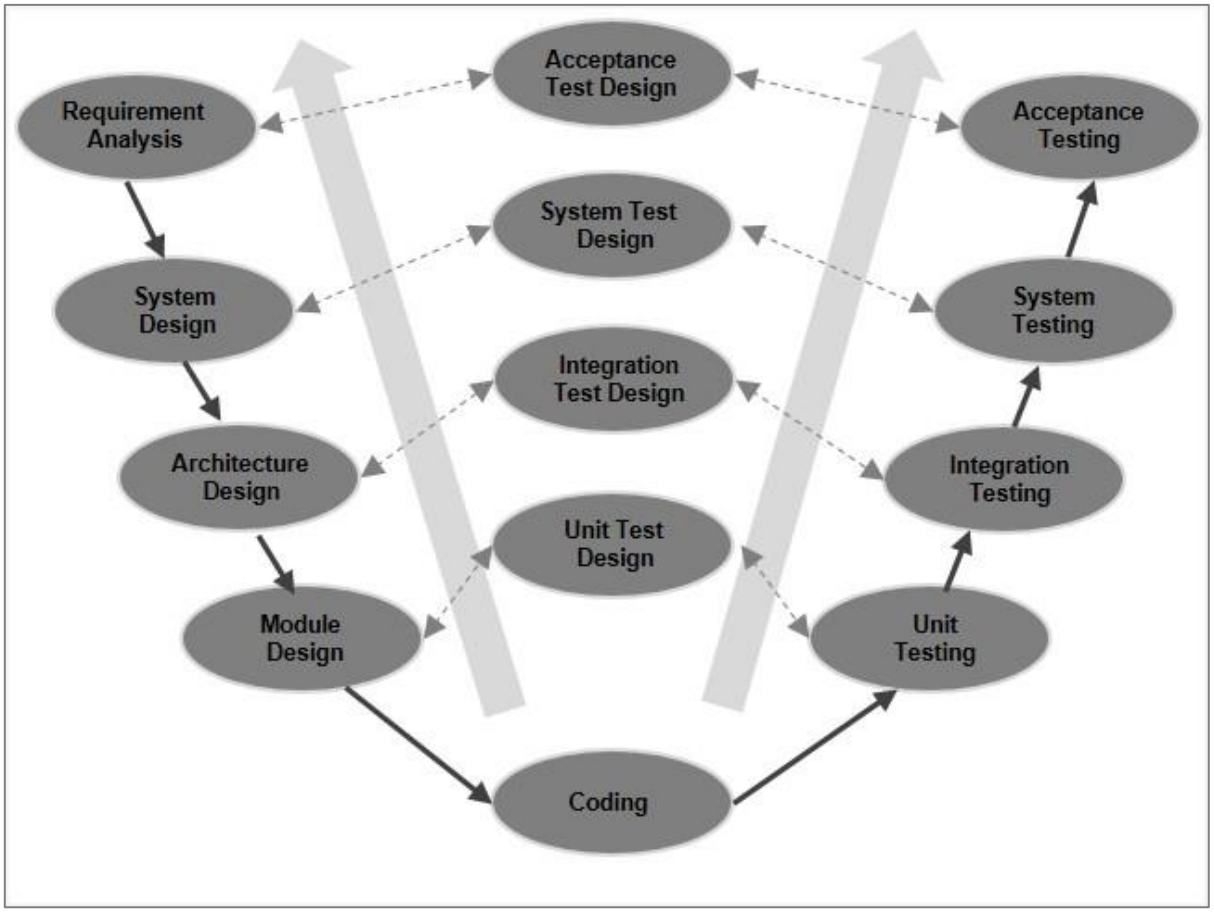

Рисунок 5 – Фазы V-модели SDLC

V-модель — этапы проверки

В V-модели есть несколько этапов проверки, каждая из которых подробно описана ниже.

Анализ бизнес-требований

Это первая фаза в цикле разработки, на которой требования к продукту понимаются с точки зрения потребителя. Этот этап включает в себя подробное общение с заказчиком, чтобы понять его ожидания и точные требования. Это очень важное мероприятие, которым необходимо хорошо управлять, поскольку большинство клиентов не уверены в том, что именно им нужно. На этом этапе выполняется планирование проектирования приемочных испытаний, поскольку бизнес-требования могут использоваться в качестве входных данных для приемочных испытаний.

Системный дизайн

После того, как у вас появятся четкие и подробные требования к продукту, пора спроектировать всю систему. Системный дизайн будет иметь понимание и подробную настройку оборудования и связи для разрабатываемого продукта. План тестирования системы разрабатывается на основе проекта системы. Выполнение этого на более раннем этапе оставляет больше времени для фактического выполнения теста позже.

Архитектурный дизайн

На этом этапе понимаются и разрабатываются архитектурные спецификации. Обычно предлагается более одного технического подхода, и на основе технической и финансовой осуществимости принимается окончательное решение. Дизайн системы разбит на модули, выполняющие различные функции. Это также называется проектом высокого уровня (HLD).

Передача данных и связь между внутренними модулями и с внешним миром (другими системами) четко понимаются и определяются на этом этапе. С помощью этой информации на этом этапе могут быть разработаны и задокументированы интеграционные тесты.

Модульный дизайн

На этом этапе определяется подробный внутренний дизайн для всех модулей системы, называемый проектированием низкого уровня (LLD). Важно, чтобы конструкция была совместима с другими модулями в системной архитектуре и с другими внешними системами. Модульные тесты являются неотъемлемой частью любого процесса разработки и помогают устранить максимальное количество сбоев и ошибок на очень ранней стадии. Эти модульные тесты могут быть разработаны на данном этапе на основе конструкций внутренних модулей.

Фаза кодирования

Фактическое кодирование системных модулей, разработанных на этапе проектирования, выполняется на этапе кодирования. Выбор наиболее подходящего языка программирования определяется на основе требований к системе и архитектуре.

Кодирование выполняется на основе руководящих принципов и стандартов кодирования. Код проходит многочисленные проверки кода и оптимизируется для обеспечения максимальной производительности перед тем, как окончательная сборка будет помещена в репозиторий.

Этапы валидации

Различные фазы валидации в V-модели подробно описаны ниже.

Модульное тестирование

Модульные тесты, разработанные на этапе проектирования модуля, выполняются в коде на этом этапе проверки. Модульное тестирование — это тестирование на уровне кода, которое помогает устранить ошибки на ранней стадии, хотя все дефекты не могут быть обнаружены с помощью модульного тестирования.

Интеграционное тестирование

Интеграционное тестирование связано с этапом архитектурного проектирования. Интеграционные тесты выполняются для проверки сосуществования и взаимодействия внутренних модулей в системе.

Системное тестирование

Системное тестирование напрямую связано с этапом проектирования системы. Системные тесты проверяют всю функциональность системы и связь разрабатываемой системы с внешними системами. Большинство проблем совместимости программного и аппаратного обеспечения могут быть обнаружены во время выполнения этого системного теста.

Приемочное тестирование

Приемочное тестирование связано с фазой анализа бизнес-требований и включает тестирование продукта в пользовательской среде. Приемочные испытания выявляют проблемы совместимости с другими системами, доступными в пользовательской среде. Он также обнаруживает нефункциональные проблемы, такие как дефекты нагрузки и производительности в реальной пользовательской среде.

V-модель — применение

Приложение V-Model почти такое же, как и модель водопада, поскольку обе модели относятся к последовательному типу. Требования должны быть очень четкими до начала проекта, потому что обычно возвращаться и вносить изменения обходится дорого. Эта модель используется в области медицинских разработок, поскольку это строго дисциплинированная область.

Следующие указатели представляют собой некоторые из наиболее подходящих сценариев использования приложения V-Model.

- Требования четко определены, четко задокументированы и зафиксированы.
- Определение продукта стабильное.
- Технология не является динамичной и хорошо понимается командой проекта.

- Нет никаких неоднозначных или неопределенных требований.
- Проект короткий.

V-модель — плюсы и минусы

Преимущество метода V-модели в том, что его очень легко понять и применить. Простота этой модели также упрощает управление. Недостатком является то, что модель не гибкая для изменений, и на случай изменения требований, что очень распространено в современном динамичном мире, внесение изменений становится очень дорогостоящим.

Преимущества метода V-модели заключаются в следующем:

- Это очень дисциплинированная модель, и фазы выполняются по одной за раз.
- Хорошо подходит для небольших проектов, где требования очень хорошо понятны.
- Просто и легко понять и использовать.
- Легко управлять за счет жесткости модели. На каждом этапе есть конкретные результаты и процесс проверки.
- Недостатки метода V-Model следующие:
- Высокий риск и неопределенность.
- Не очень хорошая модель для сложных и объектно-ориентированных проектов.
- Плохая модель для длительных и текущих проектов.
- Не подходит для проектов, требования которых подвержены умеренному или высокому риску изменения.
- Когда приложение находится на стадии тестирования, трудно вернуться назад и изменить функциональность.
- Работающее программное обеспечение не производится до конца жизненного цикла.

5. SDLC — Модель большого взрыва

Модель большого взрыва — это модель SDLC, в которой мы не следуем никакому конкретному процессу. Разработка просто начинается с необходимых денег и усилий на входе, а на выходе получается разработанное программное обеспечение, которое может соответствовать или не соответствовать требованиям заказчика. Эта модель большого взрыва не следует процессу / процедуре и требует очень небольшого планирования. Даже заказчик не уверен в том, что именно он хочет, и требования выполняются на лету без особого анализа.

Обычно этой модели следуют для небольших проектов, где команды разработчиков очень малы.

Модель большого взрыва — дизайн и применение

Модель большого взрыва включает в себя сосредоточение всех возможных ресурсов на разработке и кодировании программного обеспечения при очень небольшом планировании или вообще без него. Требования понимаются

и выполняются по мере их поступления. Любые требуемые изменения могут потребовать, а могут и не потребовать обновления всего программного обеспечения.

Эта модель идеально подходит для небольших проектов с одним или двумя разработчиками, работающими вместе, а также полезна для академических или практических проектов. Это идеальная модель для продукта, требования которого не совсем понятны, и окончательная дата выпуска не указана.

Модель большого взрыва — плюсы и минусы

Преимущество этой модели Большого взрыва в том, что она очень проста и требует очень небольшого планирования или вообще не требует его. Легко управлять, никаких формальных процедур не требуется.

Однако модель большого взрыва — это модель с очень высоким риском, и изменения требований или неправильно понятые требования могут даже привести к полному отказу от проекта или его отказу от проекта. Он идеально подходит для повторяющихся или небольших проектов с минимальными рисками.

Преимущества модели большого взрыва заключаются в следующем:

- Это очень простая модель.
- Планирование практически не требуется.
- Легко управлять.
- Требуется очень мало ресурсов.
- Дает гибкость разработчикам.
- Это хорошее учебное пособие для новичков или студентов.

Недостатки модели большого взрыва:

- Очень высокий риск и неопределенность.
- Не очень хорошая модель для сложных и объектно-ориентированных проектов.
- Плохая модель для длительных и текущих проектов.
- Может оказаться очень дорогостоящим при неправильном понимании требований.

6. SDLC — гибкая модель

Модель Agile SDLC представляет собой комбинацию итеративных и инкрементных моделей процессов с акцентом на адаптируемость процесса и удовлетворение потребностей клиентов за счет быстрой доставки работающего программного продукта. Гибкие методы разбивают продукт на небольшие инкрементальные сборки. Эти сборки предоставляются в итерациях. Каждая итерация обычно длится от одной до трех недель. Каждая итерация включает в себя кросс-функциональные команды, работающие одновременно в различных областях, таких как:

- Планирование.
- Анализ требований.
- Дизайн.

- Кодирование.
- Модульное тестирование.
- Приемочное тестирование.

В конце итерации заказчику и важным заинтересованным сторонам демонстрируется рабочий продукт.

Что такое Agile?

Модель Agile считает, что каждый проект нужно обрабатывать по-разному, а существующие методы должны быть адаптированы для наилучшего соответствия требованиям проекта. В Agile задачи разделены на временные рамки (небольшие временные рамки), чтобы предоставить определенные функции для выпуска.

Применяется итеративный подход, и после каждой итерации доставляется рабочая сборка программного обеспечения. Каждая сборка является инкрементальной с точки зрения функций; окончательная сборка содержит все функции, необходимые заказчику.

Вот графическая иллюстрация Agile-модели (рисунок 6).

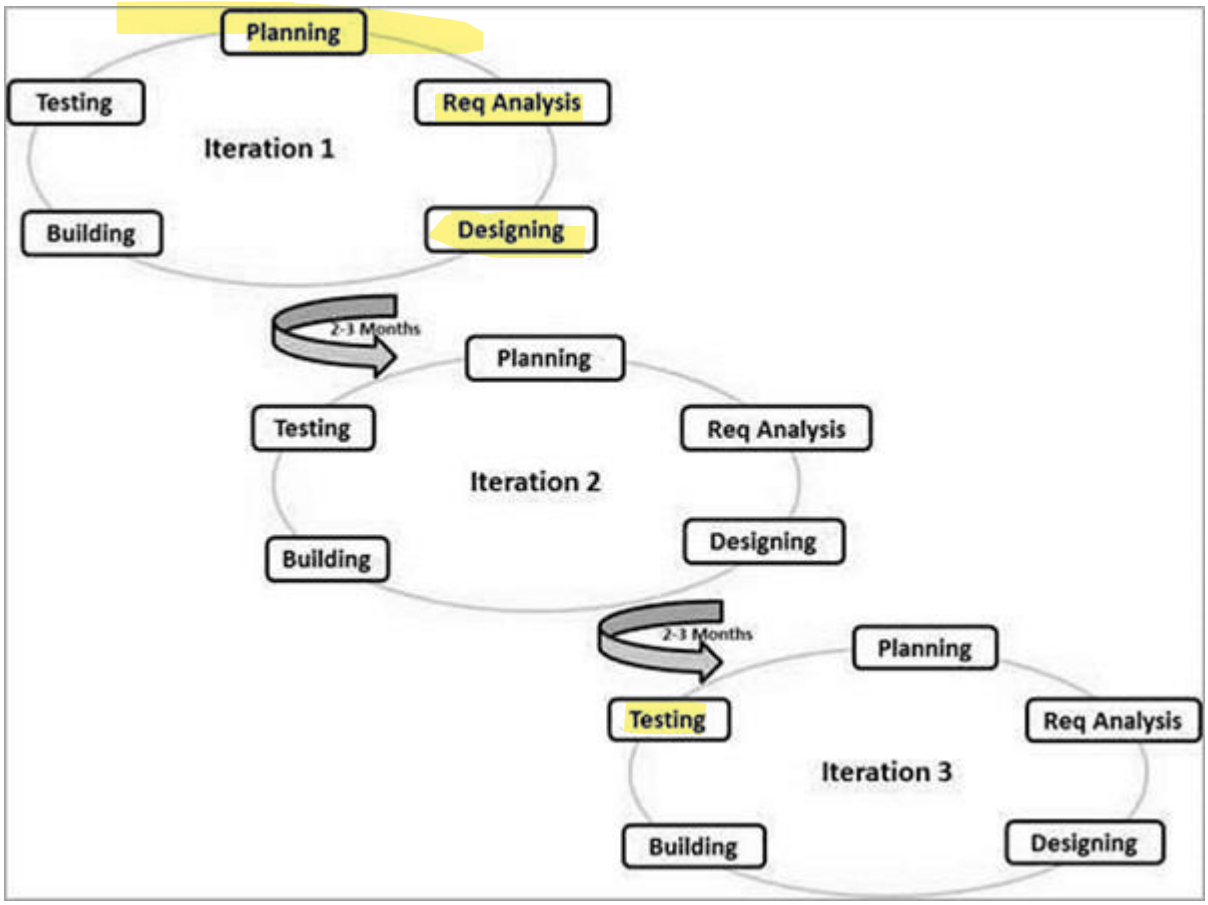

Рисунок 6 – Agile-модель

Мыслительный процесс Agile зародился на ранней стадии разработки программного обеспечения и со временем стал популярным благодаря своей гибкости и адаптируемости.

К наиболее популярным методам Agile относятся Rational Unified Process (1994), Scrum (1995), Crystal Clear, Extreme Programming (1996), адаптивная разработка программного обеспечения, разработка на основе функций и метод разработки динамических систем (DSDM) (1995). Теперь они все вместе называются Agile-методологиями после того, как в 2001 году был опубликован Agile Manifesto.

Ниже приведены принципы Agile Manifesto:

- Индивидуумы и взаимодействия. В гибкой разработке важны самоорганизация и мотивация, а также такие взаимодействия, как совместное размещение и парное программирование.
- Рабочее программное обеспечение - демонстрационное рабочее программное обеспечение считается лучшим средством связи с клиентами для понимания их требований, а не просто зависимостью от документации.
- Сотрудничество с клиентами - поскольку требования не могут быть полностью собраны в начале проекта из-за различных факторов, постоянное взаимодействие с клиентами очень важно для получения надлежащих требований к продукту.
- Реагирование на изменения - гибкая разработка ориентирована на быстрое реагирование на изменения и непрерывное развитие.

Гибкие модели против традиционных SDLC

Agile основан на адаптивных методах разработки программного обеспечения, тогда как традиционные модели SDLC, такие как модель водопада, основаны на прогнозном подходе. Группы прогнозирования в традиционных моделях SDLC обычно работают с подробным планированием и имеют полный прогноз точных задач и функций, которые будут реализованы в следующие несколько месяцев или в течение жизненного цикла продукта.

Методы прогнозирования полностью зависят от анализа требований и планирования, выполняемого в начале цикла. Любые вносимые изменения проходят строгий контроль изменений и расстановку приоритетов.

Agile использует адаптивный подход, при котором нет подробного планирования и есть ясность в отношении будущих задач только в отношении того, какие функции необходимо разработать. Существует функциональная разработка, и команда динамично адаптируется к меняющимся требованиям к продукту. Продукт тестируется очень часто, через итерации выпуска, что сводит к минимуму риск серьезных сбоев в будущем.

Взаимодействие с клиентами является основой этой гибкой методологии, а открытое общение с минимумом документации — типичные особенности среды гибкой разработки. Гибкие команды работают в тесном сотрудничестве друг с другом и чаще всего находятся в одном географическом месте.

Гибкая модель — плюсы и минусы

В последнее время в мире программного обеспечения широко применяются гибкие методы. Однако не всегда этот метод подходит для всех продуктов. Вот некоторые плюсы и минусы модели Agile.

Преимущества гибкой модели заключаются в следующем:

- Это очень реалистичный подход к разработке программного обеспечения.
- Способствует совместной работе и перекрестному обучению.
- Функциональность можно быстро развить и продемонстрировать.
- Требования к ресурсам минимальны.
- Подходит для фиксированных или изменяющихся требований
- Предоставляет ранние частичные рабочие решения.
- Хорошая модель для постоянно меняющейся среды.
- Минимальные правила, легко использовать документацию.
- Обеспечивает параллельную разработку и поставку в рамках общего запланированного контекста.
- Планирование практически не требуется.
- Легко управлять.
- Предоставляет разработчикам гибкость.

Недостатки Agile Model следующие:

- Не подходит для обработки сложных зависимостей.
- Повышенный риск устойчивости, ремонтопригодности и расширяемости.
- Общий план, гибкий лидер и гибкая практика PM — необходимость, без которой ничего не получится.
- Строгое управление доставкой диктует объем, предоставляемые функциональные возможности и корректировки для соблюдения сроков.
- В значительной степени зависит от взаимодействия с клиентом, поэтому, если клиент не ясен, команду можно направить в неверном направлении.
- Существует очень высокая индивидуальная зависимость, поскольку создается минимум документации.
- Передача технологий новым членам команды может быть довольно сложной задачей из-за отсутствия документации.

STLC — жизненный цикл тестирования программного обеспечения (Software Testing Life Cycle)

STLC означает жизненный цикл тестирования программного обеспечения. STLC — это последовательность различных действий, выполняемых группой тестирования для обеспечения качества программного обеспечения или продукта.

STLC является неотъемлемой частью жизненного цикла разработки программного обеспечения (SDLC). Но STLC имеет дело только с этапами тестирования.

STLC запускается, как только требования определены или SRD (документ с требованиями к программному обеспечению) передается заинтересованным сторонам.

STLC предоставляет пошаговый процесс для обеспечения качества программного обеспечения.

На ранней стадии STLC, пока разрабатывается программное обеспечение или продукт, тестировщик может анализировать и определять объем тестирования, критерии входа и выхода, а также тестовые случаи. Это помогает сократить время цикла тестирования наряду с лучшим качеством.

Как только этап разработки закончен, тестировщики готовы к тестированию и начинают с выполнения. Это помогает находить ошибки на начальном этапе.

Фазы STLC

STLC имеет следующие различные этапы, но не обязательно выполнять все этапы. Этапы зависят от природы программного обеспечения или продукта, времени и ресурсов, выделенных для тестирования, и модели SDLC, которая должна соблюдаться (рисунок 7).

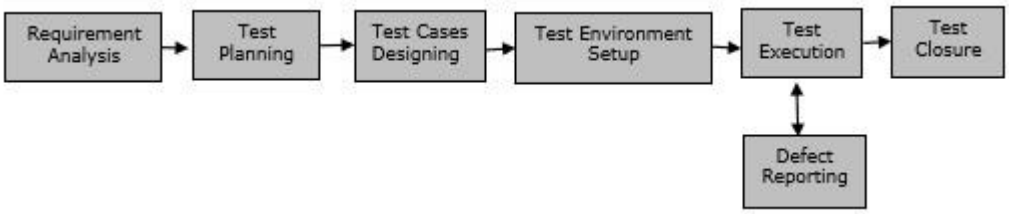

Рисунок 7 – Этапы STLC

Есть 6 основных этапов STLC:

1. Анализ требований — Когда SRD готов и представлен заинтересованным сторонам, группа тестирования начинает анализ высокого уровня, касающийся AUT (тестируемого приложения).
2. Планирование тестирования — Команда тестирования планирует стратегию и подход.
3. Проектирование тест-кейсов — Разработка тест-кейсов на основе объема и критериев.
4. Настройка тестовой среды — когда интегрированная среда готова для проверки продукта.
5. Выполнение теста — проверка продукта в режиме реального времени и поиск ошибок.
6. Закрытие теста — После завершения тестирования матрица, отчеты, результаты документируются.

Анализ требований — Когда SRD готов и представлен заинтересованным сторонам, группа тестирования начинает анализ высокого уровня, касающийся AUT (тестируемого приложения).

Планирование тестирования — Команда тестирования планирует стратегию и подход.

Проектирование тест-кейсов — Разработка тест-кейсов на основе объема и критериев.

Настройка тестовой среды — когда интегрированная среда готова для проверки продукта.

Выполнение теста — проверка продукта в режиме реального времени и поиск ошибок.

Закрытие теста — После завершения тестирования матрица, отчеты, результаты документируются.

Обобщим, что сегодня прошли:

Мы с вами изучили этапы жизненного цикла разработки ПО, а также какие модели есть, преимущества и недостатки каждого. Чем важна эта тема?

SDLC важен для разработки потому, что:

- Он предлагает основу для планирования проекта, составления графиков и оценки.
- Обеспечивает основу для стандартного набора действий и результатов.
- Это механизм отслеживания и контроля проекта.
- Повышает прозрачность планирования проекта для всех заинтересованных сторон процесса разработки.
- Увеличение и повышение скорости разработки.
- Улучшение отношений с клиентами.
- Помогает снизить риск проекта и накладные расходы на планирование и управления проектом.

Про модели разработки ПО необходимо знать, чтобы понимать на каком этапе подключается тестирование, заранее знать какие узкие места есть, в каких случаях какая модель больше подходит и просто, чтобы понимать, что происходит вокруг вокруг, видеть картину целиком, иначе погфружение в любой проект будет болезненным и трудным.