«Уникальный продукт - Удобная страховка»

***SDLC (Software development lifecycle)*- Жизненный цикл разработки ПО**-это серия из шести фаз, через которые проходит любая программная система.

1. **Планирование** отвечает на вопрос «Что мы хотим сделать?»
2. **Анализ требований** отвечает на вопрос «Какие проблемы требуют решений?»
3. **Проектирование и дизайн** отвечает на вопрос «Как мы добьемся наших целей?»
4. **Разработка ПО** регулирует процесс создания продукта.
5. **Тестирование** регулирует обеспечение качественной работы продукта. **Развертывание** регулирует использование финального продукта.
6. **Поддержка** позволяет убедиться, что система не устарела и отвечает современным стандартам и технологиям.

Team members

Заказчик проекта – лицо, формулирующие требования к результатам проекта и использующее полученные результаты проекта**.**

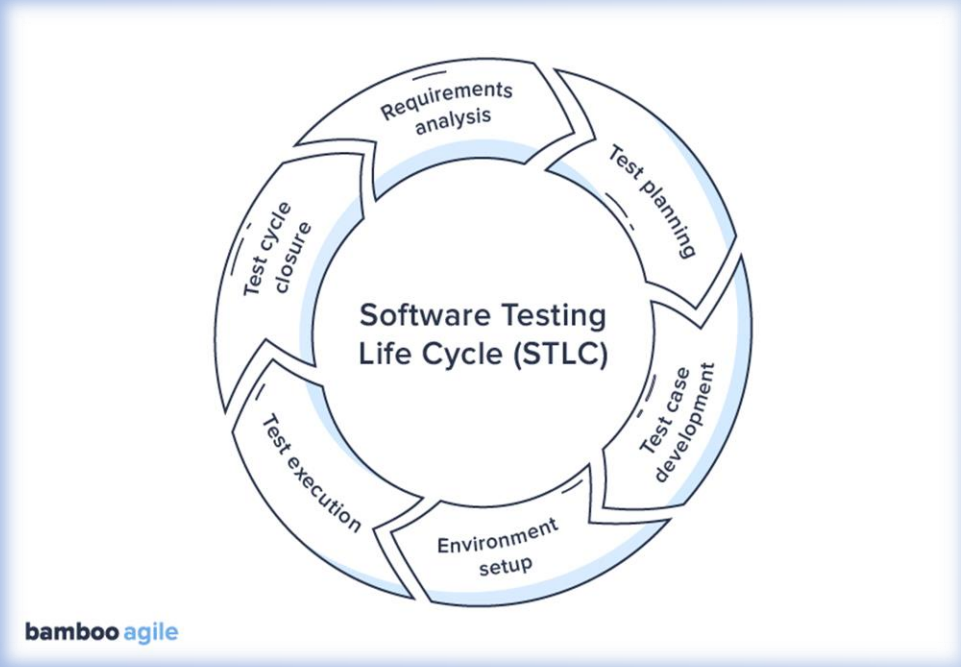
Задачи заказчика проекта:
*Описание требуемых результатов проекта.
*Контроль исполнения проекта в контрольных точках.
*Приемка промежуточных и окончательных результатов проекта.

Аналитик — понимает заказчика, его бизнес, пользователей, творчески мыслит. Аналитик отвечает за ожидания и потребности пользователей
Задачи аналитика проект:
*Исследования-собрать информацию о том, кто пользователи и что им нужно;
*Описание персонажей-выделить и описать типичные категории пользователей;
*Описание сценариев- выделить и описать типичные задачи пользователей;
*Разработка требований –проанализировать, описать и всем объяснить , что и почему нужно сделать в продукте;
Управление требованиями- вести процесс разработки вплоть до выпуска с тем, чтобы созданный продукт удовлетворял потребностям пользователей.

Разработчики-специалисты, которые пишут код, чтобы продукт работал как положено.
*Продумывает проект, возможности и функции приложения.
*Создает интерфейс (внешний вид) и настраивает внутреннюю логику.
*Пишет код на одном из языков программирования.
*Тестирует программу на всех этапах разработки, ищет и исправляет ошибки, делает обновления.
*Готовит техническую документацию для других специалистов и пользователей.

Тестировщик-это специалист, который создаёт сценарии тестирования, прогнозирует сбои и находит ошибки в продуктах.
*Изучает документацию и уточняет спорные моменты в ней**.** В результате понимает, какую функциональность продукта нужно будет тестировать.
*Разрабатывает тесты. На этом этапе тестировщик готовит набор тестов для проверки. Если в продукт вносят изменения, а их регулярно вносят, нужно скорректировать и тестовую модель.
*Проверка продукта. На этом этапе тестировщик проходит по разработанным тестам и фиксирует результат: в тех местах функциональности, где тесты прошли успешно — подтверждает, что продукт работает корректно, а там, где тесты не прошли — фиксирует ошибки.

STLC (Software Testing Lifecycle), или **жизненный цикл тестирования** — это последовательность действий, проводимых в процессе тестирования, с помощью которых гарантируется качество программного обеспечения и его соответствие требованиям.



Анализ требований (Requirement Analysis)-один из важнейших этапов, потому что именно на нем можно почти бесплатно исправить недостатки проекта. Этап анализа требований также определяет потенциальную потребность в автоматизированном тестировании и позволяет производить экономические расчеты затрат на рабочую силу на основе оценки проекта. На этом же этапе обсуждаются и документируются критерии начала и окончания тестирования.

Планирование тестирования (Test Planning): на этом этапе формируется план тестирования, т.е. мы определяем действия и ресурсы, которые помогут достичь целей тестирования (участники и их роли, инструменты, окружение).

Разработка тест-кейсов (Test Case Development): подразумевает использование ручного и автоматизированного тестирования для достижения полного охвата функциональности программного обеспечения, при этом процесс основан на заранее установленных требованиях.

Настройка тестовой среды (Test Environment Setup): в плане тестирования четко указано, какую тестовую среду следует использовать. На этом этапе STLC настраиваются операционные системы и виртуальные машины, развертываются инструменты тестирования, такие как Selenium, Katalon Studio, а также тестовая среда и базы данных проекта.

Выполнение тестов (Test Execution): тесты выполняются на основе готовой тестовой документации и правильно настроенной тестовой среды. Все результаты тестирования регистрируются в Системе управления тестированием. Отрицательно пройденные тесты, в которых фактический результат отличается от ожидаемого, регистрируются как ошибки и передаются команде разработчиков на доработку с последующей перепроверкой после исправления.

Завершение цикла испытаний (Test Cycle Closure): окончательная генерация отчетов о тестировании для клиента. Они должны включать затраченное время, процент обнаруженных ошибок и положительных результатов тестирования, общее количество обнаруженных и исправленных ошибок.

Тестирование ПО (Software testing) — проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом.

Тестирование играет жизненно важную роль в разработке программного обеспечения и является неотъемлемой частью жизненного цикла его разработки, так как:
**повышает надежность, качество и производительность программного обеспечения,*
**помогает разработчику проверить, правильно ли работает программное обеспечение, убедиться, что программное обеспечение выполняет то, для чего оно предназначено.*
**помогает понять разницу между фактическим и ожидаемым результатом, что обеспечивает качество продукта*.

Благодаря тому, что тестирование ПО становится частью программирования, разработчики имеют возможность исправлять ошибки уже на начальной стадии разработки. Это позволяет сократить риск появления дефектов в готовом продукте. Если ошибки найдены на начальном уровне, разработчик может создать надежное программное обеспечение. Таким образом, чем раньше начинается процесс, тем раньше обнаруживаются ошибки и тем ниже стоимость их исправления.

QA-тестировщик или инженер по тестированию (QA-engineer) — это специалист, который создаёт сценарии тестирования, прогнозирует сбои и находит ошибки в продуктах.

Зона ответственности тестировщика – все этапы и процессы создания продукта, начиная от помощи разработчикам и заканчивая составлением итоговых технических документов.

В его обязанности входит:

**формирование плана тестовых процессов;*

**проверка продукта с помощью методов и разных пользовательских сценариев;*

**нахождение и фиксирование багов;*

**предоставление отчета по ошибкам разработчикам;*

**анализ требований к программе;*

**работа с документами (составление чек-листов, планов и кейсов);*

проведение финального тестирования после всех исправлений.

Методологии разработки ПО

1) **Waterfall (каскадная модель, или «водопад»)**
В этой модели разработка осуществляется поэтапно: каждая следующая стадия начинается только после того, как заканчивается предыдущая.

Плюсы	Минусы
Простая в использовании модель.	С этой моделью слишком сложно и дорого адаптироваться к изменениям требований.
Каждый этап хорошо задокументирован.	Документирование каждой фазы проекта занимает много времени.
Результат проекта абсолютно предсказуем.	Вы не можете ничего предоставить заказчику, пока не завершите весь проект.
Этапы и роли четко определены с самого начала.	Различные команды (дизайн, разработка, контроль качества и т. д.) изолированы, а взаимодействие между ними ограничено.
Минимальное вмешательство клиента.	Без обратной связи клиента результат рискует не оправдать ожидания.

2) **V-образная модель (разработка через тестирование)**
Это усовершенствованная каскадная модель, в которой заказчик с командой программистов одновременно составляют требования к системе и описывают, как будут тестировать её на каждом этапе.

Плюсы	Минусы
Легко реализовывать	В V-образной модели внести изменения в середине проекта крайне сложно.
Тест-кейсы создаются заранее	При таком большом количестве процедур тестирования остается меньше времени на код.
У каждого этапа есть свои результаты, и все хорошо задокументировано.	Модель не подходит для проектов с быстро меняющимися требованиями.
Это структурированный подход с четко определенными ролями и функциями.	Не подходит для больших и сложных проектов.

3) **«Incremental Model» (инкрементная модель)**

В инкрементной модели полные требования к системе делятся на различные сборки. Терминология часто используется для описания поэтапной сборки ПО. Имеют место несколько циклов разработки, и вместе они составляют жизненный цикл «мульти-водопад». Цикл разделен на более мелкие легко создаваемые модули. Каждый модуль проходит через фазы определения требований, проектирования, кодирования, внедрения и тестирования. Процедура разработки по инкрементной модели предполагает выпуск на первом большом этапе продукта в базовой функциональности, а затем уже последовательное добавление новых функций, так называемых «инкрементов». Процесс продолжается до тех пор, пока не будет создана полная система.

Плюсы	Минусы
Не нужно вкладывать много денег на начальном этапе	Разработчики будут оттягивать доработку основной функциональности и «пилить мелочёвку»
Можно быстро получить фидбэк от пользователей и оперативно обновить техническое задание	Каждая команда программистов разрабатывает свою функциональность и может реализовать интерфейс продукта по-своему.
Ошибка обходится дешевле	

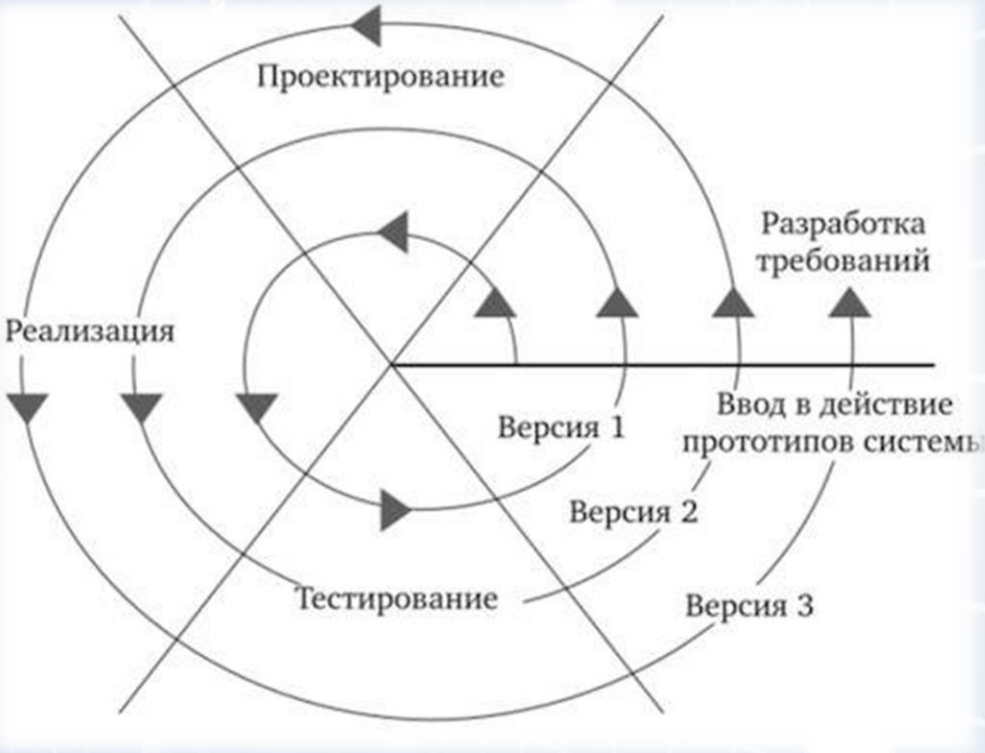
4) **Iterative Model (итеративная модель)**
Это модель, при которой заказчик не обязан понимать, какой продукт хочет получить в итоге, и может не прописывать сразу подробное техзадание.

Плюсы	Минусы
Быстрый выпуск минимального продукта даёт возможность оперативно получать обратную связь от заказчика и пользователей.	Использование на начальном этапе баз данных или серверов
Постоянное тестирование пользователями позволяет быстро обнаруживать и устранять ошибки.	Отсутствие фиксированного бюджета и сроков.

5) **Spiral Model (спиральная модель)**

Используя эту модель, заказчик и команда разработчиков серьёзно анализируют риски проекта и выполняют его итерациями. Последующая стадия основывается на предыдущей, а в конце каждого витка — цикла итераций — принимается решение, продолжать ли проект.

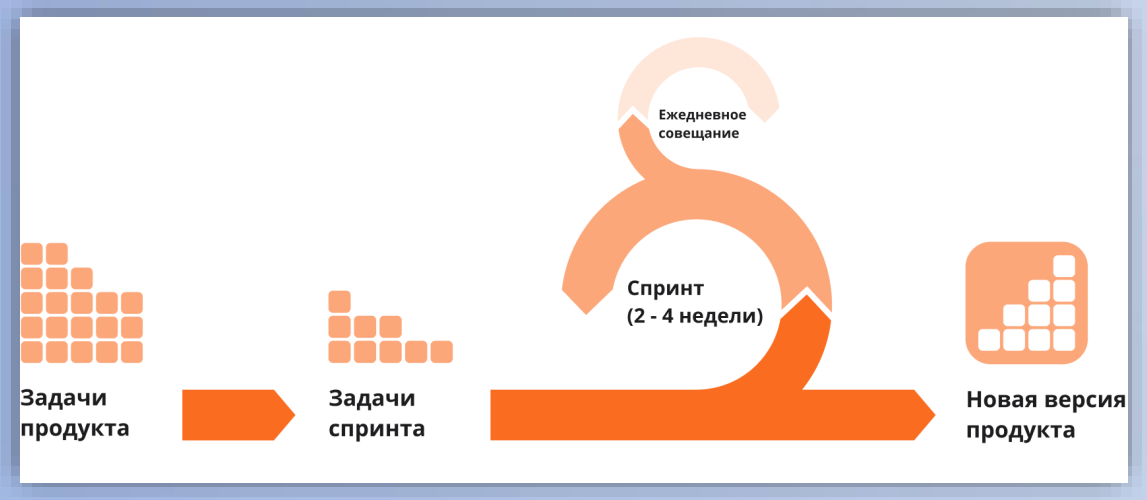
Плюсы	Минусы
Анализ рисков на каждой итерации увеличивает шансы проекта на успех.	Требуется опыт управления рисками.
Этот метод позволяет создавать стабильные и надёжные системы, поскольку они тщательно тестируются в каждом цикле.	Модель подразумевает работу с большим объёмом документации.
Можно менять требования между циклами.	Нельзя изменить требования в середине цикла.
Раннее вовлечение разработчиков помогает согласовать бизнес-требования и технические возможности.	Каждый кружок в спирали развития представляет собой «мини-каскад», а это значит, что вы не можете пропускать фазы.
Частые выпуски позволяют получать регулярную обратную связь от клиентов даже на ранних этапах цикла.	Поскольку ограничений на количество итераций нет, сложно предсказать, сколько кругов потребуется для создания окончательной версии продукта.



«Agile Model»
(или гибкая модель разработки ПО)
В такой модели все этапы жизненного цикла бывают выполнены в течение одной итерации и готовы к внесению любых изменений. То есть ваш проект делится на спринты – отрезки времени, за которые должен быть получен результат, обычно от одной до четырех недель. В каждом спринте есть свой список задач, который должен быть выполнен к концу итерации, каждая из задач имеет свой уровень оценки.

Agile Model имеет два отдельных гибких подхода:

1) ***Scrum***
Скрам-проекты разбиты на спринты. Спринт – это небольшой объём работы, который необходимо выполнить в течение определённого периода времени. Обычно заказчику доставляется часть проекта, которая была завершена во время спринта (инкремент продукта, от англ. **increment**). Скрам подразумевает активное общение и сотрудничество между всеми участниками проекта.



Плюсы	Минусы
Эффективное взаимодействие между участниками проекта.	Высокая стоимость разработчиков.
Минимум контроля и фокус на постоянные обновления.	Нежелательное расширение проекта.
	Недостаток гибкости в больших проектах.

2) **Kanban**

Сегодня это одна из наиболее популярных методологий разработки ПО. Команда ведёт работу с помощью виртуальной доски, которая разбита на этапы проекта. Каждый участник видит, какие задачи находятся в работе, какие — застряли на одном из этапов, а какие уже дошли до его столбца и требуют внимания.

В отличие от скрама, в канбане можно взять срочные задачи в разработку сразу, не дожидаясь начала следующего спринта.

Плюсы	Минусы
Высокая концентрация на текущей работе.	Не удовлетворяет требованиям больших команд.
Быстрое устранение проблем.	
Оптимизация издержек.	

ЦЕРЕМОНИИ / РИТУАЛЫ AGILE

Название	Цель	Периодичность	Продолжительность
Stand up / Daily Meeting / Ежедневный митинг	Быстро информировать всех о том, что происходит в команде. Команда разработки анализирует сделанное за последние сутки, планирует свою работу на ближайшие 24 часа.	Ежедневно	Не более 15 минут. Проводится стоя, т.к. это ускоряет встречу. Обычно в начале рабочего дня.
Планирование спринта	Определить задачи, над которыми будет трудиться команда разработки во время спринта. План создается совместными усилиями всей скрам-команды.	1 раз в спринт, в 1-ый день спринта перед стартом работы над инкрементом.	Для спринта длительностью 1 месяц - не более 8 часов. Если спринт короче, Планирование проводится быстрее. Чаще планирование занимает не более 4 часов, а для спринта длиной в 2 недели, не более 2-х часов.
Обзор спринта / Ревью	Инспекция инкремента и, по необходимости, адаптация бэклога продукта. Скрам-команда и заинтересованные лица совместно обсуждают, что было сделано за спринт. Это не статусная встреча, а неформальная. На ней проводится демонстрация инкремента продукта для получения обратной связи и развития сотрудничества.	1 раз в конце каждого спринта	Для спринта длительностью 1 месяц - не более 4 часов. Чем короче спринт, тем короче его Обзор.
Ретроспектива / Ретро	Провести анализ работы команды в спринте, создать план улучшений командной работы в следующем спринте. Ретроспективы помогают команде понять, что работает хорошо, а что нет.	1 раз в завершении каждого спринта после проведения Обзора спринта, перед Планированием следующего спринта.	Для спринта длительностью 1 месяц - не более 3 часов. Если спринт короче, Ретроспектива проводится быстрее.

Название	Цель	Периодичность	Продолжительность
Демо (демонстрация)	Продемонстрировать проделанную работу - промежуточный результат, получить оценку своей работы от заинтересованных лиц.	Чаще, чем обзор спринта. Каждая команда устанавливает для себя периодичность сама. Например: 1 раз в неделю (в четверг или пятницу) или 1 раза в 2 недели.	1 - 2 часа
Груминг	Актуализировать бэклог проекта. Провести ревизию пользовательских историй в бэклоге проекта: написать новые, удалить неактуальные,	1 раз в спринт. Не позднее, чем за 2 дня до планирования нового спринта	Не более 10 - 15% рабочего времени разработчиков.

Роли в Agile командах

Product Owner (владелец продукта)

Владелец продукта в направлении agile-менеджмента scrum представляет интересы Stakeholders (заинтересованных сторон). Он отвечает за то, в каком направлении будет развиваться продукт и как в целом должен реализовываться проект. Владелец продукта должен понимать требования Stakeholders к разрабатываемому софту и уметь донести их до команды разработки. Он также обязан обладать долгосрочным видением развития бизнеса и руководить своим проектов в соответствии с ожиданием Stakeholders .

Эта роль подразумевает управление бэклогом, резервом проекта. Резерв проекта – список требований к функциональности продукта (ПО), упорядоченный по степени важности. Также владелец продукта добавляет новые пункты в бэклог и определяет, что должно быть реализовано первым. Он может изменять требования в бэклоге и определяет их приоритетность на основе обстоятельств проекта и обратной связи от стейкхолдеров.

Scrum-мастер

Scrum-мастер в направлении agile-менеджмента scrum руководит командой по разработке продукта, координирует её членов при реализации проектов. Он получает инструкции по дальнейшему изменению софта от владельца продукта и обеспечивает, что разработка ведётся в соответствии с ними. Обязанности scrum-мастера могут подразумевать руководство ежедневной работой при реализации различных scrum-инициатив и выполнением спринтов.

Также среди обязанностей этого сотрудника – административные задачи, такие как проведение встреч, организация сотрудничества и решение проблем, которые мешают реализации проекта. Кроме того, он должен огородить членов команды от внешнего вмешательства в их работу.

Team Members (члены команды разработки)

Эта роль в направлении scrum agile-менеджмента подразумевает ответственность непосредственно за разработку продукта, но этим не исчерпывается. Сотрудники в этой команде должны брать на себя кросс-функциональные обязанности по трансформации требований к продукту и возникающих идей в реальные функции программного обеспечения. В команде должны быть сотрудники с компетенциями разработчиков, технических писателей, программистов, тестировщиков и UX-специалистов.

Stakeholders (заинтересованные стороны)

Stakeholders принято называть тех, кто напрямую не принимает участия в процессе разработки продукта, но влияет на принятие решений и работу команды разработчиков.

Stakeholders могут быть конечные пользователи продукта, руководство, сотрудники технической или клиентской поддержки, инвесторы, внешние аудиторы или члены других scrum-команд, которые работают над смежными проектами. Вклад Stakeholders имеет большое значение для развития проекта. Они помогают привести продукт в соответствие с бизнес-целями компании, ожиданиями конечных пользователей и решить проблемы, с которыми сталкиваются разработчики.

Пользователь — конечный целевой покупатель продукта