«УДОБНАЯ СТРАХОВКА» ОТ СБЕРБАНКА

УПРАВЛЯЙ ОН-ЛАЙН

СБЕР

«Удобная страховка» -
- комфорт и защита в несколько «кликов».

- Защита на время болезни и/или потери работы
- можно получить выплату для погашения кредита или решения финансовых проблем*
- Открывайте онлайн.
- Доступна в браузерной версии и мобильном приложении СберБанк Онлайн
- Расчёт платежа - на первом шаге оформления услуги
- Закрывайте, когда услуга больше не нужна: отключить автоплатёж в приложении СберБанк Онлайн.
- ** В зависимости от страхового случая*

Оформляйте страховку в мобильном приложении СберБанк Онлайн или на сайте:

→ «Страхование» → «Удобная страховка» → выбрать нужную программу страхования.

СБЕР

Блок 1. SDLC - Жизненный цикл разработки приложения.

SDLC - последовательность действий, которые необходимо выполнить, чтобы получить готовое решение.

- **1. Планирование и анализ потребностей.** Владелец продукта: что нужно сделать. Руководитель проекта: как это сделать». Бизнес-аналитик: понять потребности и перевести их в бизнес-требования.
- **2. *Анализ*. *О***пределение и документирование требований в виде ТЗ на разработку ПО и/или спецификации. Основная рабочая обязанность бизнес-аналитика.
- **3. Проектирование дизайна и архитектуры ПО.** UI/UX-дизайн. Участвуют IT-архитектор и дизайнеры, системный аналитик. Возможен системный и бизнес-аналитики в одном лице.
- **4. Создание или разработка продукта.** Непосредственная реализация всех запланированных требований. Участники: программисты и разработчики ПО.
- **5. *Тестирование и развертывание*. *Т***естировщики проверяют результат от разработчиков на соответствие требованиям и отсутствие ошибок. DevOps-инженеры и администраторы разворачивают продукт в реальной среде эксплуатации (production).
- **6. Поддержка и сопровождение. Р**аботающий продукт сопровождают администраторы и специалисты технической поддержки. Технические писатели создают руководства пользователя, администратора и прочую программную документацию, предусмотренную контрактом.

СБЕР

Блок 2. Team members.

Developer: производство программных алгоритмов; не несёт ответственности за применение результата и по финансовым рискам; данная роль часто сегментируется по разделению ответственности.

Back-end developer: разработчик программно-аппаратной части комплексного ПО;

Front-end developer: разработчик клиентской стороны пользовательского интерфейса к программно-аппаратной части.

User Experience Designer (UX): производство карт пользовательского опыта; несёт ответственность за применение результата, не несёт ответственность по финансовым рискам; выявляет у ПО простоту использования, восприятие ценности, полезность, эффективность. Продумывает и оценивает процессы и сценарии использования.

User Interface Designer (UI): производство графической составляющей интерфейсов; не несёт ответственности за применение результата и по финансовым рискам; разрабатывает визуальную часть пользовательского интерфейса. Основные цели UI дизайнера - интуитивность восприятия, простота, юзабильность и эстетика интерфейса.

Quality Assurance (QA): проверка результата; не несёт ответственности за применение результата и по финансовым рискам; умеет составлять тест план, готовить отчёты.

Team Leader: отвечает за работу группы специалистов; несёт ответственность за командную работу, не несёт финансовых рисков; обеспечивает комфортные условия работы коллектива, поддерживает эффективность; отвечает за выбор технических решений; несёт частичную ответственности за результат.

Scrum Master: отвечает за правильное применение гибкой методологии; несёт частичную ответственность за результат, но не несёт ответственность по финансовым рискам.

Project Manager (PjM): отвечает за старт, ведение и сдачу проектных работ; несёт полную ответственности за результат, частичную ответственность по финансовым рискам; работа над проектом начинается с Project Manager`а, ведётся контролируется и сдаётся им.

СБЕР

Блок 3. STLC - Жизненный цикл тестирования приложения.

STLC - метод, который позволяет тщательно протестировать разработанное ПО, убедиться, что все инициативы эффективны, надежны и полезны. Включает 5 этапов:

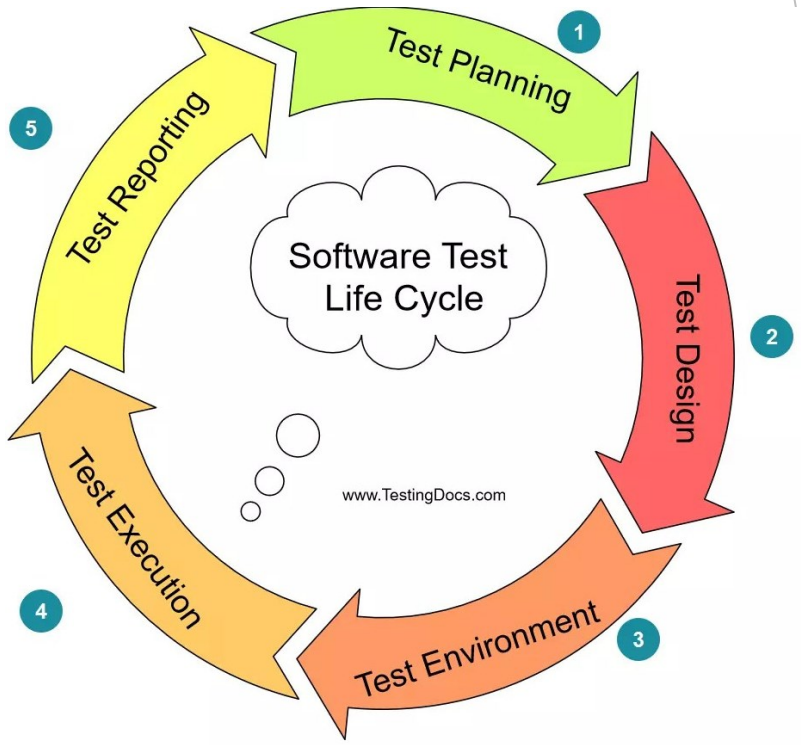
1. Анализ требований. Группа тестирования изучает документ бизнес-требований, чтобы определить важные результаты и возможности.

2. Подготовка к тестированию. Команда тестирования планирует, как внедрять тесты, создает стратегию тестирования со включением ручных, автоматических, интеграционных и модульных тестов.

3. Разработка тестов. После определения высокоуровневых тестовых примеров и методологий может начаться работа по дополнению тестовых сценариев.

4. Настройка тестовой среды. Команда приступает к созданию подходящей тестовой среды после установки работоспособной версии системы.

5. Выполнение и закрытие тестов. Команда выполняет подготовленные сценарии и сообщает о результатах проектной команде. Этот и другие шаги могут повторяться для проверки результатов.

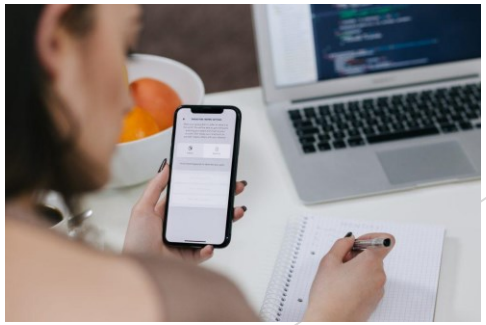


СБЕР

Блок 4. Тестирование и его значение в процессе создания программного обеспечения.

- Созданный продукт необходимо протестировать - провести проверку соответствия между реальным и ожидаемым поведением программы.
- **Тестирование** осуществляемая на конечном наборе тестов, выбранном определенным образом.
- Проводится анализ и испытания программного продукта, а, также, сопутствующей документации с целью **выявления дефектов и повышения качества** продукта.
- В более широком смысле, тестирование — это одна из техник контроля качества, включающая в себя активности по планированию работ (Test Management), проектированию тестов (Test Design), выполнению тестирования (Test Execution) и анализу полученных результатов (Test Analysis).
- Тестированием программы занимаются специалисты по контролю качества программного обеспечения — **QA-инженеры**.
- У них есть разные специализации: тестировщики баз данных, специалисты по автоматизированному тестированию, аналитики, разработчики тестов, специалисты по безопасности приложений и другие.
- Если проект большой, над ним работает целая команда: одни тестировщики готовят тесты, другие проверяют их полноту и логику, третьи занимаются непосредственно тестированием.

QA-инженер выстраивает правильный процесс создания ПО: создаёт сценарии тестирования, тесты проверки работоспособности веб- и мобильных приложений, сервисов, API, находит ошибки в продуктах. Отвечает за то, что выпускаемая программа может пережить все, даже если пользователь пытается применить ее не по назначению.



СБЕР

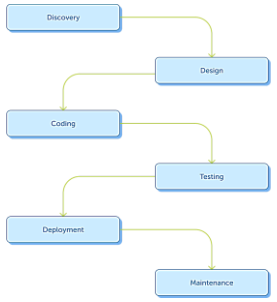
Блок 5. QA-инженер/Тестировщик: роли, обязанности, основные задачи.

Тестировщик	QA-инженер
Тестировщик ПО – проверяет программное обеспечение, выявляет ошибки, помогает улучшать продукт. Работает с продуктом как с результатом. Определяет работоспособность и оценивает качество разрабатываемого программного обеспечения путем проверки его соответствия заявленным требованиям. Тестирует продукт, описывает риски, оформляет в багтрекинговой системе и передаёт разработчикам дя устранения ошибок. Тестировщик ПО не участвует в полном цикле разработки, не может дать какие-либо рекомендации разработчикам и бизнесу. На нём не лежит ответственность за конечный результат.	QA-инженер работает с продуктом, который находится в процессе создания. Основная задача — не допустить несоответствия продукта требованиям и постараться довести как можно меньше багов до этапа тестирования. Инженер по QA не только проводит тестирование, но в некоторых случаях дает рекомендации по исправлению багов. Участвует на всех этапах разработки — от планирования до выпуска. Рекомендует бизнесу и разработчикам, что нужно сделать, чтобы повысить качество продукта. Специалисты QA устанавливают стандарты качества, выбирают инструменты и методики предотвращения ошибок, решают, как произвести и усовершенствовать процесс. Способны оценивать качество продукта, искать способы его повышения, оценивают риски. Возможно, участие в устранении багов и общение с разработчиками и заказчиками.

СБЕР

Блок 6. Традиционные методологии разработки программного обеспечения приложения (SDLC).

Водопад



Плюсы

Простая в использовании модель.

Каждый этап хорошо задокументирован.

Результат проекта абсолютно предсказуем.

Этапы и роли четко определены

Минимальное вмешательство клиента.

Минусы

Сложно и дорого адаптироваться к изменениям требований.

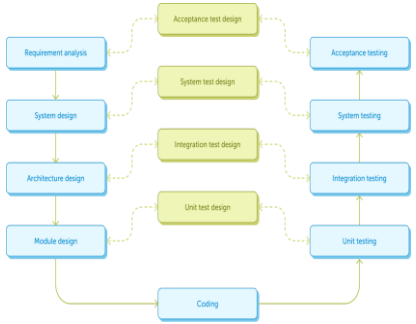
Документирование каждой фазы занимает много времени.

Предоставляется заказчику только после завершения проекта.

Команды разработчиков изолированы, взаимодействие ограничено.

Без связи с клиентом -риск не оправдать ожидания.

V-образная модель



Плюсы

Легко реализовывать.

Тест-кейсы создаются заранее.

Бюджет и продолжительность предсказуемы.

У каждого этапа свои результаты, все задокументировано.

Структурированный подход с определенными функциями.

Минусы

Внести изменения в середине проекта крайне сложно.

Остается меньше времени на код.

По сравнению с каскадной требует больше специалистов.

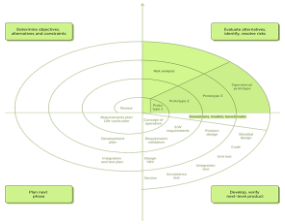
Не подходит для проектов с меняющимися требованиями.

Не подходит для больших и сложных проектов.

СБЕР

Традиционные методологии разработки программного обеспечения приложения (SDLC).

Спиральная модель



Плюсы

Анализ рисков каждой итерации увеличивает шансы проекта на успех.

Позволяет создавать стабильные и надёжные системы.

Можно менять требования между циклами.

Раннее вовлечение разработчиков помогает согласовать бизнес-требования и технические возможности.

Регулярная обратная связь от клиентов.

Минусы

Требуется опыт управления рисками.

Большой объём документации.

Нельзя изменить требования в середине цикла.

Нельзя пропускать фазы.

Неизвестно, сколько кругов потребуется для окончательной версии.

Итерационная

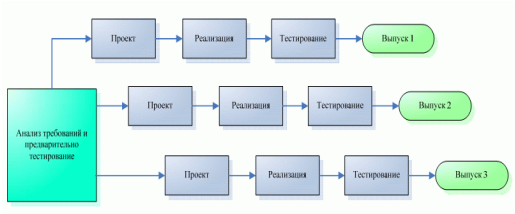
Плюсы

На этапе планирования и анализа описываются только базовые требования к продукту. Можно быстро выпустить на рынок, протестировать на реальных пользователях, вести изменения. Подходит для больших проектов с неопределёнными требованиями и инноваций.

Минусы

время жизни каждого этапа растягивается на весь период разработки
следствие большого числа итераций возникают рассогласования выполнения проектных решений и документации
запутанность архитектуры
трудности использования проектной документации на стадиях внедрения и эксплуатации вызывают необходимость перепроектирования всей системы

Инкрементная



Плюсы

Разработка тоже выполняется частями. Полный список требований имеется и воплощается частями – релизами, каждый из которых полностью проходит все этапы SDLC. Подходит для проектов с полным набором требований к результату и быстрым выводом продукта на рынок.

Минусы

Не предусмотрены итерации в рамках каждого инкремента. Определение полной функциональной системы должно осуществляться в начале жизненного цикла. Создание некоторых модулей будет завершено значительно раньше других - необходимость в четко определенных интерфейсах. формальный критический анализ и проверку для инкрементов выполнить сложнее.

СБЕР

Гибкая методология разработки программного обеспечения приложения (SDLC) Agile.

Agile — итеративная модель разработки. Программное обеспечение создается инкрементально *(incremental - постепенный)* с самого начала, в отличии от каскадных моделей, где код доставляется в конце рабочего цикла.

Основа — разбиение проектов на маленькие рабочие кусочки - пользовательские истории. Согласно приоритетности задачи решают в рамках двухнедельных циклов (итераций).

4 главные идеи Agile-манифеста разработки программного обеспечения:

Для Клиента

- Удовлетворение потребностей
- Позитивный опыт взаимодействия
- Возможность влиять на развитие любимого продукта
- Постоянное добавление нового функционала и устранение багов

Ключевой момент: следование принципам Agile не отменяет требования к управлению качеством, при четких правилах количественного управление — фиксацию и документирование требований и процедур, необходимость прохождения этапов, чек-листов и т. д.

С гибкой методологией разработки программного обеспечения небольшие проектные команды добиваются **максимальной эффективности.**

В целом, Agile кажется именно тем, что нужно большинству проектов во времена неопределённости. Более 70% компаний применяют Agile, включая Microsoft, IBM, Procter & Gamble и другие.

Мы будем придерживаться гибкой методологии разработки.

Agile реализуется через другие гибкие методы: Scrum, Kanban, XP, Lean и т.д.

Scrum, Kanban: **автономные команды из 5—9 человек. Нет формального руководителя, никто извне не диктует, как организовывать работу над продуктом.**

СБЕР

Блок 7. Церемонии наиболее универсальных и распространенных Agile методологий - Scrum и Kanban.

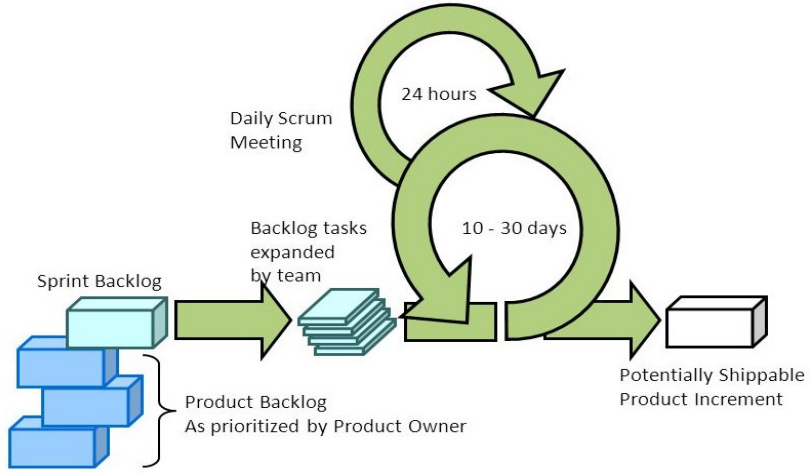
Scrum

Гибкость и ориентированность на клиента с его непосредственным участием в процессе. Не требует внедрения дорогостоящих инструментов. Пошаговая сдача проекта - минимизации рисков. Возможность быстрее показывать клиенту продукт и получать обратную связь. Больше подходит для малых и средних проектов.

Работа ведется короткими циклами — спринтами при постоянной связи с заказчиком. Команда — около 7 чел. Самоорганизация, все участники равны.

Команда и роли: Владелец продукта представляет интересы клиента, управляет бэклогом продукта и помогает определить приоритеты для команды разработчиков. Scrum-мастер следит, чтобы команда соблюдала принципы scrum. Команда разработчиков выбирает, какую работу нужно сделать, поставляет инкременты и несет коллективную ответственность.

Ограничения: не подразумевает наличие фиксированного бюджета и фиксированного тех. задания; не всегда можно адаптировать под сферу деятельности; требует регулярной коммуникации с заказчиком



Scrum - вид командной игры, позволяющий завладеть мячом и вести его дальше по полю. Слаженность, единство намерений, четкое понимание цели.

СБЕР

Церемонии наиболее универсальных и распространенных Agile методологий - Scrum и Kanban.

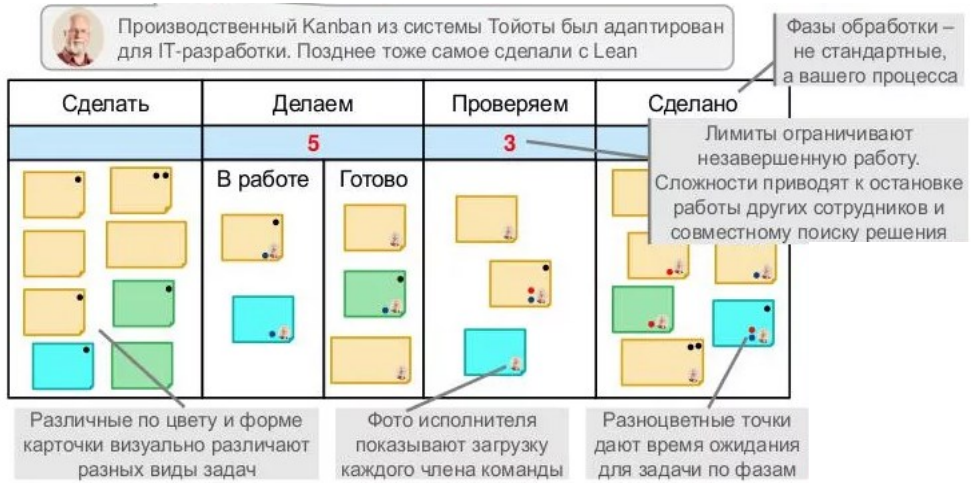
Kanban

Способ получать готовый качественный продукт вовремя. Обсуждение производительности в режиме реального времени, полная прозрачность рабочих процессов. Рабочие задачи на доске Kanban позволяют команде видеть состояние каждой задачи в любой момент времени.

План задач по приоритету меняется по необходимости. Количество задач спринта ограничено. Можно менять лимит на кол-во задач. Хорошо работает в стартапах.

Kanban-команда: концентрируется только на текущей работе., нет ролей как таковых. По завершении рабочей задачи команда забирает следующую задачу с верха бэклога. Владелец продукта может менять приоритет задач в бэклоге, не мешая работе команды, поскольку изменения происходят за пределами текущих рабочих задач.

Ограничения: плохо работает с большими командами (более 5 человек); плохо работает с кросс-функциональными командами - тяжело совместить тестирование и разработку в одной команде; не предназначен для долгосрочного планирования.



Kanban – система организации производства и снабжения, реализующая принцип «точно в срок». На японском означает «рекламный щит, вывеска».

СБЕР

«УДОБНАЯ СТРАХОВКА» -
- комфорт и защита в несколько «кликов».

УПРАВЛЯЙ ОН-ЛАЙН

СБЕР