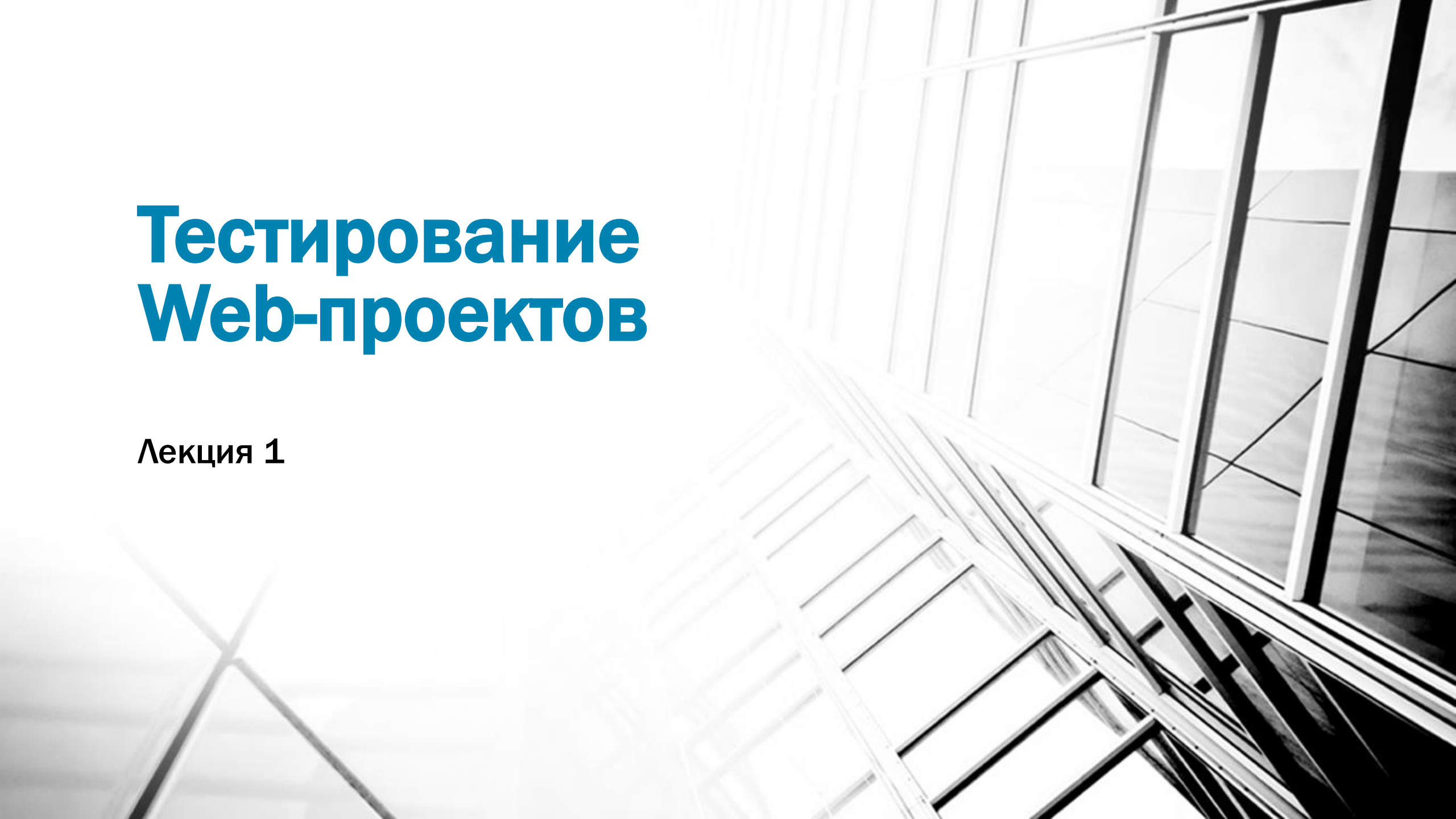
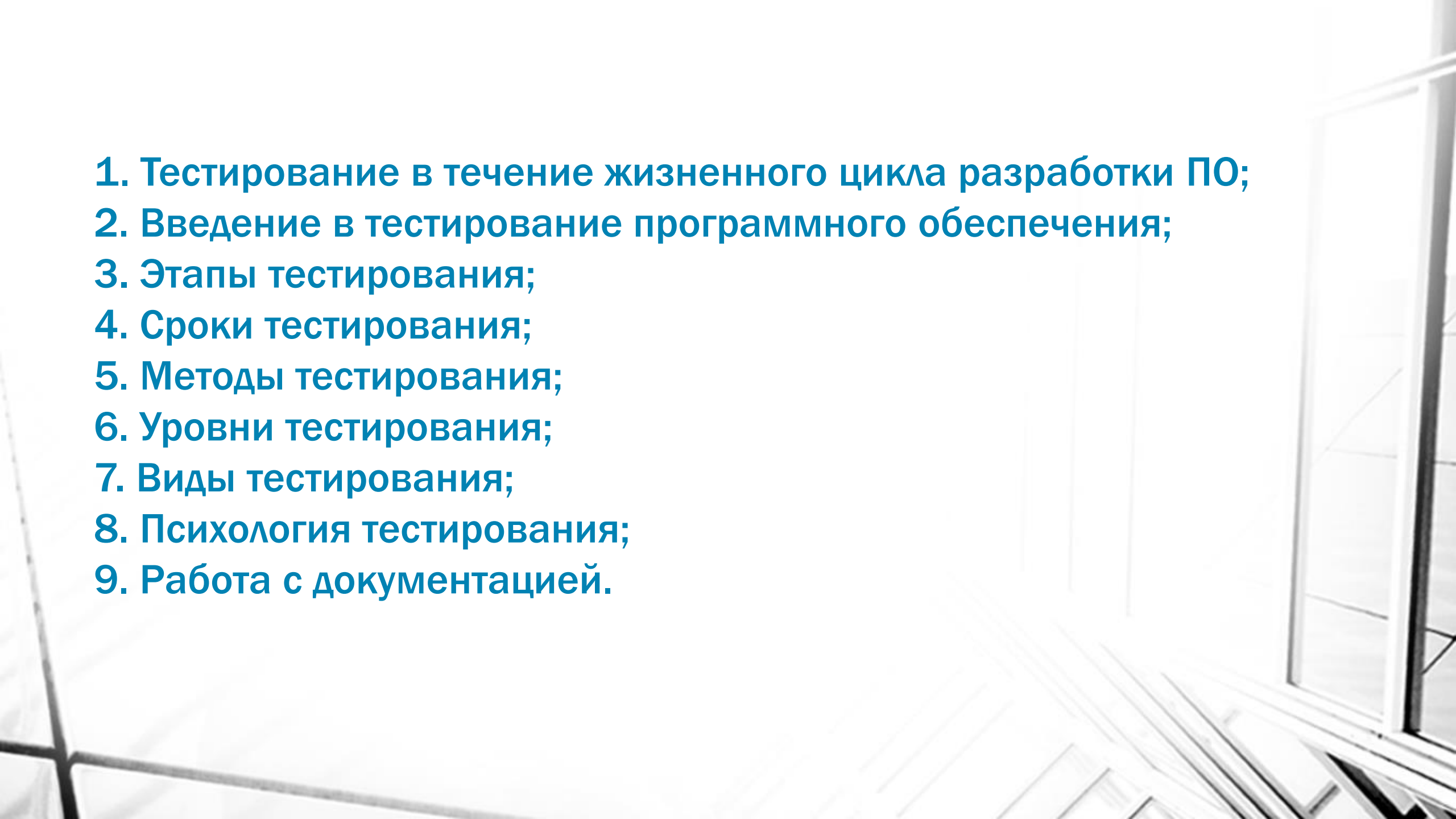


Тестирование Web-проектов

Лекция 1



- 
1. Тестирование в течение жизненного цикла разработки ПО;
 2. Введение в тестирование программного обеспечения;
 3. Этапы тестирования;
 4. Сроки тестирования;
 5. Методы тестирования;
 6. Уровни тестирования;
 7. Виды тестирования;
 8. Психология тестирования;
 9. Работа с документацией.

Тестирование в течение жизненного цикла разработки ПО

Модели жизненного цикла разработки ПО

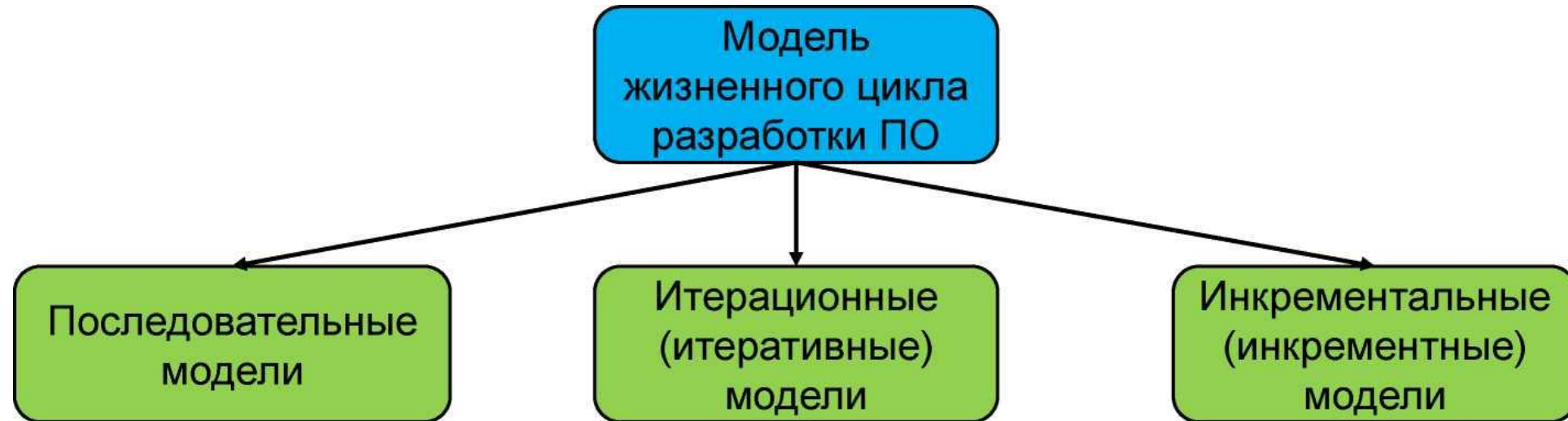
Модель жизненного цикла - структура процессов и видов деятельности, связанных с жизненным циклом, которые могут быть организованы по этапам, которая также служит общим ориентиром для общения и понимания [ISO/IEC 12207].

Процессы жизненного цикла ПО описаны в стандарте ISO/IEC 12207 System and software engineering - Software life cycle processes.



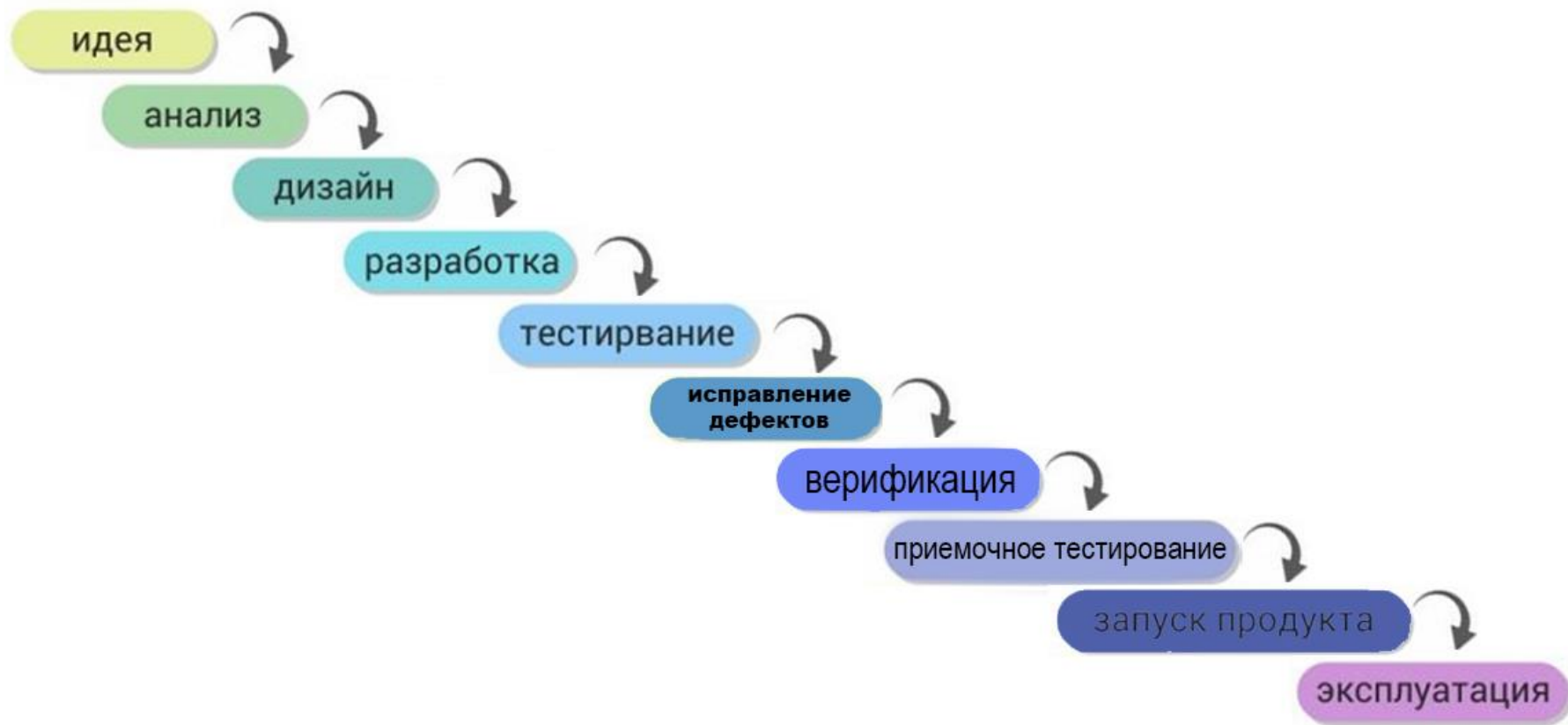
Модели жизненного цикла разработки ПО

Классификация моделей жизненного цикла разработки ПО



Модели жизненного цикла разработки ПО

Каскадная модель разработки ПО (водопадная).



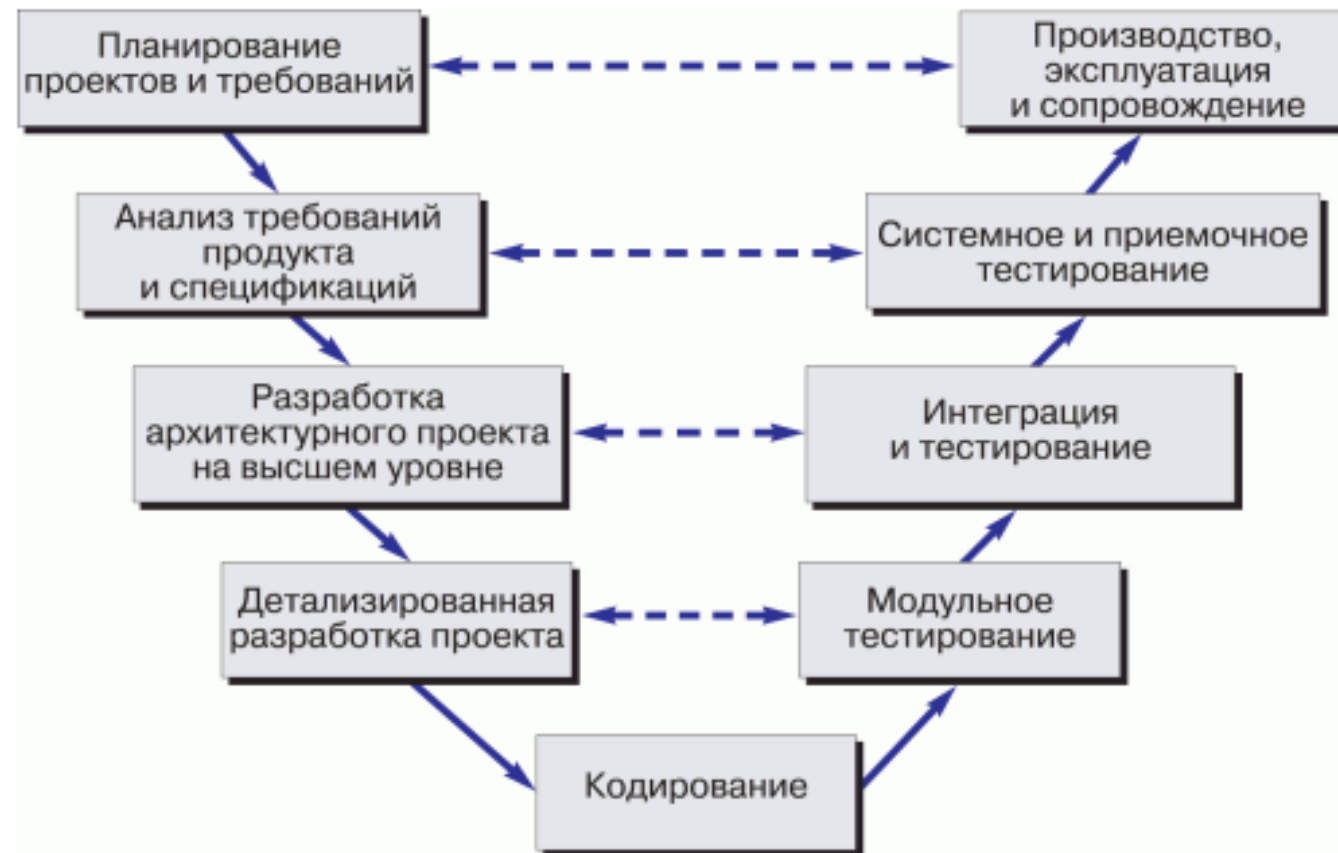
Модели жизненного цикла разработки ПО

Каскадная модель разработки ПО (водопадная)

Достоинства	Недостатки
Последовательное выполнение этапов проекта в строгом фиксированном порядке.	Низкая гибкость в управлении проектом.
Позволяет оценивать качество продукта на каждом этапе.	Тестирование начинается только с середины развития проекта.
Стабильность требований в течение всего жизненного цикла разработки.	Невозможность динамического изменения требований во всем жизненном цикле.
На каждой стадии формируется законченный набор проектной документации.	Интеграция всех полученных результатов происходит внезапно в завершающей стадии работы модели.
Определенность и понятность шагов модели и простота её применения.	Отсутствие обратных связей между этапами.
Четкое планирование сроков и затрат.	До завершения процесса разработки нельзя убедиться, качествен ли продукт.

Модели жизненного цикла разработки ПО

V - образная модель разработки ПО.



Модели жизненного цикла разработки ПО

V - образная модель разработки ПО.

Достоинства	Недостатки
Модель проста в использовании.	Не предусмотрено изменение требований на разных этапах жизненного цикла.
Строгая последовательность процесса разработки.	Тестирование требований в жизненном цикле происходит слишком поздно.
Раннее планирование, направленное на тестирование продукта.	Модель не учитывает итерации между фазами.
Каждому этапу разработки соответствует определенный этап тестирования.	В модель не входят действия, направленные на анализ рисков.
Определяет продукты, которые должны быть получены в результате разработки.	Длительное время разработки.
Четкое планирование сроков и затрат.	

Модели жизненного цикла разработки ПО

Спиральная модель разработки ПО.



Модели жизненного цикла разработки ПО

Спиральная модель разработки ПО.

Достоинства	Недостатки
Позволяет быстрее показать пользователям системы работоспособный продукт.	Модель имеет усложненную структуру.
Допускает изменение требований при разработке ПО.	Большое количество промежуточных циклов может привести к увеличению документации.
Обеспечивается определение непреодолимых рисков.	Необходимость в высокопрофессиональных знаниях для оценки рисков.
В модели предусмотрена возможность гибкого проектирования.	Спираль может продолжаться до бесконечности.
Позволяет получить более надежную и устойчивую систему.	Использование модели может оказаться дорогостоящим.
Разрешает пользователям активно принимать участие при планировании и разработке.	Постоянные отзывы и реакция заказчика может провоцировать новые итерации.

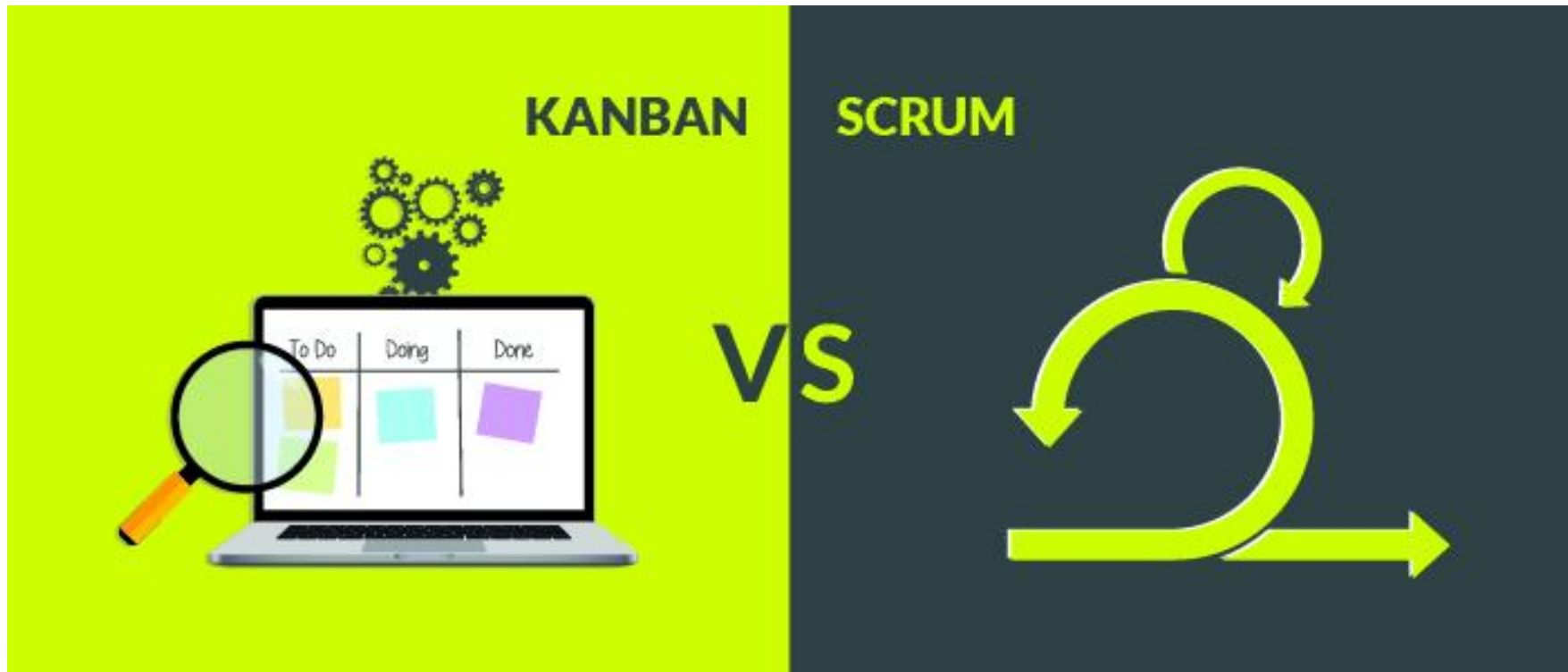
Модели жизненного цикла разработки ПО

Agile (agile software development, от англ. agile – проворный) – это семейство «гибких» подходов к разработке программного обеспечения. Такие подходы также иногда называют фреймворками или agile-методологиями.



Модели жизненного цикла разработки ПО

Scrum и **Kanban** — это гибкие методологии создания продукта. По ним можно работать в любой отрасли, но особенно хорошо они подходят для ИТ.



Модели жизненного цикла разработки ПО

Для любой модели жизненного цикла разработки существуют несколько показателей качественного тестирования:

- Для каждой активности разработки существует соответствующая активность тестирования.
- У каждого уровня тестирования есть цели, характерные для данного уровня.
- Анализ и проектирование тестов для определенного уровня тестирования начинаются на стадии соответствующих активностей разработки.
- Тестировщики должны участвовать в обсуждениях для определения и уточнения требований и дизайна, а также вовлекаться в рецензирование рабочих продуктов, как только будут доступны первые их черновые версии.



Введение в тестирование программного обеспечения

Введение в тестирование программного обеспечения

Тестирование программного обеспечения (software testing) — процесс анализа программного средства и сопутствующей документации с целью выявления дефектов и повышения качества продукта.

Тестирование ПО является процессом в силу того факта, что оно выполняется большим количеством людей на протяжении длительного периода деятельности.

QA, QC и тестирование

Quality Assurance	Quality Control	Тестирование
Комплекс мероприятий, который охватывает все технологические аспекты на всех этапах разработки, выпуска и введения в эксплуатацию программных систем для обеспечения необходимого уровня качества программного продукта	Процесс контроля соответствия разрабатываемой системы предъявляемым к ней требованиям	Процесс, отвечающий непосредственно за составление и прохождение тест-кейсов, нахождение и локализацию дефектов и т.д.
Фокус в большей степени на процессы и средства, чем на непосредственно исполнение тестирования системы	Фокус на исполнение тестирования путем выполнения программы с целью определения дефектов с использованием утвержденных процессов и средств	Фокус на исполнение тестирования как такового
Процессно-ориентированный подход	Продуктно-ориентированный подход	Продуктно-ориентированный подход
Превентивные меры	Корректирующий процесс	Превентивный процесс
Подмножество процессов Software Test Life Cycle – цикла тестирования ПО	Подмножество процессов QA	Подмножество процессов QC

Тестирование и качество

Обеспечение качества - является неотъемлемой частью разработки программного обеспечения.

Качество продукции:

- Снижает стоимость производства;
- Дает конкурентные преимущества.

Обеспечение качества программного обеспечения

Обеспечение качества (Quality Assurance - QA) - это совокупность мероприятий, охватывающих все технологические этапы разработки, выпуска и эксплуатации программного обеспечения (ПО) информационных систем, предпринимаемых на разных стадиях жизненного цикла ПО, для обеспечения требуемого уровня качества выпускаемого продукта.

Контроль качества (Quality Control - QC) - это совокупность действий, проводимых над продуктом в процессе разработки, для получения информации о его актуальном состоянии в разрезах: "готовность продукта к выпуску", "соответствие зафиксированным требованиям", "соответствие заявленному уровню качества продукта".

Предпосылки для развития QA

- Экономическая целесообразность;
- Конкурентные преимущества;
- Репутационная составляющая

Основные этапы QA

- Оценка уровня качества имеющихся на рынке аналогичных изделий, анализ требований покупателей;
- Долгосрочное прогнозирование;
- Планирование уровня качества;
- Разработка стандартов;
- Проектирование качества в процессе конструирования и разработки.

Дефект и ожидаемый результат

Дефект (defect, bug, баг, глюк) – любое несоответствие фактического и ожидаемого результата (согласно требованиям или здравому смыслу).



Ожидаемый результат (expected result) – такое поведение программного средства, которое мы ожидаем в ответ на наши действия.

Верификация и валидация

Эти два понятия тесно связаны с процессами тестирования и обеспечения качества. К сожалению, их часто путают, хотя отличия между ними достаточно существенны.

Верификация (Verification) — это статическая практика проверки документов, дизайна, архитектуры, кода, т.д.

Валидация (validation) – это процесс оценки конечного продукта, необходимо проверить, соответствует ли программное обеспечение ожиданиям и требованиям клиента.

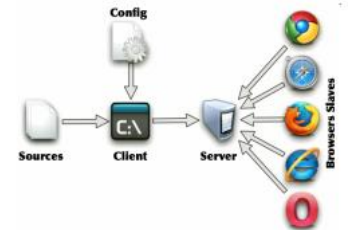
Основные тестовые документы и понятия

Тест план (test plan) – часть проектной документации, описывающая и регламентирующая процесс тестирования.



Чек лист (check list) – документ, описывающий какие функции должны быть проверены.

Тест кейс (test case) – набор входных данных, условий выполнения и ожидаемых результатов, разработанный с целью проверки того или иного свойства или поведения программного средства.



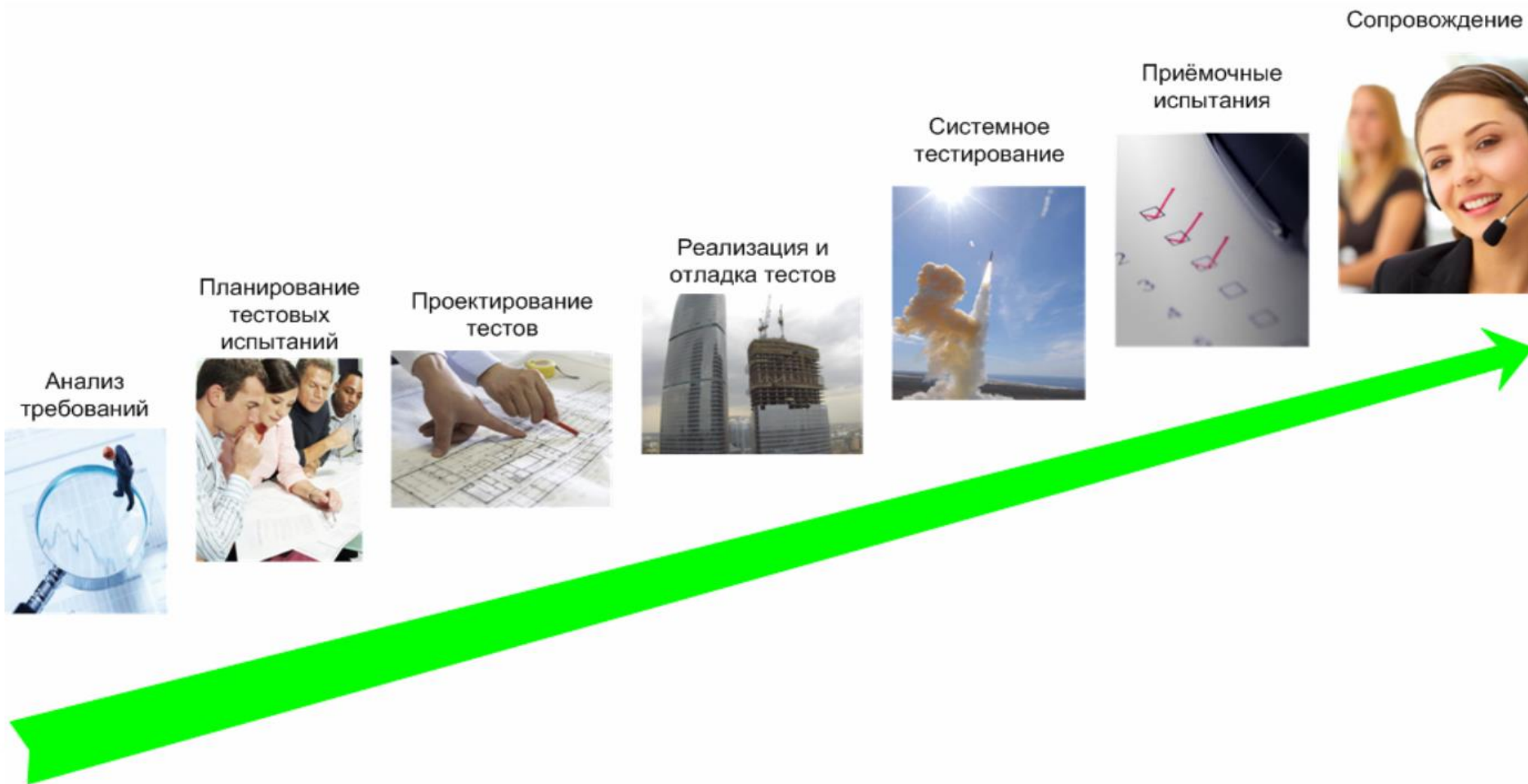
Билд (build) – промежуточная версия программного средства (финальный бил часто называют релизом (release)).

Основные направления тестирования

Статическое тестирование (static testing) – это процесс анализа самой разработки программного обеспечения, иными словами – это тестирование без запуска программы (проверка кода, требований, функциональной спецификации, архитектуры, дизайна и т.д.)

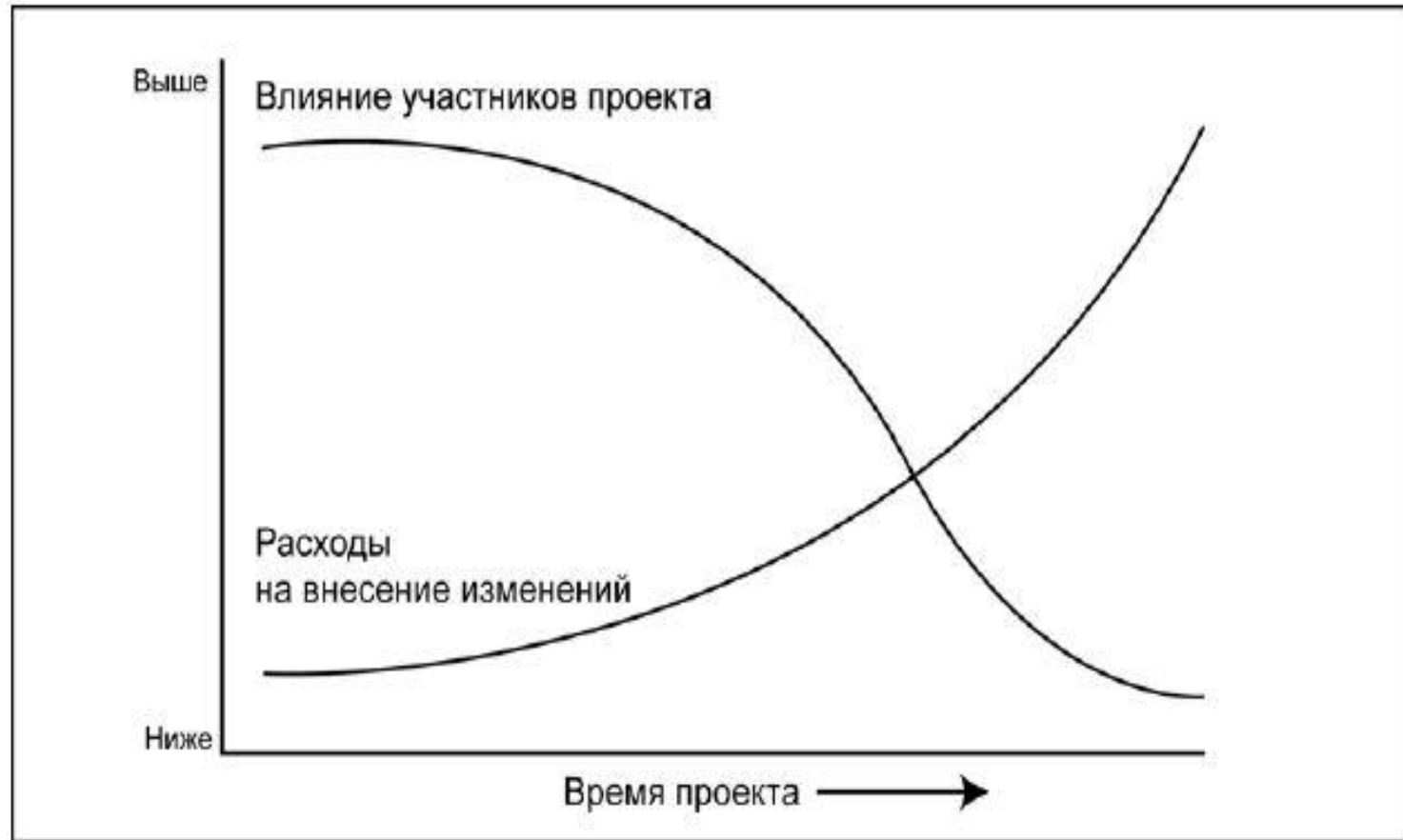
Динамическое тестирование (dynamic testing) – это тестовая деятельность, предусматривающая эксплуатацию (запуск) программного средства (ПС).

Этапы тестирования



Сроки тестирования

Когда начинать и заканчивать тестирование?



Когда начинать тестирование?

На разных этапах жизненного цикла ПО тестирование проводится в разных формах:

- на этапе определения требований: их анализ и верификация также могут считаться тестированием;
- контроль процесса проектирования на этапе разработки дизайна системы – это тоже форма тестирования;
- как уже упоминалось, разработчики тоже участвуют в тестировании на уровне модульного тестирования.

Когда заканчивается тестирование?

Критерий окончания тестирования:

1. граничные сроки, установленные заранее;
2. выполнение всех предусмотренных тест-кейсов;
3. достижение определенного уровня тестового покрытия;
4. когда после определенного момента, мы практически не находим новых багов или критических дефектов;
5. решение менеджмента.

Методы тестирования

Метод белого ящика (white-box testing) – используется для тестирования программного кода без запуска.



Тестировщик имеет доступ к исходному коду ПС. Тесты основаны на знании кода приложения и его внутренних механизмов.

Метод белого ящика часто используется на стадии, когда приложение еще не собрано воедино, но необходимо проверить каждый из его компонентов, модулей, процедур и подпрограмм.

Методы тестирования

Метод черного ящика (black-box testing) – заключается в том, что тестировщик имеет доступ к ПО только через те же интерфейсы, что и заказчик или пользователь.



Тестирование черного ящика ведется с использованием спецификаций или иных документов, описывающих требования к системе на основе применения пользовательского интерфейса для ввода входных и получения выходных данных.

Цель данного метода – проверить работу всех функций приложения на соответствие функциональным требованиям.

Методы тестирования

Метод серого ящика (gray box testing) – совокупность подходов из методов белого и черного ящика.



Этот метод, как правило, используется при тестировании web-приложений, когда тестировщик знает принципы функционирования технологий, на которых построено приложение, но может не видеть кода самого приложения.

Уровни тестирования

Компонентное тестирование (component testing) – тестирование отдельного модуля программного средства (под модулем может пониматься в т.ч. отдельный класс, метод и т.д.).

Интеграционное тестирование (integration testing) – проверка того, как отдельные компоненты, проверенные на предыдущем уровне, взаимодействуют друг с другом.

Системное тестирование (system testing) – полная проверка приложения: проверяются как функциональные, так и нефункциональные требования.



Компонентное



Интеграционное



Системное

Функциональное тестирование

Функциональное тестирование (functional testing) – процесс проверки программного обеспечения, сконцентрированный на анализе соответствия ПО требованиям и спецификациям.

Функциональное тестирование может быть ручным (manual testing) или автоматизированным (automated testing).

Цели функционального тестирования

- Обнаружить дефекты в программном продукте.
- Определить степень соответствия программного продукта требованиям и ожиданиям заказчика.
- Принять решение о возможности передачи программного продукта заказчику.

Уровни функционального тестирования

Приемочный тест (smoke test) – самый первый и быстрый уровень. Если тесты этого уровня не проходят, тестирование прекращается.

Тест критического пути (critical path test) – проверка таких функций приложения, с которыми обычный пользователь работает в повседневной жизни.

Расширенный тест (extended test) – полная проверка приложения на соответствие всем требованиям.

Виды тестирования

Не существует стандартного перечня видов тестирования, однако некоторые из них настолько очевидны, что стали общепринятыми. Итак ...

Инсталляционное тестирование (installation testing) – проверка всего того, что связано с инсталляцией продукта в систему и удалением продукта из системы.

Регрессионное тестирование (regression testing) – проверка того, что внесенные в приложение изменения не привели к потере работоспособности того, что ранее работало и/или привели к работоспособности того, что ранее не работало.

Виды тестирования

Тестирование нового функционала (new feature testing) – проверка того, что заявленный в данном билде новый функционал работает должным образом.

Конфигурационное тестирование (configuration testing) – проверка того, как приложение работает с различным оборудованием и/или собственными настройками.

Тестирование совместимости (compatibility testing) – проверка того, как приложение взаимодействует с другими приложениями и операционной системой. В случае web-ориентированных приложений особое внимание уделяется совместимости с различными браузерами.

Виды тестирования

Тестирование удобства использования (usability testing) – проверка того, насколько пользователю удобно и приятно работать с приложением.

Тестирование интернационализации (internationalisation testing) – проверка готовности продукта к переводу на различные языки.

Тестирование локализации (localisation testing) – проверка качества перевода продукта на конкретный язык.

Виды тестирования

Позитивное тестирование (positive testing) – проверка того, как приложение работает в заведомо «тепличных условиях» (корректные данные, условия работы и т.п.).

Негативное тестирование (negative testing) – проверка того, как приложение реагирует на различные негативные моменты (пропала сеть, поврежден файл и т.п.).

Исследовательское тестирование (exploratory testing) – редкий вид тестирования, основанный на профессиональной интуиции тестировщика, которая может подсказать опасные и малоисследованные способы работы с приложением.

Психология тестирования

Всего выделяют четыре уровня независимости – от низкого до высокого:

1. тесты для программы разрабатываются и проводятся человеком, который является ее автором;
2. тесты разрабатываются и выполняются другими людьми (например, другим разработчиком);
3. тесты разработаны представителями другой организационной группы (например, из отдела тестирования) или специализированными тестировщиками (например, специалистами по тестированию производительности или безопасности);
4. тесты разрабатываются и выполняются специалистами из другой организации (например, аутсорсинговой или аудиторской).

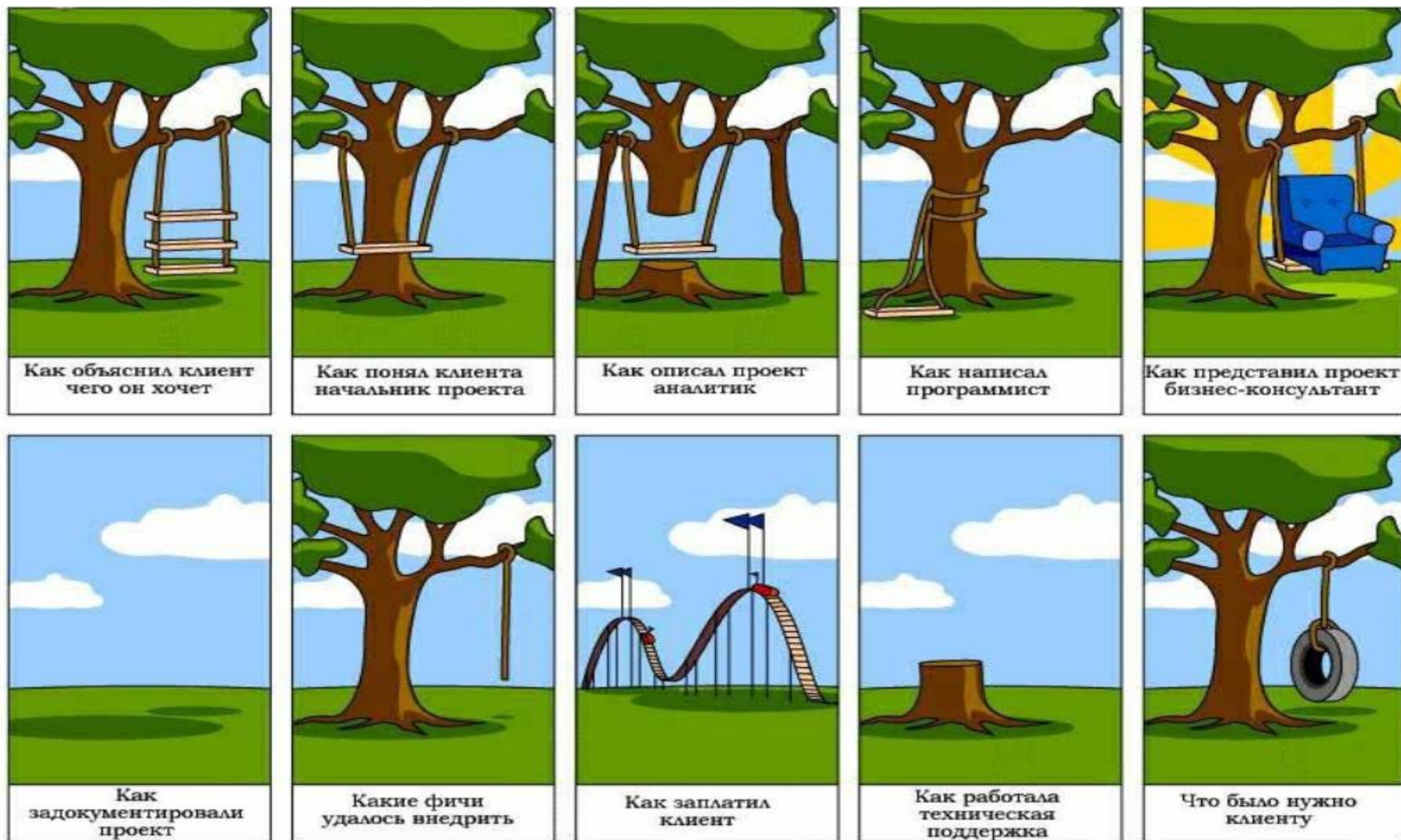
Психология тестирования

Есть несколько простых советов для улучшения коммуникации с коллегами:

- помните о том, что все вы работаете над одним проектом и идете к одной цели – созданию качественного и востребованного продукта;
- оформляйте результаты своей работы в нейтральном тоне, сфокусируйтесь на фактах;
- поставьте себя на место других и попытайтесь понять причины их поведения;
- всегда убеждайтесь в том, что другой человек понял Вас, а Вы его.

Работа с документацией

Важность документации



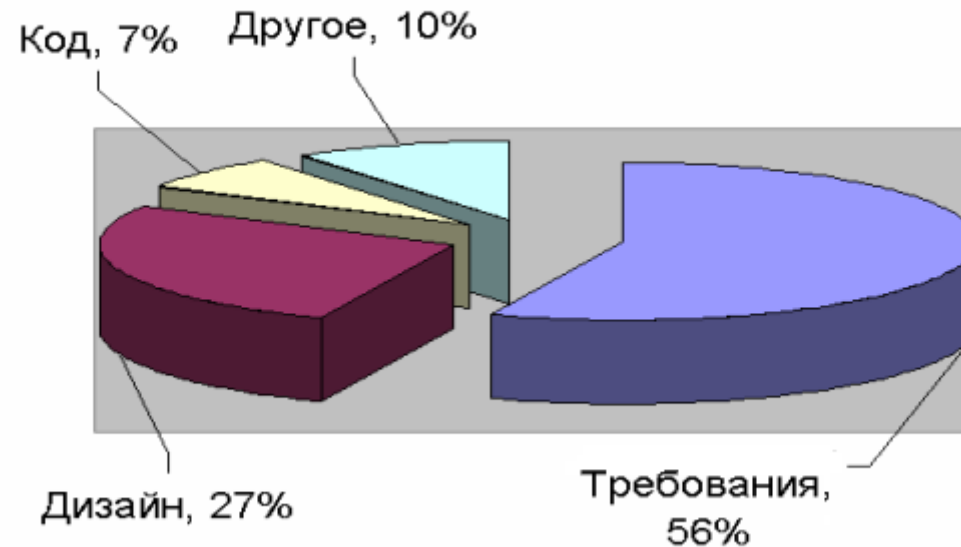
Важность тестирования документации



Требование

... всё так или иначе начинается с документации и требований.

Именно в требованиях берет начало больше всего дефектов (а не в коде, как думают многие).



Требование

Требование (requirement) — описание того, какие функции и с соблюдением каких условий должно выполнять приложение в процессе решения полезной для пользователя задачи.



Источники и пути выявления требований

Интервью. Самый универсальный путь выявления требований, заключающийся в общении проектного специалиста (как правило, специалиста по бизнесанализу) и представителя заказчика (или эксперта, пользователя и т.д.).

Работа с фокусными группами. Может выступать как вариант «расширенного интервью», где источником информации является не одно лицо, а группа лиц (как правило, представляющих собой целевую аудиторию, и/или обладающих важной для проекта информацией, и/или уполномоченных принимать важные для проекта решения).

Источники и пути выявления требований

Анкетирование. Этот вариант выявления требований вызывает много споров, т.к. при неверной реализации может привести к нулевому результату при объёмных затратах. Ключевым фактором успеха является правильное составление анкеты, правильный выбор аудитории и правильное преподнесение анкеты.

Семинары и мозговой штурм. Семинары позволяют группе людей очень быстро обмениваться информацией (и наглядно продемонстрировать те или иные идеи), а также хорошо сочетаются с интервью, анкетированием, прототипированием и моделированием — в том числе для обсуждения результатов и формирования выводов и решений.

Источники и пути выявления требований

Наблюдение. Может выражаться как в буквальном наблюдении за некоторыми процессами, так и во включении проектного специалиста в эти процессы в качестве участника.

Прототипирование. Состоит в демонстрации и обсуждении промежуточных версий продукта (например, дизайн страниц сайта может быть сначала представлен в виде картинок, и лишь затем сверстан).

Анализ документов. К этой технике относится изучение документов, регламентирующих бизнес-процессы в предметной области заказчика или в конкретной организации, что позволяет приобрести необходимые для лучшего понимания сути проекта знания.

Источники и пути выявления требований

Моделирование процессов и взаимодействий. Может применяться как к «бизнес-процессам и взаимодействиям» (например: «договор на закупку формируется отделом закупок, визируется бухгалтерией и юридическим отделом...»), так и к «техническим процессам и взаимодействиям» (например: «платёжное поручение генерируется модулем “Бухгалтерия”, шифруется модулем “Безопасность” и передаётся на сохранение в модуль “Хранилище”»).

Самостоятельное описание. Является не столько техникой выявления требований, сколько техникой их фиксации и формализации. Очень сложно (и даже нельзя!) пытаться самому «придумать требования за заказчика», но в спокойной обстановке можно самостоятельно обработать собранную информацию и аккуратно оформить её для дальнейшего обсуждения и уточнения.

Уровни требований

Уровень бизнес-требований.

Бизнес правила.

Уровень пользовательских требований.

Пользовательские требования.

Атрибуты качества.

Уровень продуктовых требований.

Функциональные требования.

Нефункциональные требования.

Ограничения.

Требования к интерфейсам.

Требование к данным.

Типы требований

Бизнес-требования (business requirements) выражают цель, ради которой разрабатывается продукт (зачем вообще он нужен, какая от него ожидается польза, как заказчик с его помощью будет получать прибыль).

Примеры бизнес-требований:

- Нужен инструмент, в реальном времени отображающий наиболее выгодный курс покупки и продажи валюты.
- Необходимо в два-три раза повысить количество заявок, обрабатываемых одним оператором за смену.
- Нужно автоматизировать процесс выписки товарно-транспортных накладных на основе договоров.

Бизнес-правила (business rules) описывают особенности принятых в предметной области (и/или непосредственно у заказчика) процессов, ограничений и иных правил.

Типы требований

Пользовательские требования (user requirements) описывают задачи, которые пользователь может выполнять с помощью разрабатываемой системы (реакцию системы на действия пользователя, сценарии работы пользователя).

Примеры пользовательских требований:

- При первом входе пользователя в систему должно отображаться лицензионное соглашение.
- Администратор должен иметь возможность просматривать список всех пользователей, работающих в данный момент в системе.
- При первом сохранении новой статьи система должна выдавать запрос на сохранение в виде черновика или публикацию.

Атрибуты качества (quality attributes) расширяют собой нефункциональные требования и на уровне пользовательских требований могут быть представлены в виде описания ключевых для проекта показателей качества (свойств продукта, не связанных с функциональностью, но являющихся важными для достижения целей создания продукта — производительность, масштабируемость, восстанавливаемость).

Типы требований

Функциональные требования (functional requirements) описывают поведение системы, т.е. её действия (вычисления, преобразования, проверки, обработку и т.д.).

Примеры функциональных требований:

- В процессе инсталляции приложение должно проверять остаток свободного места на целевом носителе.
- Система должна автоматически выполнять резервное копирование данных ежедневно в указанный момент времени.

Типы требований

Нефункциональные требования (non-functional requirements) описывают свойства системы (удобство использования, безопасность, надёжность, расширяемость и т.д.), которыми она должна обладать при реализации своего поведения.

Примеры нефункциональных требований:

- При одновременной непрерывной работе с системой 1000 пользователей, минимальное время между возникновением сбоев должно быть более или равно 100 часов.
- Ни при каких условиях общий объём используемой приложением памяти не может превышать 2 ГБ.

Типы требований

Ограничения (limitations, constraints) представляют собой факторы, ограничивающие выбор способов и средств (в том числе инструментов) реализации продукта.

Примеры ограничений:

- Все элементы интерфейса должны отображаться без прокрутки при разрешениях экрана от 800x600 до 1920x1080.
- Не допускается использование Flash при реализации клиентской части приложения.

Типы требований

Требования к интерфейсам (external interfaces requirements) описывают особенности взаимодействия разрабатываемой системы с другими системами и операционной средой.

Примеры требований к интерфейсам:

- Обмен данными между клиентской и серверной частями приложения при осуществлении фоновых AJAX-запросов должен быть реализован в формате JSON.
- Протоколирование событий должно вестись в журнале событий операционной системы.

Типы требований

Требования к данным (data requirements) описывают структуры данных (и сами данные), являющиеся неотъемлемой частью разрабатываемой системы.

Примеры требований к данным:

- Все данные системы, за исключением пользовательских документов, должны храниться в БД под управлением СУБД MySQL, пользовательские документы должны храниться в БД под управлением СУБД MongoDB.
- Информация о кассовых транзакциях за текущий месяц должна храниться в операционной таблице, а по завершении месяца переноситься в архивную.

Типы требований

Спецификация требований (software requirements specification, SRS) объединяет в себе описание всех требований уровня продукта и может представлять собой весьма объёмный документ (сотни и тысячи страниц).

Важность тестирования требований

Хорошо проработанные требования позволяют:

- Выработать общее понимание между заказчиком и разработчиком.
- Определить рамки проекта.
- Более точно определить финансовые и временные характеристики проекта.
- Обезопасить заказчика от риска получить продукт, в котором он не сможет работать.
- Обезопасить разработчика от риска попасть в ситуацию «неконтролируемого размытия границ», которое может привести к непредвиденным затратам ресурсов сверх начальных ожиданий.

Документирование требований

Стандарт IEEE 830-1998 содержит развёрнутое описание требований, которое может быть оптимизировано для нужд конкретной организации.

В описание требований рекомендуется включать:

1. Введение

1.1. Назначение документа.

1.2. Поддерживаемые соглашения.

1.3. Предполагаемая аудитория и рекомендации по последовательности работы с документом для каждого класса читателей.

1.4. Границы проекта. Здесь содержится ссылка на документ «Концепция», если таковой имеется, либо краткое резюме продукта.

1.5. Ссылки.

Документирование требований

2. Общее описание

2.1. Общий взгляд на продукт. Здесь необходимо определить, является ли описываемый продукт новым членом растущего семейства продуктов, новой версией существующей системы, заменой существующего приложения или совершенно новым продуктом. Если спецификация требований определяет компонент более крупной системы, укажите, как это ПО соотносится со всей системой и определите основные интерфейсы между ними.

2.2. Особенности продукта. Перечисляются ключевые особенности продукта или его главные свойства. Здесь уместно поместить контекстную диаграмму (в виде диаграммы вариантов использования, потоков данных или др. спецификаций).

2.3. Классы и характеристики пользователей. Документируется процесс поиска актеров (в терминологии RUP), в котором выявляются все пользователи системы и осуществляется обобщение (выделение классов) пользователей. Найденные классы описываются (например - уровень квалификации, доступный функционал и т.д.)

Документирование требований

2.4. Операционная среда. Рассматривается среда функционирования программного продукта, включая аппаратные средства, операционные системы, для распределённых систем - географическое расположение пользователей и серверов, топология сети.

2.5. Ограничения проектирования и реализации. Рассмотрим классификацию ограничений:

- определённые технологии, средства, языки программирования и базы данных, которые следует использовать или избегать;
- ограничения, налагаемые операционной средой продукта;
- обязательные соглашения или стандарты разработки;
- обратная совместимость с продуктами, выпущенными ранее;
- ограничения, налагаемые бизнес-правилами;
- ограничения, связанные с оборудованием, например требования к быстродействию, ограничения памяти или процессора;
- соглашения, связанные с пользовательским интерфейсом существующего продукта, которые необходимо соблюдать при его улучшении;
- форматы и протоколы обмена данными.

2.6 Документация для пользователей.

2.7 Предположения и зависимости

Документирование требований

3. Функции системы

Для каждой *i*-й функции составляется следующее описание.

3.1 Наименование *i*-й функции системы.

3.1.1 Описание и приоритеты. Приводится краткое описание функции и указывается её приоритет (степень важности/очередности реализации).

3.1.2 Последовательности «воздействие - реакция». Необходимо перечислить последовательность воздействий, оказываемых на систему (действия пользователей, сигналы внешних устройств и др.), и отклики системы, определяющие реакцию конкретной функции.

3.1.3 Функциональные требования. Необходимо дать детализацию *i*-й функции, перечислить детализированные функциональные требования, включая реакцию на ожидаемые ошибки и неверные действия. Каждому детальному функциональному требованию присваивается уникальный идентификатор.

Документирование требований

4. Требования к внешнему интерфейсу

4.1 Интерфейсы пользователя (user interface, UI)

Основные характеристики UI:

- ссылки на стандарты графического интерфейса пользователей или стилевые рекомендации для семейства продукта, которые необходимо соблюдать;
- стандарты шрифтов, значков, названий кнопок, изображений, цветовых схем, последовательностей полей вкладок, часто используемых элементов управления и т.д.;
- конфигурация экрана или ограничения разрешения;
- стандартные кнопки, функции или ссылки перемещения, одинаковые для всех экранов (например, кнопка справки);
- быстрые клавиши («клавиатурные шорт-каты», «short-cut»);
- стандарты отображения сообщений;
- стандарты конфигурации для упрощения локализации ПО;
- специальные возможности для пользователей с проблемами со зрением.

Документирование требований

4.2 Интерфейсы оборудования

Опишите характеристики каждого интерфейса между компонентами ПО и оборудованием системы. В описание могут входить типы поддерживаемых устройств, взаимодействие данных и элементов управления между ПО и оборудованием, а также протоколы взаимодействия.

4.3 Интерфейсы ПО

Опишите соединения продукта и других компонентов ПО (идентифицированные по имени и версии), в том числе базы данных, операционные системы, средства, библиотеки и интегрированные коммерческие компоненты. Укажите назначение элементов сообщений, данных и элементов управления, обмен которыми происходит между компонентами ПО. Опишите службы, необходимые внешним компонентам ПО, и природу взаимодействия между компонентами. Определите данные, к которым будут иметь доступ компоненты ПО.

Документирование требований

4.4 Интерфейсы передачи информации

Укажите требования для любых функций взаимодействия, которые будут использоваться продуктом, включая электронную почту, веб-браузер, протоколы сетевого соединения и электронные формы. Определите соответствующие форматы сообщений. Опишите особенности безопасности взаимодействия или шифрования, частоты передачи данных и механизмов синхронизации.

Документирование требований

5. Другие нефункциональные требования

В этом разделе описываются остальные нефункциональные требования, не относящиеся к требованиям к интерфейсу, которые представлены в разделе 4, и к ограничениям, описываемым в разделе 2.5.

5.1 Требования к производительности

Укажите специальные требования к производительности для различных системных операций. Обоснуйте их необходимость для того, чтобы помочь разработчикам принять правильные решения, касающиеся дизайна. Например, из-за жёстких требований к времени отклика базы данных разработчики могут зеркализовать базу данных в нескольких географических положениях или денормализовать связанные таблиц баз данных для получения более быстрого ответа на запрос.

Документирование требований

6. Приложения

Приложение А. Словарь терминов (гlossарий).

Приложение Б. Модели анализа. В этот раздел помещаются все модели, построенные в процессе анализа требований.

Приложение В. Список вопросов. Это динамический список ещё не решённых проблем, связанных с требованиями. Это могут быть элементы, помеченные как "TBD" («to be determined», «to be defined» - «будет определено», «необходимо определить»), отложенные решения, необходимая информация, неразрешённые конфликты и т.д.

Хорошее требование

Каждое требование должно быть:

Завершённым (complete). Все важные аспекты должны быть включены. Ничто не должно быть оставлено «для будущего определения» (2BD - to be defined).

Непротиворечивым (consistent). Требование не должно содержать противоречий как внутри себя, так и с другими требованиями.

Корректным (correct). Требование должно чётко указывать на то, что должно выполнять приложение. Недопустимо при написании требования предполагать, что что-то окажется очевидным. Каждый человек понимает это «очевидное» по-своему, и в итоге система получится не такой, как задумывалось.

Недвусмысленным (unambiguous). Требование не должно допускать разночтений.

Проверяемым (verifiable). Требование должно быть сформулировано так, чтобы существовали способы однозначной проверки - выполнено требование или нет.

Хороший набор требований

Наборы требований должны быть:

Модифицируемыми (modifiable). Структура и стиль набора требований должны быть такими, чтобы набор требований можно было легко модифицировать. Должна отсутствовать избыточность. Должно быть построено корректное содержание всего документа.

Прослеживаемыми (traceable). У каждого требования должен быть уникальный идентификатор, по которому на это требование можно сослаться.

Проранжированными по важности, стабильности и срочности (ranked for importance, stability and priority). Для каждого требования должен быть указан уровень его важности (насколько оно важно для заказчика), стабильности (насколько высока вероятность, что это требование ещё будет изменено в процессе обсуждения деталей проекта) и срочности (как быстро требование должно быть реализовано).

Проблемы с требованиями

Проблемы незавершённости (неполноты)

Хорошо, когда вся важная информация присутствует. Только вот вопрос - а **как можно найти то, чего нет, но должно быть?** Как догадаться, что что-то отсутствует?

Следует обратить внимание на такие типичные случаи.

Отсутствуют нефункциональные требования или нефункциональные составляющие требования. Мы знаем, что система должна делать. А про то, как (как быстро, как безопасно) в лучшем случае узнаём только в самом конце.

Проблемы с требованиями

Проблемы незавершённости (неполноты) продолжение

Чтобы легче было понять суть потребностей заказчика, рекомендуется задавать общие вопросы.

Их преимущества:

- Они универсальные - большая часть их применима к любому проекту, независимо от специфики продукта.
- Они не навязывают решение - больше шансов что, заказчик случайно упомянет то, что для него очевидно (и поэтому он нам об этом раньше не говорил).
- Они не загоняют заказчика в ситуацию, когда ему приходится выбирать из имеющихся вариантов (а на самом деле всё будет по тому варианту, который вы забыли упомянуть).

Хорошая аналогия - вопросы репортера (или маленького ребёнка): «А что?», «А зачем?», «А почему?»

Проблемы с требованиями

Проблемы противоречивости

Бывают противоречия как внутри одного требования, так и между двумя и более требованиями. Бывают противоречия между таблицами и текстом, картинкой и текстом. Противоречия между требованием и прототипом.

Для устранения противоречий в требованиях, **надо хотя бы один раз прочитать ВСЕ требования** (звучит страшно, но такова жизнь) - не только ту часть, за которую отвечаете именно вы, - чтобы как минимум знать о чём там говорится и представлять, где ещё есть вещи которые могут вас касаться.

Во-вторых, **надо убедиться, что одно и тоже называется во всех частях документации одним и тем же словом** - тут помогает глоссарий.

В-третьих, изучая «требования про А», надо проверить, не противоречат ли они:

- другим «требованиям про А»;
- требованиям, более общим, покрывающим «требования про А»;
- требованиям, связанным с «требованиями про А»
- требованиям про что-то другое, но аналогичным «требованиям про А»;
- как, следуя «требованиям про А», можно нарушить другие требования.

Проблемы с требованиями

Проблемы некорректности

Документы часто бывают большими, объёмными, сложными. Они меняются по ходу разработки. Как и любой продукт деятельности человека, документы (и требования в т.ч.) с большой вероятностью содержат ошибки.

Ошибки могут быть вызваны:

- опечатками, последствиями «copy-paste»;
- остатками устаревших требований (когда мы забыли что-то где-то изменить);
- наличием «озолочения» («gold plating») - требованиями по реализации бессмысленного, но очень дорогостоящего функционала;
- наличием технически невыполнимых требований;
- наличием неаргументированных требований к дизайну и архитектуре.

Проблемы с требованиями

Проблемы двусмысленности

Если что-то можно понять несколькими способами, можно быть уверенным, что разные люди поймут это по-разному.

Например. В требовании сказано, что «функциональность X» является опциональной.

Что думает отдел разработки? «Чудесно. Опционально - значит необязательно. Можно не реализовывать.»

Что думает отдел маркетинга? «Ага. Мы выпустим две версии продукта. В более дорогой будет эта функциональность.»

Что думает заказчик? «Я за те же деньги получу ещё и вот эту функциональность».

Вывод? Следует писать требования так, чтобы исключить возможность их двоякого понимания.

Проблемы с требованиями

Проблемы двусмысленности продолжение

В английском языке есть много слов-индикаторов, наличие которых в требовании должно насторожить тестировщика.

Adequate, be able to, easy, provide for, as a minimum, be capable of, effective, timely, as applicable, if possible, TBD, as appropriate, if practical, at a minimum, but not limited to, capability of, capability to, normal, minimise, maximise, optimise, rapid, user-friendly, simple, often, usual, large, flexible, robust, state-of-the-art, improved, efficient.

Очевидно, их русские аналоги приводят к тем же проблемам.

Оцените: «Приложение должно обрабатывать большие файлы». Большие? Два мегабайта - это большой файл? Или надо 100 терабайт?

Проблемы с требованиями

Проблемы непроверяемости

Для начала следует отметить, что все вышеперечисленные проблемы ведут в том числе к тому, что мы не можем проверить, удовлетворяет ли продукт требованию.

Как понять, что требование непроверяемо? Попробовать придумать несколько тестов для его проверки. Если тесты не придумываются - вот она, проблема.

А бывает ли «просто непроверяемое требование»? Да.

Например: «В приложении должно быть ноль ошибок». «Приложение должно поддерживать все версии всех операционных систем».

Проблемы с требованиями

Проблемы непроранжированности

Если требования не проранжированы по важности и срочности, мы рискуем уделить основное внимание не тому, что на самом деле важно для заказчика.

Хорошие требования (и наборы требований) всегда проранжированы по трём показателям:

- важности (насколько требование важно);
- приоритетности выполнения (как быстро его надо реализовать);
- стабильности (какова вероятность, что требование изменится).

Проблемы с требованиями

Ещё раз самое важное

Если мы не можем проверить требование - это проблема.

В конечном итоге именно тестировщики отвечают за то, чтобы требование было проверено.

Если мы не можем проверить требование по объективным причинам, мы уточняем его до тех пор, пока оно не станет проверяемым. В зависимости от ситуации, мы расспрашиваем заказчика, разработчиков, своих более опытных коллег и т.д.

В случае очень важных и нерешаемых простыми способами проблем, решение будет приниматься менеджером проекта и заказчиком.

Проблемы с наборами требований

Набор немодифицируем

Это значит, что после каждого обновления требований понадобится тратить недели чтобы выловить все появившиеся противоречия.

Набор непроранжирован

Можно потратить кучу усилий на неважную функциональность, а самое основное не успеть. Или можно потратить много времени на то, что полностью изменится в новой версии требований, и работу придётся повторять заново. Можно обмануть ожидания заказчика, ожидавшего получить тот или иной набор функций к такому-то времени.

Набор непрослеживаем

Если требования не пронумерованы, не имеют чёткого оглавления, не имеют работающих перекрёстными ссылок - это хаос. А в хаосе ошибки плодятся с удивительной скоростью.

Работа с требованиями

Одна из наиболее распространённых техник работы с требованиями - взаимный перепросмотр.

Суть взаимного перепросмотра требований проста: после того, как один человек создал требование, другой человек это требование проверяет.

Обычно, выделяют три уровня перепросмотра:

- 1) **Неформальный перепросмотр.** Двое коллег просто обмениваются листиками (файликами) и правят найденные ошибки, которые потом обсуждаются за чашкой чая или в любое другое относительно свободное время.
- 2) **Технический перепросмотр.** Это немного более формализованный процесс, требующий подготовки, выделенного времени, участия некоторой группы специалистов (желательно, из различных областей).
- 3) **Формальная инспекция.** Проводится редко и в случае очень больших проектов и крайней необходимости. Описывается специальными стандартами, требует соблюдения широкого спектра правил и протоколирования результатов.

Техники работы с требованиями

Существует три простые техники работы с требованиями:

- задавать вопросы,
- писать тест кейсы,
- рисовать рисунки.

Самый простой и не требующий большого опыта способ - задавать как можно больше вопросов. Получая разнообразные ответы от разных участников проекта (как наших коллег так и представителей заказчика), мы расширяем, углубляем и уточняем своё представление о том, что и как должно работать.

Второй способ - создавать тест-кейсы. Когда вы видите требование, спросите себя: «Как я буду его тестировать? Какие тесты очевидно покажут, что это требование реализовано в ПС правильно?» Если с придумыванием таких тестов вы испытываете сложность - это тревожный звоночек: скорее всего, в требовании есть проблемы.

Третий способ - рисовать. Чтобы увидеть общую картину требований целиком, очень удобно использовать рисунки, схемы, диаграммы и т.д.

Немного юмора

Тестирование требований в действии



**Благодарю за
внимание**