# 1. Тестирование. Вводная. Цели.

Какова цель тестирования?

Цель тестирования в любом его проявлении и на любой стадии развития проекта – это повышение качества проекта.

Эта цель достигается за счет оценки текущего состояния программного средства и формулировки рекомендаций.

Таким образом, тестирование программного обеспечения — процесс анализа программного средства и сопутствующей документации с целью выявления дефектов и повышения качества продукта

*Согласно глоссарию ISTQB: тестирование (testing) – это процесс, содержащий в себе все активности жизненного цикла, как динамические, так и статические, касающиеся планирования, подготовки и оценки программного продукта и связанных с этим результатов работ с целью определить, что они соответствуют описанным требованиям, показать, что они подходят для заявленных целей и для определения дефектов.*

*Согласно ISTQB цели тестирования могут включать в себя:*

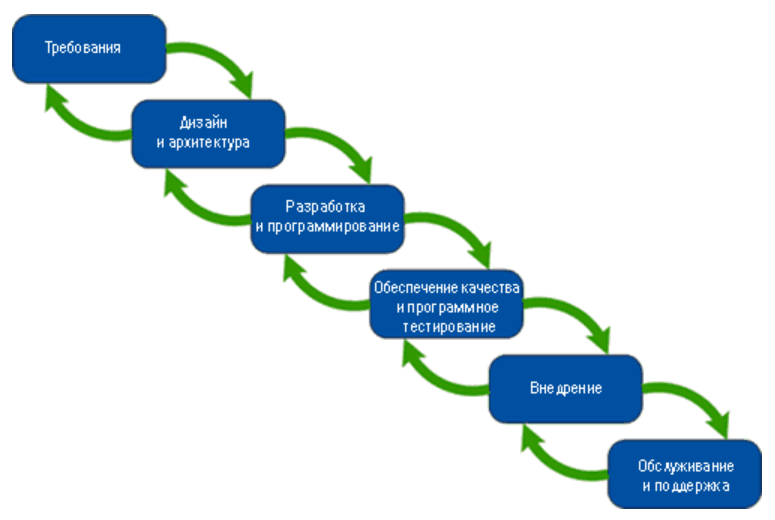
* *Оценку рабочих продуктов, таких как требования, пользовательские истории, проектирование и код*
* *Проверку, все ли указанные требования выполнены*
* *Проверку, завершен ли объект тестирования и работает, как ожидают пользователи и заинтересованные лица*
* *Создание уверенности в уровне качества объекта тестирования*
* *Предотвращение дефектов*
* *Обнаружение отказов и дефектов*
* *Предоставление заинтересованным лицам достаточной информации, позволяющей им принять обоснованные решения, особенно в отношении уровня качества объекта тестирования*
* *Снижение уровня риска ненадлежащего качества программного обеспечения (например, пропущенные сбои в работе)*
* *Соблюдение договорных, правовых или нормативных требований, или стандартов и/или проверка соответствия объекта тестирования таким требованиям и стандартам*

# 2. Жизненный цикл программного обеспечения. Жизненный цикл разработки программного обеспечения

Поговорим о жизненном цикле программного обеспечения (далее - ПО)

Чтобы лучше разобраться в том, как тестирование соотносится с программированием и иными видами проектной деятельности, для начала рассмотрим самые основы —жизненный цикл ПО (SLC или Software lifecycle) и модели разработки ПО (SDLC или Software development lifecycle).

**Жизненным циклом программного обеспечения (SLC)** является период времени, начинающийся с момента появления концепции ПО и заканчивающийся тогда, когда использование ПО более невозможно. Жизненный цикл программного обеспечения обычно включает в себя следующие этапы: концепт, описание требований, дизайн, реализация, тестирование, инсталляция и наладка, эксплуатация и поддержка и, иногда, этап вывода из эксплуатации. Данные фазы могут накладываться друг на друга или проводиться итерационно.



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

Далее мы рассмотрим в общих чертах фазы жизненного цикла разработки системы, держа в уме, что все системы различны по уровню сложности, требуемым компонентам и ожидаемой функциональности.

#### **1. Планирование системы**

Фаза планирования - наиболее критичный шаг в создании успешной системы. Во время этой фазы решается, что необходимо сделать и какие проблемы решить, при помощи:

* определения проблем, целей и ресурсов (таких, как персонал и издержки);
* изучения возможностей альтернативных решений путем встреч с клиентами, поставщиками, консультантами и сотрудниками;
* изучения, как сделать ваш продукт лучше, чем у конкурентов.

После анализа этих данных будет три варианта: разработать новую систему, улучшить существующую или оставить систему как есть.

#### **2. Анализ системы**

На этом этапе определяется и фиксируются [требования](http://myalm.ru/glossary/%D0%A2%D1%80%D0%B5%D0%B1%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) конечного пользователя системы - в чем его ожидания и как их осуществить. Кроме того, для проекта делается технико-экономическое обоснование, которое выясняет, является ли проект организационно, экономически, социально, технологически осуществимым.

#### **3. Дизайн системы**

Фаза дизайна наступает после того, как достигнуто хорошее понимание требований потребителя. Эта фаза определяет элементы системы, компоненты, уровень безопасности, модули, архитектуру, различные интерфейсы и типы данных, которыми оперирует система. Дизайн системы в общих чертах может быть сделан ручкой на листке бумаги - он определяет, как система будет выглядеть и как функционировать. Затем делается расширенный, детальный дизайн, с учетом всех функциональных и [технических требований](http://myalm.ru/news/%D0%9E%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D0%BD%D0%B5%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D1%85-%D1%82%D1%80%D0%B5%D0%B1%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B9), как логически, так и физически.

#### **4. Разработка, внедрение и развертывание**

Эта фаза следует за полным пониманием системных требований и спецификаций. Это и есть собственно процесс разработки системы, когда дизайн системы уже полностью завершен и нагляден. В жизненном цикле разработки системы именно здесь пишется код, а если система включает апаратную часть, фаза внедрения будет включать в себя конфигурацию и настройку «железа» под определенные требования и функции. На этой стадии система готова к установке у заказчика, к запуску в боевом режиме. Возможно, конечным пользователям потребуется обучение, чтобы они освоились с системой и знали, как ее использовать. Фаза внедрения может быть очень долгой - это зависит от сложности системы.

#### **5. Опытная эксплуатация и интеграция**

Здесь происходит сборка различных компонентов и подсистем в одну целостную систему. Затем мы подаем системе различные входящие данные и анализируем выход, поведение и функционирование. Тестирование становится все важнее для удовлетворения потребителя, при этом оно не требует познаний ни в кодинге, ни в конфигурировании оборудования, ни в дизайне. Тестирование может выполняться настоящими пользователями или специальной командой сотрудников, также оно может быть систематическим и автоматизированным, с тем, чтобы удостовериться, что актуальные результаты работы системы совпадают с предусмотренными и желательными.

#### **6. Поддержка системы**

На этой фазе осуществляется периодическая техническая поддержка системы, чтобы убедиться, что система не устарела. Сюда входит замена старого оборудования и постоянная оценка производительности. Также здесь осуществляются апдейты определенных компонентов с целью удостовериться, что система отвечает нужным стандартам и новейшим технологиям, чтобы не быть подверженной текущим угрозам безопасности.

Это шесть основных стадий жизненного цикла разработки системы, и это повторяющийся процесс для каждого проекта. Важно отметить, что должен поддерживаться отличный уровень коммуникации с заказчиком.

# 3. Основные методологии разработки ПО

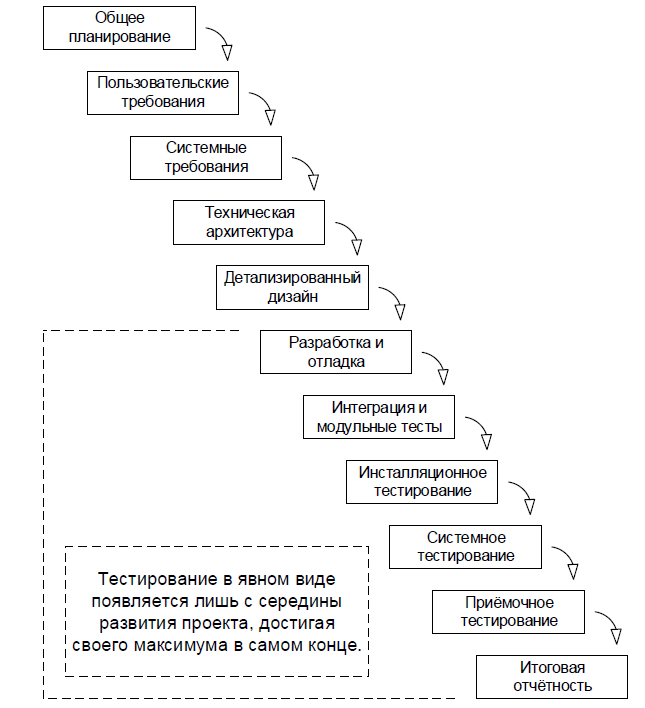
Выбор модели разработки ПО серьёзно влияет на процесс тестирования, определяя выбор стратегии, расписание, необходимые ресурсы и т.д.

Моделей разработки ПО много, но в общем случае классическими можно считать водопадную, v-образную, итерационную инкрементальную, спиральную и гибкую.

Знать и понимать модели разработки ПО нужно затем, чтобы уже с первых дней работы осознавать, что происходит вокруг, что, зачем и почему вы делаете. Многие начинающие тестировщики отмечают, что ощущение бессмысленности происходящего посещает их, даже если текущие задания интересны. Чем полнее вы будете представлять картину происходящего на проекте, тем яснее вам будет виден ваш собственный вклад в общее дело и смысл того, чем вы занимаетесь.

Ещё одна важная вещь, которую следует понимать, состоит в том, что никакая модель не является догмой или универсальным решением. Нет идеальной модели. Есть та, которая хуже или лучше подходит для конкретного проекта, конкретной команды, конкретных условий.

#### **1. Модель разработки: Водопадная модель (англ.Waterfall) (или каскадная)**

Самой старой и известной моделью построения многоуровневого процесса разработки является водопадная (или каскадная) модель: в ней каждый этап разработки, соответствующий стадии жизненного цикла ПО, продолжает предыдущий. То есть, для того, чтобы перейти на новый этап, мы полностью должны завершить текущий. 

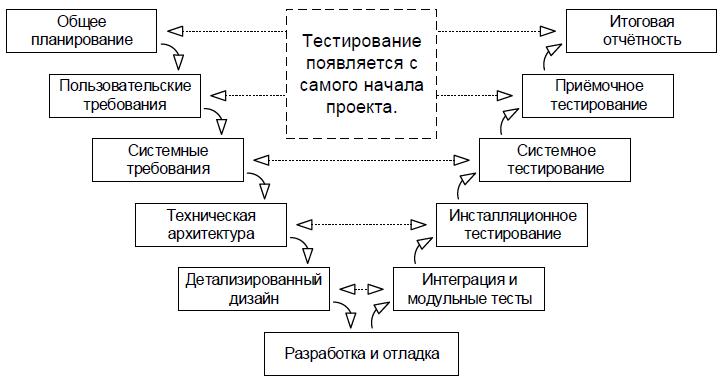
Водопадная модель проста и понятна, но не так практична, как раньше. В условиях динамично изменяющихся требований, строго структурированный процесс может из преимущества превратиться в помеху на пути успешного завершения разработки системы. С точки зрения тестирования эта модель плоха тем, что тестирование в явном виде появляется здесь лишь с середины развития проекта, достигая своего максимума в самом конце. Поэтому сегодня водопадная модель применяется преимущественно крупными компаниями для больших и сложных проектов, которые предполагают всеобъемлющий контроль рисков.

Видео про waterfall: <https://www.youtube.com/watch?v=i_F07BL2nhw>

#### **2. Модель разработки: V-образная модель**

V-образная модель (V-model) является логическим развитием водопадной. В общем случае как водопадная, так и v-образная модели жизненного цикла ПО могут содержать один и тот же набор стадий, но принципиальное отличие заключается в том, как эта информация используется в процессе реализации проекта.

Очень упрощённо можно сказать, что при использовании v-образной модели на каждой стадии «на спуске» нужно думать о том, что и как будет происходить на соответствующей стадии «на подъёме». Тестирование здесь появляется уже на самых ранних стадиях развития проекта, что позволяет минимизировать риски, а также обнаружить и устранить множество потенциальных проблем до того, как они станут проблемами реальными.

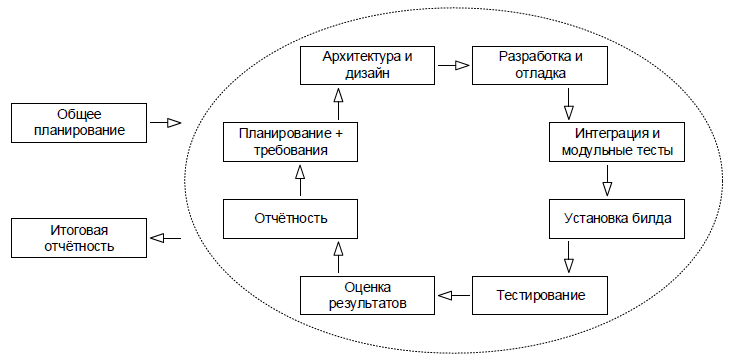


#### **3. Модель разработки: Итерационная инкрементальная модель**

Итерационная инкрементальная модель (англ. iterative model, incremental model) является фундаментальной основой современного подхода к разработке ПО. Как следует из названия модели, ей свойственна определённая двойстве-ность:

* с точки зрения жизненного цикла модель является **итерационной**, т.к. подразумевает многократное повторение одних и тех же стадий;
* с точки зрения развития продукта (приращения его полезных функций) модель является **инкрементальной**.

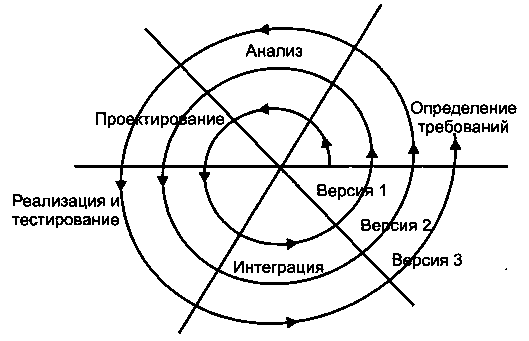
Ключевой особенностью данной модели является разбиение проекта на относительно небольшие промежутки (итерации), каждый из которых в общем случае может включать в себя все классические стадии, присущие водопадной и v-образной моделям. Итогом итерации является приращение (инкремент) функциональности продукта, выраженное в промежуточном билде



Длина итераций может меняться в зависимости от множества факторов, однако сам принцип многократного повторения позволяет гарантировать, что и тестирование, и демонстрация продукта конечному заказчику (с получением обратной связи) будет активно применяться с самого начала и на протяжении всего времени разработки проекта.

#### **4. Модель разработки: Спиральная модель**

Спиральная модель (spiral model) представляет собой частный случай итерационной инкрементальной модели, в котором особое внимание уделяется управлению рисками, в особенности влияющими на организацию процесса разработки проекта и контрольные точки. Все этапы разработки последовательно повторяются по кругу до тех пор, пока текущая версия программы не станет полностью соответствовать требованиям. Здесь также нужно иметь предел и вовремя остановиться.



#### **5. Модель разработки: Гибкая методология разработки (англ. Agile software development)**

Гибкая методология разработки, [agile](https://ru.wiktionary.org/wiki/agile#%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9)-методы — обобщающий термин для целого ряда подходов и практик, основанных на ценностях [Манифеста гибкой разработки программного обеспечения](https://ru.wikipedia.org/wiki/Agile_Manifesto) и 12 принципах, лежащих в его основе. К гибким методологиям, в частности, относят [экстремальное программирование](https://ru.wikipedia.org/wiki/%D0%AD%D0%BA%D1%81%D1%82%D1%80%D0%B5%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [DSDM](https://ru.wikipedia.org/wiki/DSDM), [Scrum](https://ru.wikipedia.org/wiki/Scrum), [FDD](https://ru.wikipedia.org/wiki/Feature_driven_development), [BDD](https://ru.wikipedia.org/wiki/BDD_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) и др.

**Scrum**— гибкий метод управления проектами, целью которого является повышение производительности труда в командах. Над каждым проектом работает универсальная команда специалистов, к которой присоединяется еще два человека: владелец продукта и scrum-мастер. Первый соединяет команду с заказчиком и следит за развитием проекта; это не формальный руководитель команды, а скорее куратор. Второй помогает первому организовать бизнес-процесс: проводит общие собрания, решает бытовые проблемы, мотивирует команду и следит за соблюдением scrum-подхода. В команде столько разноплановых специалистов, сколько нужно для решения любой задачи проекта. В основе концепции лежат «спринты». Спринт — короткая итерация, строго ограниченная по времени (обычно 2−4 недели). В это время минимизируется длительность совещаний, но увеличивается их частота.

Благодаря этому контроль за выполнением становится более гибким, а разработчики быстрее реагируют на возникающие проблемы. Традиционное планирование отходит на второй план, его место занимает журнал спринтов.

Видео про scrum: <https://www.youtube.com/watch?v=pN0auDS_8F8>

**Kanban**— метод управления [разработкой](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F), реализующий принцип «[точно в срок](https://ru.wikipedia.org/wiki/%D0%A2%D0%BE%D1%87%D0%BD%D0%BE_%D0%B2_%D1%81%D1%80%D0%BE%D0%BA)» и способствующий равномерному распределению нагрузки между работниками. При данном подходе весь процесс разработки прозрачен для всех членов команды. Задачи по мере поступления заносятся в отдельный список, откуда каждый разработчик может извлечь требуемую задачу. Над задачей может работать несколько узкопрофильных команд. К примеру, сначала работают аналитики, потом [дизайнеры рисуют](http://netology.ru/programs/web-design?utm_source=blog&utm_medium=747&utm_campaign=linkblog) прототип, а на третьем этапе включаются [разработчики](http://netology.ru/programs/node?utm_source=blog&utm_medium=747&utm_campaign=linkblog). В Kanban внутри команды нет ролей. Бизнес-процесс делится не на универсальные спринты, а на стадии выполнения конкретных задач: «Планируется», «Разрабатывается», «Тестируется», «Завершено» и др.

Видео про kanban: <https://www.youtube.com/watch?v=RzAPjmng97E>

Статья про отличия между Scrum и Kanban <https://netology.ru/blog/scrum-kanban>

# 4. Жизненный цикл тестирования ПО

В наше время, работа в командах регламентируется какими-то методологиями, и работа тестировщиков не исключение. Сегодня мы рассмотрим самую «прямую» и простую, и тем хорошую, методологию STLC.

Основной процесс тестирования состоит из следующих направлений

деятельности:

**• Планирование и управление**:

При планировании тестирования определяется цель тестирования и описываются задачи тестирования для достижения этих целей.

Управление тестированием – постоянное сопоставление текущих положения дел с планом, подготовка отчетности

**• Анализ требований**:

На данном этапе происходит анализ требований к ПО с целью выявить какие требования и как будут проверяться. Требования подразделяются на 4 направления:

1) Бизнес-требования – это требования самого высокого уровня. Они определяют цель создания продукта и задачи, которые необходимо выполнить для достижения цели.

2) Требования к архитектуре и дизайну – описывают элементы, интерфейсы, соединение элементов в структуру, а также характеристики, такие как гибкость, возможность реализации, безопасность и т.д

3) Функциональные требования – описывают действия, которые система должна быть способна выполнить

4) Критерии входа – основные условия, которые должны быть выполнены до начала тестирования. Цель требований – предотвратить начало тестирования перед окончанием подготовительных работ

**• Планирование**

На данном этапе определяется тест-стратегия с разными видами тестирования, определяется инструментарий и рассчитываются трудозатраты, определяются приоритеты проверок, распределяются роли и обязанности, описывается сам процесс тестирования. Подготавливаются тест-стратегия, тест-план, расчет рисков

**• Проектирование тестов**

На данном этапе описываются и согласовываются сценарии тестирования.

**• Настройка тестового окружения**

На данном этапе определяется программно-аппаратное обеспечение, при котором тестируется программный продукт. Например, настройка тестового сервера, настройка сети, тестовых устройств, формирование тестовых данных.

**• Выполнение тестирования**

На данном этапе выполняется процедура тестирования либо вручную, либо используя инструменты выполнения тестов, согласно плану тестирования; регистрируется результат проверок; выполняется подготовка отчета тестирования с описанием несоответствий, ошибок; после исправления - ретест всех ранее найденных ошибок.

**• Закрытие цикла**

На данном этапе подводятся итоги, оценка работы: стоимости, качества работы, охвата тестирования, затраченного времени, отправка отчетов тестирования и т.д.

Несмотря на логическую последовательность, действия в процессе могут накладываться друг на друга или происходить одновременно. Для конкретных системы и проекта обычно требуется адаптация этих направлений деятельности