**1.   Основы проекта, участники проектной команды**

* + Что такое проект

Проект - «Projectus» (лат.) – «брошенный вперёд»

Проект – это совокупность действий, в которых человеческие,

материальные и финансовые ресурсы организованы для выполнения

уникальных работ, определённого содержания в условиях

ограниченной стоимости и времени

«Проект» – это временное предприятие, предназначенное для создания

уникальных продуктов, услуг и результатов

Характеристики проекта:

* Нацеленность на достижение конкретной **цели**
* **Координированное** выполнение взаимосвязанных действий
* **Ограниченность** во времени выполнения, наличие конкретной

даты завершения

* Наличие **бюджета** ( финансового, материального и т.д)
* Неповторимость, **уникальность**, новизна

**Управление проектом** – это применение специальных знаний, навыков,

инструментов и методов для удовлетворения требований, предъявляемых

к проекту

**Виды стандартов в области управления проектами:**

* Международные стандарты в части методологии управления проектами:

PMBO (2008), ISO: 10006:2003

* Национальные стандарты в области управления проектами:

P2M, PRINCE 2, AFITER

* Корпоративные стандарты управления проектами: MSF, Oracle AIM, и т.д.

Существует несколько моделей разработки (жизненного цикла)

программного обеспечения. Но для начала давайте определимся с тем, что

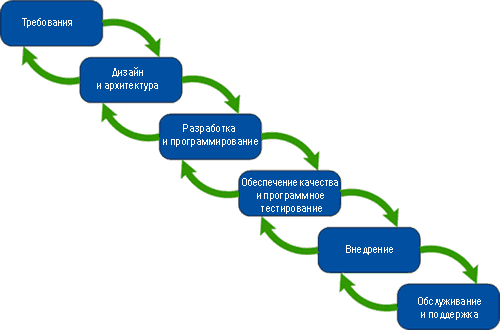
же это такое - жизненный цикл. В это понятие включаются все этапы,

которые проходит программный продукт. Первый будет - постановка

задачи, а последний - внедрение.

Тестирование – не изолированный процесс. Это часть модели жизненного цикла программного обеспечения (Software Development Life Cycle, SDLC). Именно поэтому выбор средств и методик тестирования будет напрямую зависеть от выбранной модели разработки. В этом разделе мы рассмотрим наиболее часто применяемые подходы к разработке программного обеспечения, а также популярные сегодня методологии и практики, такие как Agile и Scrum.

Жизненный цикл программного обеспечения (также называемый циклом разработки) – это условная схема, включающая отдельные этапы, которые представляют стадии процесса создания ПО. При этом на каждом этапе выполняются разные действия.



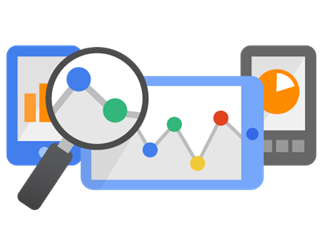
Цикл разработки предлагает шаблон, использование которого облегчает проектирование, создание и выпуск качественного программного обеспечения. Это методология, определяющая процессы и средства, необходимые для успешного завершения проекта.

Хотя реализация принципов построения модели жизненного цикла для разных компаний может существенно отличаться, существуют стандарты, такие как [ISO/IEC 12207](https://ru.wikipedia.org/wiki/ISO/IEC_12207:2008), определяющие принятые практики разработки и сопровождения программного обеспечения.

Цель использования модели жизненного цикла – создать эффективный, экономически выгодный и качественный программный продукт.

**Стадии цикла разработки ПО**

**1. Анализ требований**

****

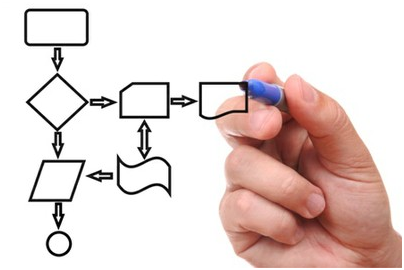
Жизненный цикл разработки ПО начинается со стадии анализа, во время которого участники процесса обсуждают требования, предъявляемые к конечному продукту. Цель этой стадии – определение детальных требований к системе. Кроме этого, необходимо убедиться  в том, что все участники правильно поняли поставленные задачи и то, как именно каждое требование будет реализовано на практике.

Зачастую, в обсуждении участвуют также и специалисты по тестированию, которые уже на стадии разработки требований могут вносить собственные пожелания и, при необходимости, корректировать процесс.

В зависимости от выбранной модели разработки, могут отличаться подходы к определению момента перехода с одной стадии на другую. К примеру, в каскадной или V-модели стадия анализа требований закрепляется в документе – спецификации требований к программному обеспечению (Software Requirement Specification, SRS), оформление которого должно быть закончено до перехода на следующую стадию.

Таким образом, этот этап предполагает **сбор требований** к разрабатываемому программному обеспечению, их **систематизацию**, **документирование**, **анализ**, а также **выявление и разрешение противоречий**.

**2. Проектирование**

****

На стадии проектирования (называемой также стадией дизайна и архитектуры) программисты и системные архитекторы, руководствуясь требованиями, разрабатывают высокоуровневый дизайн системы.

Разнообразные технические вопросы, возникающие в процессе проектирования, обсуждаются со всеми заинтересованными сторонами, включая заказчика. Определяются технологии, которые будут использоваться в проекте, загрузка команды, ограничения, временные рамки и бюджет. В соответствии с уточненными требованиями выбираются наиболее подходящие проектные решения.

Утвержденный дизайн системы определяет перечень разрабатываемых программных компонентов, взаимодействие с третьими сторонами, функциональные характеристики программы, используемые базы данных и многое другое. Дизайн, как правило, закрепляется отдельным документом – дизайн-спецификацией (Design Specification Document, DSD).

На этом этапе для упрощения визуализации процесса проектирования используются так называемые **нотации**– схематическое выражение характеристик разрабатываемой системы. Основные используемые нотации:

– [Блок-схемы](https://ru.wikipedia.org/wiki/%D0%91%D0%BB%D0%BE%D0%BA-%D1%81%D1%85%D0%B5%D0%BC%D0%B0);

– [ER-диаграммы](https://ru.wikipedia.org/wiki/ER-%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85);

– [UML-диаграммы](https://ru.wikipedia.org/wiki/UML);

– Макеты – например, нарисованный в фотошопе прототип сайта.

**3. Разработка и программирование**

****

После того как требования и дизайн продукта утверждены, происходит переход к следующей стадии жизненного цикла – непосредственно разработке. Здесь начинается написание программистами кода программы в соответствии с ранее определенными требованиями.

Системные администраторы настраивают программное окружение, front-end программисты разрабатывают пользовательский интерфейс программы и логику ее взаимодействия с сервером.

Кроме того, программисты пишут Unit-тесты для проверки правильности работы кода каждого компонента системы, проводят ревью написанного кода, создают билды и разворачивают готовое ПО в программной среде. Этот цикл повторяется до тех пор, пока все требования не будут реализованы.

Программирование предполагает четыре основных стадии:

**1)** Разработка [алгоритмов](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) – фактически, создание логики работы программы;

**2)** Написание исходного кода;

**3)** [Компиляция](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80) – преобразование в машинный код;

**4)** Тестирование и отладка – речь, главным образом, о юнит-тестировании.

**4. Документация**

****

Этот этап выделяют достаточно условно, поскольку, как мы видели, те или иные документы создаются на всех стадиях жизненного цикла программы. Тем не менее, помимо проектной документации и сопровождающих разработку записей, существуют также и другие текстовые документы, описывающие, например, функции программы и способы ее использования.

Всего существует **четыре уровня документации**:

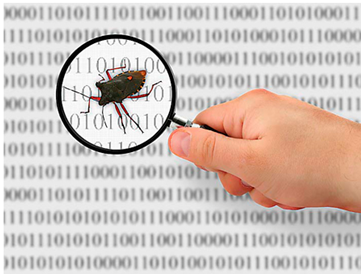
– **Архитектурная** (проектная) – например, дизайн-спецификация. Это документы, описывающие модели, методологии, инструменты и средства разработки, выбранные для данного проекта.

– **Техническая** – вся сопровождающая разработку документация. Сюда входят различные документы, поясняющие работу системы на уровне отдельных модулей. Как правило, пишется в виде комментариев к исходному коду, которые впоследствии структурируются в виде HTML-документов.

– **Пользовательская** – включает справочные и поясняющие материалы, необходимые конечному пользователю для работы с системой. Это, к примеру, Readme и Userguide, раздел справки по программе.

– **Маркетинговая** – включает рекламные материалы, сопровождающие выпуск продукта. Ее цель – в красочной форме представить функциональность и конкурентные преимущества продукта.

**5. Тестирование**

****

Основные этапы тестирования мы уже рассматривали ранее, в разделе [Фундаментальный процесс тестирования](http://qalight.com.ua/baza-znanij/fundamentalnyj-protsess-testirovaniya.html)

Тестировщики занимаются поиском дефектов в программном обеспечении и сравнивают описанное в требованиях поведение системы с реальным.

В фазе тестирования обнаруживаются пропущенные при разработке баги. При обнаружении дефекта, тестировщик составляет отчет об ошибке, который передается разработчикам. Последние его исправляют, после чего тестирование повторяется – но на этот раз для того, чтобы убедиться, что проблема была исправлена, и само исправление не стало причиной появления новых дефектов в продукте.

Тестирование повторяется до тех пор, пока не будут достигнуты [критерии его окончания](http://qalight.com.ua/baza-znanij/kogda-nachinat-i-zakanchivat-testirovanie.html)*.*

Виды, методы и техники тестирования мы подробно рассмотрим дальше в этом пособии.

**6. Внедрение и сопровождение**

****

Когда программа протестирована и в ней больше не осталось серьезных дефектов, приходит время релиза и передачи ее конечным пользователям.

После выпуска новой версии программы в работу включается отдел технической поддержки. Его сотрудники обеспечивают обратную связь с пользователями, их консультирование и поддержку.

В случае обнаружения пользователями тех или иных пост-релизных багов, информация о них передается в виде отчетов об ошибках команде разработки, которая, в зависимости от серьезности проблемы, либо немедленно выпускает исправление (т.н. hot-fix), либо откладывает его до следующей версии программы.

Кроме того, команда технической поддержки помогает собирать и систематизировать различные [метрики](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D1%80%D0%B8%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) – показатели работы программы в реальных условиях.

**Заключение**

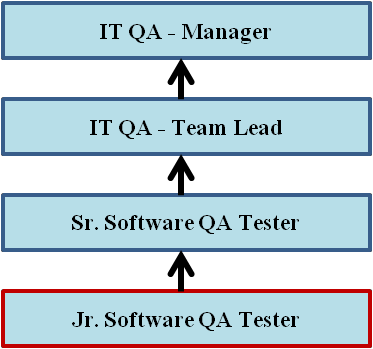
Все стадии жизненного цикла ПО, представленные выше, применяются в любой модели разработки, но их продолжительность и порядок следования могут отличаться.

* + **Участники проектной команды и их роли**

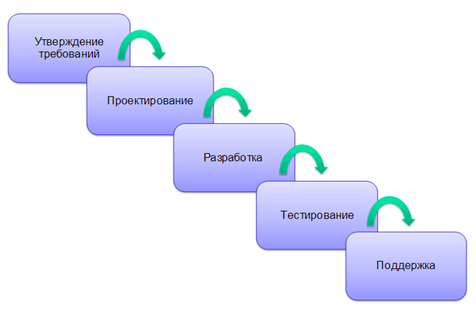
Хорошо сформированная проектная команда подразумевает, что каждый член команды понимает свою роль в формировании успеха проекта. Каждый член команды знает, какой вклад ему необходимо произвести в проект, когда ему необходимо это сделать, чем занимаются другие члены проекта, и что представляет собой успех. Столь же важно, чтобы каждый член команды оказывал поддержку остальным, и таким образом гарантировал успех всего проекта.



|  |  |
| --- | --- |
| **Проектная роль** | **Описание проектной роли** |
| Руководитель проекта (Project Manager) | Руководитель проектной команды, ответственный за управление проектом, достижение целей проекта в рамках бюджета, в срок и с заданным уровнем качества. Организовывает необходимое взаимодействие и способен разрешать проблемы, возникающие по ходу выполнения проекта |
| Системный архитектор (SA) | Формирует архитектуру и идеологию проекта, оценку имеющихся технических возможностей, взаимосвязи со смежными системами, проектную документацию. |
| Бизнес-аналитик  (BA) | Обеспечивает сбор требований, их обработку, документирование и передачу специалистам-разработчикам |
| Администратор БД  (DBA) | Проводит установку и настройку СУБД. Непосредственно участвует в разработке физической модели данных. Отвечает за выработку требований к БД, её проектирование, реализацию, оптимизацию, сопровождение и архивирование, включая управление учётными записями пользователей БД и защиту от несанкционированного доступа. |
| Администратор приложений (SysAdmin) | Обеспечивает установку и настройку сервера приложений и его компонентов, разработку дополнительных приложений. Обеспечивает разграничение прав доступа на прикладном уровне. |
| Лидер команды разработчиков (TeamLead) | Организовывает процесс *разработки* на проекте, распределяет задачи, контролирует их выполнение, проводит «код ревью» и оценивает эффективность работы команды. |
| Технический лидер- разработчик (TechLead) | Разрабатывает дизайны компонентов и их взаимодействие |
| Разработчик приложений (Developer) | Разрабатывает, развивает и сопровождает приложения |
| Лидер команды тестировщиков (QA-lead) | Организовывает процесс *тестирования* на проекте, распределяет задачи, контролирует их выполнение, проводит «ревью тестовой документации» и оценивает эффективность работы команды. |
| Тестировщик (QA  engineer) | Разрабатывает тест-дизайны, проводит тестирование |



**Каскадная модель (Waterfall model)**



Однажды, на заре эпохи программирования, единственными участниками процесса разработки были программисты. Они писали программные функции для облегчения математических расчетов или автоматизации других рутинных действий. Сегодня все иначе. Современные системы настолько большие и сложные, что над их созданием трудятся целые команды специалистов разного профиля: программисты, аналитики, системные администраторы, тестировщики и конечные пользователи. Все они работают вместе для разработки программ, содержащих миллионы строк кода.

Самой старой и известной моделью построения многоуровневого процесса разработки является каскадная (или попросту водопадная) 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.

Каскадная 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. Поэтому сегодня водопадная модель применяется преимущественно крупными компаниями для больших и сложных проектов, которые предполагают всеобъемлющий контроль рисков.

Плюсы и минусы каскадной модели:

+ Полное документирование каждого этапа;

+ Четкое планирование сроков и затрат;

+ Прозрачность процессов для заказчика;

- Необходимость утверждения полного объема требований к системе еще на первом этапе;

- В случае необходимости внесения изменений требований позднее – возврат к первой стадии и переделка заново всей проделанной работы;

- Увеличение затрат средств и времени в случае необходимости изменения требований.

Несмотря на то, что каскадная модель все еще используется, она уже утратила былые позиции. Сегодня ей на смену приходят более продвинутые модели и методологии разработки программного обеспечения.

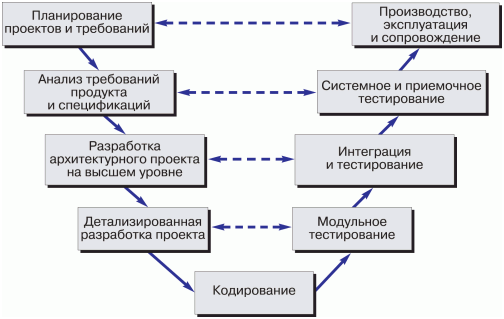
Когда использовать каскадную модель:

– В проектах с четко определенными требованиями, для которых не предусматривается их изменений в процессе разработки;

– Для проектов, которые мигрируют с одной платформы на другую. То есть, требования остаются те же, меняется только системное окружение и/или язык программирования;

– Когда от компании-разработчика не требуется проводить тестирования – к примеру, его обеспечением займется сам заказчик или сторонняя фирма.

**V-модель (V-model)**



V-модель – это улучшенная версия классической каскадной модели. Здесь на каждом этапе происходит контроль текущего процесса, для того чтобы убедится в возможности перехода на следующий уровень. В этой модели тестирование начинается еще со стадии написания требований, причем для каждого последующего этапа предусмотрен свой уровень тестового покрытия.

Для каждого уровня тестирования разрабатывается отдельный тест-план, то есть во время тестирования текущего уровня, мы также занимаемся разработкой стратегии тестирования следующего. Создавая тест-планы, мы также определяем ожидаемые результаты тестирования и указываем критерии входа и выхода для каждого этапа.

В V-модели каждому этапу проектирования и разработки системы соответствует отдельный уровень тестирования. Здесь процесс разработки представлен нисходящей последовательностью в левой части условной буквы V, а стадии тестирования – на ее правом ребре. Соответствие этапов разработки и тестирования показано горизонтальными линиями.

Плюсы и минусы V-модели:

+ строгая этапизация;

+ планирование тестирования и верификация системы производятся на ранних этапах;

+ улучшенный, по сравнению с каскадной моделью, тайм-менеджмент;

+ промежуточное тестирование.

- недостаточная гибкость модели;

- собственно создание программы происходит на этапе написания кода, то есть уже в середине процесса разработки;

- недостаточный анализ рисков;

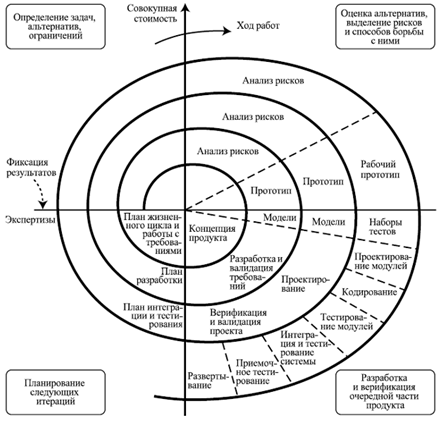
- нет работы с параллельными событиями и возможности динамического внесения изменений.

Когда использовать V-модель:

– В проектах, в которых существуют временные и финансовые ограничения;

– Для задач, которые предполагают более широкое, по сравнению с каскадной моделью, тестовое покрытие.

**Спиральная модель (Spiral model)**



Спиральная модель представляет шаблон процесса разработки ПО, который сочетает идеи итеративной и каскадной моделей. Суть ее в том, что весь процесс создания конечного продукта представлен в виде условной плоскости, разбитой на 4 сектора, каждый из которых представляет отдельные этапы его разработки: определение целей, оценка рисков, разработка и тестирование, планирование новой итерации.

В спиральной модели жизненный путь разрабатываемого продукта изображается в виде спирали, которая, начавшись на этапе планирования, раскручивается с прохождением каждого следующего шага. Таким образом, на выходе из очередного витка мы должны получить готовый протестированный прототип, который дополняет существующий билд. Прототип, удовлетворяющий всем требованиям – готов к релизу.

Главная особенность спиральной модели – концентрация на возможных рисках. Для их оценки даже выделена соответствующая стадия. Основные типы рисков, которые могут возникнуть в процессе разработки ПО:

- Нереалистичный бюджет и сроки;

- Дефицит специалистов;

- Частые изменения требований;

- Чрезмерная оптимизация;

- Низкая производительность системы;

- Несоответствие уровня квалификации специалистов разных отделов.

Плюсы и минусы спиральной модели:

+ улучшенный анализ рисков;

+ хорошая документация процесса разработки;

+ гибкость – возможность внесения изменений и добавления новой функциональности даже на относительно поздних этапах;

+ раннее создание рабочих прототипов.

- может быть достаточно дорогой в использовании;

- управление рисками требует привлечения высококлассных специалистов;

- успех процесса в большой степени зависит от стадии анализа рисков;

- не подходит для небольших проектов.

Когда использовать спиральную модель:

– когда важен анализ рисков и затрат;

– крупные долгосрочные проекты с отсутствием четких требований или вероятностью их динамического изменения;

– при разработке новой линейки продуктов.

**Итеративная модель (Iterative model)**



Не все модели жизненного цикла последовательны. Существуют также итеративные (или инкрементальные) модели, в которых используется другой подход. Вместо одной продолжительной последовательности действий здесь весь жизненный цикл продукта разбит на ряд отдельных мини-циклов. Причем каждый из них состоит из все тех же базовых стадий модели жизненного цикла. Эти мини-циклы называются итерациями. В каждой из итераций происходит разработка отдельного компонента системы, после чего этот компонент добавляется к уже ранее разработанному функционалу.

Итеративная модель не предполагает полного объема требований для начала работ над продуктом. Разработка программы может начинаться с требований к части функционала, которые могут впоследствии дополняться и изменяться.  Процесс повторяется, обеспечивая создание новой версии продукта для каждого цикла.

В несколько упрощенном виде, итеративная модель состоит из четырех основных стадий, которые повторяются в каждой из итераций (plan-do-check-act):

– определение и анализ требований;

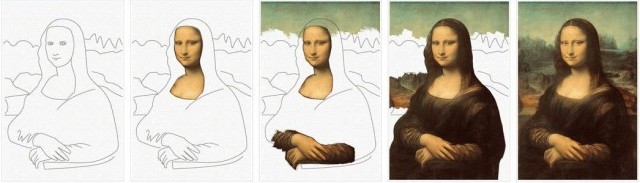
– дизайн и проектирование – согласно требованиями. Причем дизайн может как разрабатываться отдельно для данной функциональности, так и дополнять уже существующий;

– разработка и тестирование – кодирование, интеграция и тестирование нового компонента;

– фаза ревью – оценка, пересмотр текущих требований и предложения дополнений к ним.

По результатам каждой итерации принимается решение – будут ли использованы ее результаты для дополнения существующей функциональности в качестве входной точки для начала следующей итерации (т.н. инкрементальное прототипирование). В конечном итоге, достигается точка, в которой все требования были воплощены в продукте – происходит релиз.

В математических терминах, итеративная модель представляет реализацию методики  последовательной аппроксимации – то есть, постепенное приближение к образу готового продукта:



Ключ к успешному использованию этой модели – строгая валидация требований и тщательная верификация разрабатываемой функциональности в каждой из итераций.

Основные стадии процесса разработки в итеративной модели фактически повторяют модель водопада. В каждой итерации создается программное обеспечение, требующее тестирования на всех уровнях.

Плюсы и минусы итеративной модели:

+ раннее создание работающего ПО;

+ гибкость – готовность к изменению требований на любом этапе разработки;

+ каждая итерация – маленький этап, для которого тестирование и анализ рисков обеспечить проще, чем для всего жизненного цикла продукта.

- каждая фаза – самостоятельна, отдельные итерации не накладываются;

- могут возникнуть проблемы с реализацией общей архитектуры системы, поскольку не все требования известны к началу проектирования.

Когда использовать итеративную модель:

– для крупных проектов;

– когда известны, по крайней мере, ключевые требования;

– когда требования к проекту могут меняться в процессе разработки.

Итеративная модель ­является ключевым элементом так называемых «гибких» (Agile) подходов к разработке программного обеспечения, основные из которых мы рассмотрим в дальнейшем.