**Жизненный цикл программного обеспечения** (ПО) — период времени, который начинается с момента принятия решения о необходимости создания программного продукта и заканчивается в момент его полного изъятия из эксплуатации[1]. Этот цикл — процесс построения и развития ПО.

**Модель жизненного цикла ПО** — структура, определяющая последовательность выполнения и взаимосвязи процессов, действий и задач на протяжении жизненного цикла. Модель жизненного цикла зависит от специфики, масштаба и сложности проекта и специфики условий, в которых система создается и функционирует.

## Водопадная (каскадная, последовательная) модель

Основная статья: **Модель водопада**

Водопадная модель жизненного цикла (англ. *waterfall model*) была предложена в 1970 г. Уинстоном Ройсом. Она предусматривает последовательное выполнение всех этапов проекта в строго фиксированном порядке. Переход на следующий этап означает полное завершение работ на предыдущем этапе. Требования, определенные на стадии формирования требований, строго документируются в виде технического задания и фиксируются на все время разработки проекта. Каждая стадия завершается выпуском полного комплекта документации, достаточной для того, чтобы разработка могла быть продолжена другой командой разработчиков.

Этапы проекта в соответствии с каскадной моделью:

1. Формирование требований;
2. Проектирование;
3. Реализация;
4. Тестирование;
5. Внедрение;
6. Эксплуатация и сопровождение.

**Преимущества**:

- Полная и согласованная документация на каждом этапе;
- Легко определить сроки и затраты на проект.

**Недостатки**:

В водопадной модели переход от одной фазы проекта к другой предполагает полную корректность результата (выхода) предыдущей фазы. Однако неточность какого-либо требования или некорректная его интерпретация в результате приводит к тому, что приходится «откатываться» к ранней фазе проекта и требуемая переработка не просто выбивает проектную команду из графика, но приводит часто к качественному росту затрат и, не исключено, к прекращению проекта в той форме, в которой он изначально задумывался. По мнению современных специалистов, основное заблуждение авторов водопадной модели состоит в предположениях, что проект проходит через весь процесс один раз, спроектированная архитектура хороша и проста в использовании, проект

осуществления разумен, а ошибки в реализации легко устраняются по мере тестирования. Эта модель исходит из того, что все ошибки будут сосредоточены в реализации, а потому их устранение происходит равномерно во время тестирования компонентов и системы[2]. Таким образом, водопадная модель для крупных проектов мало реалистична и может быть эффективно использована только для создания небольших систем

## Спиральная модель

**Спиральная модель** (англ. *spiralmodel*) была разработана в середине 1980-х годов Барри Боэмом. Она основана на классическом цикле Деминга PDCA (plan-do-check-act). При использовании этой модели ПО создается в несколько итераций (витков спирали) методом прототипирования.

Каждая итерация соответствует созданию фрагмента или версии ПО, на ней уточняются цели и характеристики проекта, оценивается качество полученных результатов и планируются работы следующей итерации.

На каждой итерации оцениваются:

- риск превышения сроков и стоимости проекта;
- необходимость выполнения ещё одной итерации;
- степень полноты и точности понимания требований к системе;
- целесообразность прекращения проекта.

Важно понимать, что спиральная модель является не альтернативой эволюционной модели (модели IID), а специально проработанным вариантом. К сожалению, нередко спиральную модель либо ошибочно используют как синоним эволюционной модели вообще, либо (не менее ошибочно) упоминают как совершенно самостоятельную модель наряду с IID[3].

Отличительной особенностью спиральной модели является специальное внимание, уделяемое рискам, влияющим на организацию жизненного цикла, и контрольным точкам. Боэм формулирует 10 наиболее распространённых (по приоритетам) рисков:

1. Дефицит специалистов.
2. Нереалистичные сроки и бюджет.
3. Реализация несоответствующей функциональности.
4. Разработка неправильного пользовательского интерфейса.
5. Перфекционизм, ненужная оптимизация и оттачивание деталей.
6. Непрекращающийся поток изменений.
7. Нехватка информации о внешних компонентах, определяющих окружение системы или вовлеченных в интеграцию.
8. Недостатки в работах, выполняемых внешними (по отношению к проекту) ресурсами.
9. Недостаточная производительность получаемой системы.
10. Разрыв в квалификации специалистов разных областей.