Оглавление

[Введение 1](#_Toc185778164)

[https://www.informicus.ru/items-waterfall.html Водопадная модель жизненного цикла 8](#_Toc185778165)

[https://worksection.com/blog/waterfall.html Бессмертная классика Waterfall 8](#_Toc185778166)

## Введение

##### В современном мире разработки программного обеспечения существует множество подходов и методологий, которые помогают организовать процесс создания программных продуктов.

1970

год, 1

Уинстон

1970, 1, 3

##### Одним из самых известных и традиционных методов является водопадный жизненный цикл. Этот подход, основанный на последовательном выполнении этапов разработки, был предложен в 1970-х годах и до сих пор широко используется в различных проектах. В данном реферате мы рассмотрим основные этапы водопадного жизненного цикла, его преимущества и недостатки, а также области применения.

**Жизненным циклом ПО** называется период существования ПО, связанный с подготовкой к его разработке, разработкой, использованием и переработками, начиная с того момента, когда принимается решение разработать новую систему до того момента, когда полностью прекращается всякое ее использование. Модель жизненного цикла ПО выделяет конкретные наборы видов деятельности, артефактов, ролей и их взаимосвязи.

**Водопадная модель жизненного цикла ПО** (англ. *waterfall model)*

##### Водопадная модель была предложена в 1970 году ом Ройсом в его статье "Managing the Development of Large Software Systems". Метод предполагает последовательное выполнение различных этапов деятельности, включая анализ требований, проектирование, кодирование и тестирование отдельных модулей (компонентов), тестирование сборок и интегрированное тестирование всего конечного продукта. При этом предполагается четкое разграничение этапов, на которых набор документов, выработанный на предыдущей этапе, передается в качестве входных данных для следующего. Таким образом, каждый вид деятельности выполняется на какой-то одной фазе жизненного цикла ПО, движение в обратную сторону по этой цепочке невозможно.

**ФУНКЦИИ РАЗРАБОТКИ КОМПЬЮТЕРНЫХ ПРОГРАММ**

Существует два основных этапа, общих для всех разработок компьютерных программ, независимо от размера или сложности. Сначала идет этап анализа, за которым следует этап кодирования. Такая очень простая концепция внедрения на самом деле все, что требуется, если усилия достаточно малы и, если конечный продукт должен эксплуатироваться теми, кто его создал, как это обычно делается с компьютерными программами для внутреннего использования. Это также тот тип усилий по разработке, за который большинство клиентов с радостью платят, поскольку оба этапа включают в себя действительно творческую работу, которая напрямую способствует полезности конечного продукта. Однако план внедрения для производства более крупных программных систем, привязанный только к этим этапам, обречен на провал. Требуется много дополнительных этапов разработки, ни один из которых не вносит такого прямого вклада в конечный продукт, как анализ и кодирование, и все они увеличивают стоимость разработки. Персонал заказчика, как правило, предпочел бы не платить за них, а персонал разработки предпочел бы не внедрять их. Основная функция управления — продать эти концепции обеим группам, а затем обеспечить соблюдение требований со стороны персонала разработки.

Более грандиозный подход к разработке программного обеспечения существует в виде “Системные требования => требования к программному обеспечению => анализ => дизайн программы => написание кода => тестирование => операции”.

Итеративная связь между последовательными фазами разработки. Порядок шагов основан на следующей концепции: по мере продвижения каждого шага и дальнейшей детализации дизайна происходит итерация с предыдущими и последующими шагами, но редко с более отдаленными шагами в последовательности. Достоинство всего этого в том, что по мере продвижения дизайна процесс изменений сводится к управляемым пределам. В любой точке процесса проектирования после завершения анализа требований существует твердая и близкая, движущаяся базовая линия, к которой можно вернуться в случае непредвиденных трудностей проектирования.

Описанная выше реализация рискованна и ведет к неудаче. Проблема происходит при тестировании и возвращении к стадии дизайна. Фаза тестирования, которая происходит в конце цикла разработки, является первым событием, для которого синхронизация, хранение, передача ввода/вывода и т. д. воспринимаются как отличные от анализа. Эти явления не поддаются точному анализу. Они не являются решениями стандартных уравнений в частных производных математической физики, например. Тем не менее, если эти явления не удовлетворяют различным внешним ограничениям, то неизменно требуется серьезная переработка. Требуемые изменения в дизайне, вероятно, будут настолько разрушительными, что требования к программному обеспечению, на которых основан дизайн и которые обеспечивают обоснование всего, будут нарушены. Либо требования должны быть изменены, либо требуется существенное изменение дизайна. По сути, процесс разработки вернулся к исходной точке, и можно ожидать до 100-процентного превышения графика и/или затрат.

Уинстон Ройс считает, что проиллюстрированный подход в основе своей надежен. Оставшаяся часть этого обсуждения представляет пять дополнительных функций, которые необходимо добавить к этому базовому подходу, чтобы устранить большинство рисков разработки.

**Типовые фазы водопадного процесса**

1. **Анализ** (определение требований, область проблемы). На этапе анализа изучается и определяется задача, которую должна выполнять программа. Результатом выполнения этой фазы является совокупность требований, предъявляемых к ПО.
2. **Проектирование** (архитектура, область решения). На этом этапе требования, выявленные при анализе, преобразуются в описание принципов решения – документ, в соответствии с которым принимаются конкретные решения при реализации программы. Основным итогом второй фазы является получение проекта, который может включать текст на естественном языке, модель ПО, алгоритмы, таблицы, математические формулы и т. п. Детальное проектирование предполагает выделение компонент ПО, определение их структуры и методов взаимодействия.
3. **Разработка** (реализация, конструирование, кодирование) **-** на этом этапе происходит непосредственная разработка программного обеспечения. Программисты пишут код согласно спецификациям и проекту, созданному на предыдущем этапе. Важно соблюдать стандарты кодирования и проводить регулярные проверки качества кода.
4. **Интеграция -** процесс разработки и внедрения программного обеспечения, с помощью которого отдельные компоненты могут быть связаны в единую систему
5. **Тестирование и отладка** (верификация) - процесс исследования, испытания программного продукта, имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, к примеру, unit-тесты, integration-тесты
6. **Установка** (deploy, инсталляция) – После успешного тестирования продукт готов к внедрению. На этом этапе программное обеспечение устанавливается на рабочие места пользователей, проводятся обучающие сессии и предоставляется техническая поддержка.
7. **Передача в эксплуатацию** (сдача-приемка, вывод) – Готовый программный продукт передается заказчику, производятся приемо-сдаточные испытания, осуществляется обучение пользователей и опытная эксплуатация, после чего ПО ставится на сопровождение и начинается производственная эксплуатация программной системы.
8. **Поддержка** (support) – программное обеспечение получает поддержку в виде исправления багов и минорных изменений

**ШАГ 1: РАЗРАБОТКА ПРОГРАММЫ**

Предварительная фаза проектирования программы была вставлена между фазой генерации требований к программному обеспечению и фазой анализа. Эту процедуру можно критиковать на том основании, что разработчик программы вынужден проектировать в относительном вакууме начальных требований к программному обеспечению без какого-либо существующего анализа. В результате его предварительный проект может оказаться существенно ошибочным по сравнению с его проектом, если бы он ждал завершения анализа. Эта критика верна, но она упускает суть. С помощью этого метода разработчик программы гарантирует, что программное обеспечение не выйдет из строя из-за проблем с хранением, синхронизацией и потоком данных. По мере продолжения анализа на следующей фазе разработчик программы должен наложить на аналитика ограничения по хранению, синхронизации и эксплуатации таким образом, чтобы он чувствовал последствия. Когда ему обоснованно требуется больше такого рода ресурсов для реализации своих уравнений, они должны быть одновременно изъяты у его коллег-аналитиков. Таким образом, все аналитики и все разработчики программ внесут свой вклад в осмысленный процесс проектирования, который завершится правильным распределением времени выполнения и ресурсов хранения. Если общих ресурсов, которые необходимо задействовать, недостаточно или если начальный операционный проект неверен, это будет выявлено на этой ранней стадии, и итерация с требованиями и предварительным проектированием может быть переделана до начала окончательного проектирования, кодирования и тестирования.

Как реализуется эта процедура? Необходимо выполнить следующие шаги.

1) Начинайте процесс проектирования с проектировщиков программ, а не с аналитиков или программистов.

2) Спроектировать, определить и выделить режимы обработки данных, даже рискуя ошибиться. Выделить обработку, функции, спроектировать базу данных, определить обработку хэша данных, выделить время выполнения, определить интерфейсы и режимы обработки с операционной системой, описать обработку ввода и вывода и определить предварительные рабочие процедуры.

3) Напишите обзорный документ, который понятен, информативен и актуален. Каждый работник должен иметь элементарное понимание системы.

**ШАГ 2 ДОКУМЕНТИРОВАНИЕ ДИЗАЙНА**

В этот момент уместно поднять вопрос "сколько документации?" Мое собственное мнение - "довольно много", определенно больше, чем большинство программистов, аналитиков или проектировщиков программ готовы признать, если предоставить их самим себе. Первое правило управления разработкой программного обеспечения - беспощадное соблюдение требований к документации.

Зачем столько документации?

1) Каждый дизайнер должен общаться с дизайнерами интерфейсов, со своим руководством и, возможно, с заказчиком. Устная запись слишком неосязаема, чтобы обеспечить адекватную основу для решения по интерфейсу или управлению

2) На ранней стадии разработки ПО документация — это спецификация и дизайн. Пока не началось кодирование, эти три существительных (документация, спецификация, дизайн) обозначают одно и то же.

3) Реальная денежная стоимость хорошей документации начинается в процессе разработки на этапе тестирования и продолжается в ходе эксплуатации и перепроектирования.

a) На этапе тестирования, при наличии хорошей документации, менеджер может сконцентрировать персонал на ошибках в программе.

b) Во время фазы эксплуатации, при наличии хорошей документации менеджер может использовать ориентированный на эксплуатацию персонал для управления программой и выполнения лучшей работы, дешевле.

c) После начальных операций, когда улучшения системы в порядке, хорошая документация позволяет эффективно перепроектировать, обновить и модернизировать на месте. Если документации нет, то, как правило, вся существующая структура операционного программного обеспечения должна быть выброшена, даже для относительно скромных изменений.

**ШАГ 3: СДЕЛАЙТЕ ЭТО ДВАЖДЫ**

После документации второй по важности критерий успеха вращается вокруг того, является ли продукт полностью оригинальным. Если рассматриваемая компьютерная программа разрабатывается впервые, организуйте все так, чтобы версия, окончательно предоставленная заказчику для оперативного развертывания, фактически была второй версией, поскольку это касается критических областей проектирования/эксплуатации. Обратите внимание, что это просто весь процесс, выполненный в миниатюре, в масштабе времени, который относительно мал по сравнению с общими усилиями. Характер этих усилий может сильно различаться в зависимости, прежде всего, от общего масштаба времени и характера критических проблемных областей, которые должны быть смоделированы. Если усилия длятся 30 месяцев, то эта ранняя разработка пилотной модели может быть запланирована на 10 месяцев. Для этого графика могут использоваться довольно формальные средства контроля, процедуры документирования и т. д. Однако, если общие усилия были сокращены до 12 месяцев, то пилотные усилия могли бы быть сжаты до трех месяцев, возможно для того, чтобы получить достаточный рычаг для основной разработки. В этом случае от вовлеченного персонала требуется совершенно особый вид широкой компетенции. Они должны иметь интуитивное чутье для анализа, кодирования и проектирования программ.

**ШАГ 4: ПЛАНИРОВАНИЕ, КОНТРОЛЬ И МОНИТОРИНГ ТЕСТИРОВАНИЯ**

Без сомнения, самым большим потребителем ресурсов проекта, будь то рабочая сила, компьютерное время или управленческое суждение, является фаза тестирования. Это фаза наибольшего риска с точки зрения денег и графика. Она происходит в последней точке графика, когда резервные альтернативы наименее доступны, если вообще доступны.

Предыдущие три рекомендации по проектированию программы до начала анализа и кодирования, ее полному документированию и созданию пилотной модели направлены на выявление и решение проблем до начала фазы тестирования. Однако даже после выполнения этих действий все еще есть фаза тестирования и есть еще важные вещи, которые нужно сделать.

1) Многие части процесса тестирования лучше всего выполняются специалистами по тестированию, которые не обязательно принимали участие в разработке исходного проекта.

2) Большинство ошибок имеют очевидный характер и могут быть легко обнаружены при визуальном осмотре.

3) Проверьте каждый логический путь в компьютерной программе хотя бы один раз с помощью какой-либо числовой проверки.

**ШАГ 5. ВОВЛЕЧЕНИЕ КЛИЕНТА**

По какой-то причине то, что будет делать дизайн программного обеспечения, подлежит широкому толкованию даже после предварительного соглашения. Важно вовлекать заказчика формальным образом, чтобы он взял на себя обязательства на более ранних этапах до окончательной поставки. Давать подрядчику свободу действий между определением требований и эксплуатацией означает навлекать проблемы.

##### **Недостатки водопадного подхода**

* Накопление различных ошибок, допущенных на ранних стадиях проекта. Если только к концу проекта, становится очевидно, что были допущены ошибки, то любой возврат к предыдущим стадиям с целью исправления ошибок становится крайне дорогостоящим. Метод "водопада" не позволяет эффективно выявлять и нивелировать последствия подобных рисков.
* Неоправданное увеличение времени реализации, превышение бюджета и риск полного срыва проекта из-за накопления ошибок от этапа к этапу.
* Метод водопада не дает возможности быстрой адаптации к изменениям, особенно на поздних стадиях жизненного цикла ПО.
* Конечный продукт может оказаться невостребованным из-за неточного изложения требований или их изменения за длительное время создания ПО.

Компании, использующие водопадную модель: Cisco, EPAM, IBM, AT Consulting, Parallels, SAP, Toyota

Список литературы:

## <https://www.informicus.ru/items-waterfall.html> Водопадная модель жизненного цикла

# <https://worksection.com/blog/waterfall.html> Бессмертная классика Waterfall

<https://web.archive.org/web/20160318002949/http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf>