Оглавление

[Введение (ТРПО) 2](#_Toc506053579)

[Глава 1 4](#_Toc506053580)

[Процесс разработки программного обеспечения 4](#_Toc506053581)

[Жизненный цикл программного обеспечения 8](#_Toc506053582)

[Модели жизненного цикла ПО 9](#_Toc506053583)

[1 Водопадная модель 9](#_Toc506053584)

[2 Итерационная модель 11](#_Toc506053585)

[3 Спиральная модель 12](#_Toc506053586)

[4 V-Model 14](#_Toc506053587)

[Методологии 17](#_Toc506053588)

# Введение (ТРПО)

**Технология разработки программного обеспечения (ТРПО**) – это совокупность процессов и методов создания программного продукта.

**Технология разработки программного обеспечения** – это система инженерных принципов для создания экономичного ПО, которое надежно и эффективно работает в реальных компьютерах. Данное определение имеет частный характер, поскольку учитывает только две из шести характеристик качества ПО – надежность и эффективность. С учетом этого можно сформулировать более общее определение.

**Технология разработки программного обеспечения** – это система инженерных принципов для создания экономичного ПО с заданными характеристиками качества. Любая технология разработки ПО базируется на некоторой методологии.

Под **методологией** понимается система принципов и способов организации процесса разработки программ. Цель методологии разработки ПО – внедрение методов разработки программ, обеспечивающих достижение соответствующих характеристик качества.

Термин «**технология**» подчеркивает аналогию между созданием программного продукта и промышленным производством. Он отражает современную тенденцию к вводу дисциплины, организации и инструментирования в такой творческий процесс, как программирование. Слово фиксирует ту точку зрения, что программирование, несмотря на интеллектуальность и творческий характер этой деятельности, нуждается в организации и регламентировании, наборе соглашений и правил, не говоря уже об инструментальном обеспечении. Сейчас это кажется тривиальным, но в 60-е годы такую точку приходилось отстаивать. Да и сейчас порой возникают трения на почве регламентирования деятельности разработчиков. Сам русский термин «технология программирования» был введен русским академиком Андреем Петровичем Ершовым. Он трактовал термин «программирование» в обобщенном виде и подразумевал все виды деятельности, выполняемые в ходе создания программных систем. На западе для определения этой деятельности использовался термин «engineering». Сейчас обобщённый термин, применимый к созданию программных средств, обозначают как «разработка» или «конструирование». Справедлива формула: **разработка = анализ + проектирование + программирование (кодирование) + тестирование + отладка**. Иногда сюда также включают “сопровождение”. Чтобы подчеркнуть промышленно-производственный аспект, говорят о “технологии разработки” или “технологии конструирования”. [1]

**Программное обеспечение (ПО)** – компьютерные программы и соответствующая документация. Разрабатывается по частному заказу или для продажи на рынке ПО.

**Процесс создания ПО** – совокупность процессов, приводящих к созданию программного продукта.

Процесс современной разработки программного обеспечения ориентирован на жизненный цикл программного продукта. Все существующие в настоящее время технологии, методики и стандарты напрямую или косвенно касаются или регламентируют этапы жизненного цикла, как по функциональному наполнению, так и по содержанию.

Процесс разработки программных систем тесно связан с областью управления проектами, потому что любой программный продукт является уникальным результатом. От организации этого процесса напрямую зависят основные характеристики выполнения программного проекта – сроки выполнения, запланированный бюджет, качество выпускаемого продукта.

# Глава 1

## Процесс разработки программного обеспечения

Процесс разработки состоит из множества подпроцессов, или дисциплин, некоторые из которых показаны ниже. В модели водопада они идут одна за другой, в других аналогичных процессах их порядок или состав изменяется.

* Анализ требований
* Проектирование программного обеспечения
* Программирование
* Тестирование программного обеспечения
* Системная интеграция (System integration)
* Внедрение программного обеспечения (или Установка программного обеспечения)
* Сопровождение программного обеспечения

**Анализ требований** — часть процесса разработки программного обеспечения, включающая в себя сбор требований к программному обеспечению (ПО), их систематизацию, выявление взаимосвязей, а также документирование.

В процессе сбора требований важно принимать во внимание возможные противоречия требований различных заинтересованных лиц, таких как заказчики, разработчики или пользователи.

Анализ требований включает три типа деятельности:

* Сбор требований — общение с клиентами и пользователями, чтобы определить, каковы их требования; анализ предметной области.
* Анализ требований — определение, являются ли собранные требования неясными, неполными, неоднозначными или противоречащими; решение этих проблем; выявление взаимосвязи требований.
* Документирование требований — требования могут быть задокументированы в различных формах, таких как простое описание, сценарии использования, пользовательские истории, или спецификации процессов.

Традиционный способ документировать требования — это создание **списков требований**.

**Преимуществ**

* Обеспечивает контрольный список требований.
* Обеспечивает договор между заказчиками и разработчиками.
* Для большой системы может обеспечить описание высокого уровня.

**Недостатки**

* Такие списки могут занимать сотни страниц. Фактически невозможно прочитать такие документы в целом и получить чёткое понимание системы.
* Такие списки требований перечисляют отдельные требования абстрактно, оторванно друг от друга и от контекста использования
* Эти списки создают ложное чувство взаимопонимания между заинтересованными лицами и разработчиками.
* Эти списки дают заинтересованным лицам ложное чувство защищённости, что разработчики должны достигнуть определённых вещей. Однако, из-за природы этих списков, они неизбежно упускают важные требования, которые будут выявлены позже в процессе. Разработчики могут использовать новые требования для пересмотра сроков и условий в их пользу.

**Проектирование программного обеспечения** — процесс создания проекта программного обеспечения (ПО), а также дисциплина, изучающая методы проектирования. Проектирование ПО является частным случаем проектирования продуктов и процессов.

Целью проектирования является определение внутренних свойств системы и детализации её внешних (видимых) свойств на основе выданных заказчиком требований к ПО (исходные условия задачи). Эти требования подвергаются анализу.

Проектирование ПО включает следующие основные виды деятельности:

* выбор метода и стратегии решения;
* выбор представления внутренних данных;
* разработка основного алгоритма;
* документирование ПО;
* тестирование и подбор тестов;
* выбор представления входных данных.

Первоначально программа рассматривается как чёрный ящик. Ход процесса проектирования и его результаты зависят не только от состава требований, но и выбранной модели процесса, опыта проектировщика.

Модель предметной области накладывает ограничения на бизнес-логику (совокупность правил, принципов, зависимостей поведения объектов предметной области) и структуры данных.

В процессе проектирования ПО для выражения его характеристик используются различные нотации — блок-схемы, ER-диаграммы, UML-диаграммы, DFD-диаграммы, а также макеты.

* Проектированию обычно подлежат:
* Архитектура ПО;
* Устройство компонентов ПО;
* Пользовательские интерфейсы.

В российской практике проектирование ведется поэтапно в соответствии со стадиями, регламентированными ГОСТ 2.103-68.

На каждом из этапов формируется свой комплект документов, называемый проектом (проектной документацией).

**Программирование** - процесс создания компьютерных программ.

В более широком смысле под программированием понимают весь спектр деятельности, связанный с созданием и поддержанием в рабочем состоянии программ — программного обеспечения. Эта инженерно-техническая дисциплина называется «программная инженерия». Сюда входят анализ и постановка задачи, проектирование программы, построение алгоритмов, разработка структур данных, написание текстов программ, отладка и тестирование программы (испытания программы), документирование, настройка (конфигурирование), доработка и сопровождение.

Программирование основывается на использовании языков программирования, на которых записываются инструкции для компьютера. Современное приложение содержит множество таких инструкций, связанных между собой.

**Тестирование программного обеспечения** — процесс исследования, испытания программного продукта, имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, выбранных определенным образом (ISO/IEC TR 19759:2005).

**Внедрение программного обеспечения** — процесс настройки программного обеспечения под определённые условия использования, а также обучения пользователей работе с программным продуктом.

Внедрение программного обеспечения требует действий в трёх следующих плоскостях работ.

* Выделение критических, с точки зрения общего результата, процедур в деятельности организации. Когда набор таких процедур определен, необходимо в первую очередь использовать ИТ-решение для автоматизации операций внутри именно этих процедур. Таким образом, разработанное ИТ-решение автоматически становится жизненно важным и востребованным для организации. Также это обеспечит публичность процесса внедрения.
* Расширение нормативной базы организации путём включения в неё регламентов, описывающих порядок выполнения процедур автоматизируемых процессов. В противном случае есть опасность возникновения рассогласования между автоматизированными процедурами и остальными процессами организации.
* Выполнение работ по общей стандартизации существующей деятельности организации. Лучшие практики выполнения процедур выделяются и включаются в ИТ-решение, в соответствии с принципом наибольшей полезности для большинства участников. Процент таких процедур относительно общего объема автоматизации может быть невелик, но это придает процессу построения решения вес в организации за счет увеличения его «полезности».

**Cопровожде́ние (поддержка) программного обеспечения** — процесс улучшения, оптимизации и устранения дефектов программного обеспечения (ПО) после передачи в эксплуатацию. В ходе сопровождения в программу вносятся изменения, с тем, чтобы исправить обнаруженные в процессе использования дефекты и недоработки, а также для добавления новой функциональности, с целью повысить удобство использования (юзабилити) и применимость ПО.[2]

## Жизненный цикл программного обеспечения

Одним из базовых понятий методологии проектирования ПО является понятие жизненного цикла ее программного обеспечения (ЖЦ ПО). ЖЦ ПО — это непрерывный процесс, который начинается с момента принятия решения о необходимости его создания и заканчивается в момент его полного изъятия из эксплуатации.

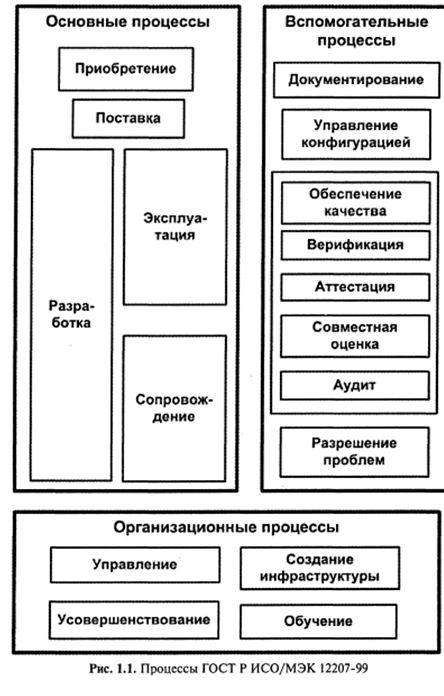
Основным нормативным документом, регламентирующим ЖЦ ПО, является международный стандарт ISO/IEC 12207 (ISO — International Organization of Standardization — Международная организация по стандартизации, IEC — International Electrotechnical Commission — Международная комиссия по электротехнике). Он определяет структуру ЖЦ, содержащую процессы, действия и задачи, которые должны быть выполнены во время создания ПО. В данном стандарте ПО (программный продукт) определяется как набор компьютерных программ, процедур и, возможно, связанной с ним документации и данных. Процесс определяется как совокупность взаимосвязанных действий, преобразующих некоторые входные данные в выходные. Каждый процесс характеризуется определенными задачами и методами их решения, исходными данными, полученными от других процессов, и результатами.

Структура ЖЦ ПО по стандарту ISO/IEC 12207 базируется на трех группах процессов:

· основные процессы ЖЦ ПО (приобретение, поставка, разработка, эксплуатация, сопровождение);

· вспомогательные процессы, обеспечивающие выполнение основных процессов (документирование, управление конфигурацией, обеспечение качества, верификация, аттестация, оценка, аудит, решение проблем);

· организационные процессы (управление проектами, создание инфраструктуры проекта, определение, оценка и улучшение самого ЖЦ, обучение). [3]



## Модели жизненного цикла ПО

**Модель процесса создания ПО** – последовательность этапов, необходимых для разработки создаваемого ПО.

**Модели процесса разработки ПО:**

1. [Каскадная (Водопадная) модель](#Водопадная)
2. [Итерационная модель](#Итерационная)
3. [Спиральная модель](#Спиральная)
4. [V model](#Vmodel)

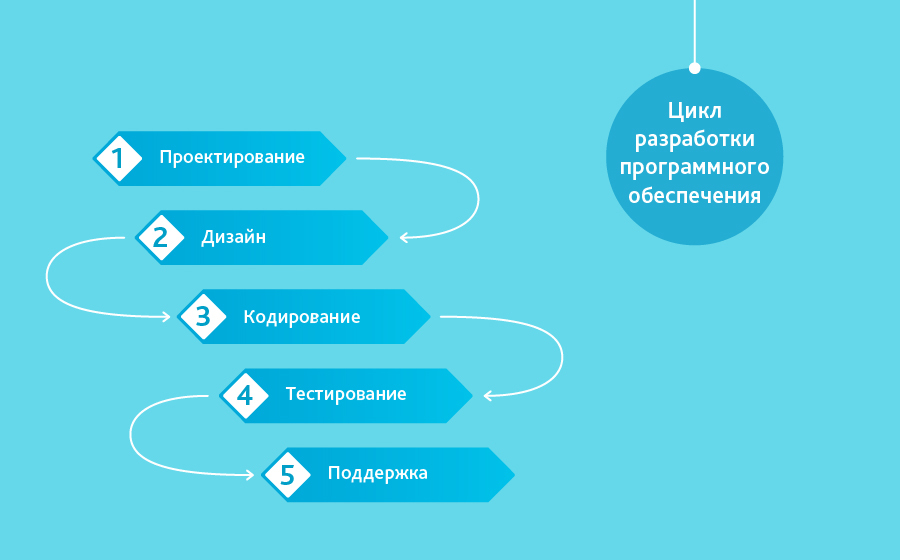
1 Водопадная модель жизненного цикла (англ. waterfall model) была предложена в 1970 г. Уинстоном Ройсом. Она предусматривает последовательное выполнение всех этапов проекта в строго фиксированном порядке. Переход на следующий этап означает полное завершение работ на предыдущем этапе. Требования, определенные 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:

* Формирование требований;
* Проектирование;
* Реализация;
* Тестирование;
* Внедрение;
* 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. Таким образом, водопадная модель для крупных проектов мало реалистична и может быть эффективно использована только для создания небольших систем.

2 Итерационная модель (70-80-е годы XX в.) модель итеративной и инкрементальной разработки (англ. iterative and incremental development, IID). Модель IID предполагает разбиение жизненного цикла проекта на последовательность итераций, каждая из которых напоминает «мини-проект», включая все процессы разработки в применении к созданию меньших фрагментов функциональности, по сравнению с проектом в целом. Цель каждой *итерации* — получение работающей версии программной системы, включающей функциональность, определённую интегрированным содержанием всех предыдущих и текущей итерации. Результат финальной итерации содержит всю требуемую функциональность продукта. Таким образом, с завершением каждой итерации продукт получает приращение — *инкремент* — к его возможностям, которые, следовательно, развиваются *эволюционно*. Итеративность, инкрементальность и эволюционность в данном случае есть выражение одного и то же смысла разными словами со слегка разных точек зрения.

По выражению Т. Гилба, «эволюция — прием, предназначенный для создания видимости стабильности. Шансы успешного создания сложной системы будут максимальными, если она реализуется в серии небольших шагов и если каждый шаг заключает в себе четко определённый успех, а также возможность «отката» к предыдущему успешному этапу в случае неудачи. Перед тем, как пустить в дело все ресурсы, предназначенные для создания системы, разработчик имеет возможность получать из реального мира сигналы обратной связи и исправлять возможные ошибки в проекте»[4].

Подход IID имеет и свои отрицательные стороны, которые, по сути, — обратная сторона достоинств. Во-первых, целостное понимание возможностей и ограничений проекта очень долгое время отсутствует. Во-вторых, при итерациях приходится отбрасывать часть сделанной ранее работы. В-третьих, добросовестность специалистов при выполнении работ всё же снижается, что психологически объяснимо, ведь над ними постоянно довлеет ощущение, что «всё равно всё можно будет переделать и улучшить позже»[3].



Модель предусматривает обобщение полученных проектных решений отдельных задач в общесистемные решения. При этом возникает потребность в пересмотре ранее сформулированных требований.

**Достоинство:** возможность оперативно вносить коррективы в проект.

**Недостаток:** при большом числе итераций растет время проектирования, возникают расхождения в проектных решениях и документации, запутывается функциональная и системная архитектура созданной ПО. Необходимость в перепроектировании старой или создании новой системы может возникнуть сразу после этапа внедрения или эксплуатации.

3 Спиральная модель (80-90-е годы XX в.) соответствует технологии проектирования «сверху — вниз». Предполагает использование программного прототипа, допускающего программное расширение. Проект системы циклически повторяет путь от детализации требований к детализации программного кода.



При проектировании архитектуры системы сначала определяется состав функциональных подсистем и решаются общесистемные вопросы (организация интегрированной базы данных, технология сбора, передачи и накопления информации). Затем формулируются отдельные задачи и разрабатывается технология их решения.

При программировании сначала разрабатываются головные программные модули, а затем — модули, исполняющие отдельные функции. Сначала обеспечивается взаимодействие модулей между собой и с базой данных, а затем — реализация алгоритмов.

**Достоинства:**

1. сокращение число итераций и, следовательно, число ошибок и несоответствий, которые необходимо исправлять;

2. сокращение сроков проектирования;

3. упрощение создания проектной документации.

**Недостаток:** высокие требования к качеству общесистемного репозитория (общей базы проектных данных).

Спиральная модель лежит в основе **технологии быстрой разработки приложений** или RAD-технологии (rapid application development), которая предполагает активное участие конечных пользователей будущей системы в процессе ее создания. Основные стадии информационного инжиниринга следующие:

· **Анализ и планирование информационной стратегии.** Пользователи вместе со специалистами-разработчиками участвуют в идентификации проблемной области.

· **Проектирование.**Пользователи под руководством разработчиков принимают участие в техническом проектировании.

· **Конструирование.** Разработчики проектируют рабочую версию ПО с использованием языков 4-го поколения;

· **Внедрение.** Разработчики обучают пользователей работе в среде новой ПО. [3][4]

4 V-Model (или VEE модель) является моделью разработки информационных систем (ИС), направленной на упрощение понимания сложностей, связанных с разработкой систем. Она используется для определения единой процедуры разработки программных продуктов, аппаратного обеспечения и человеко-машинных интерфейсов.

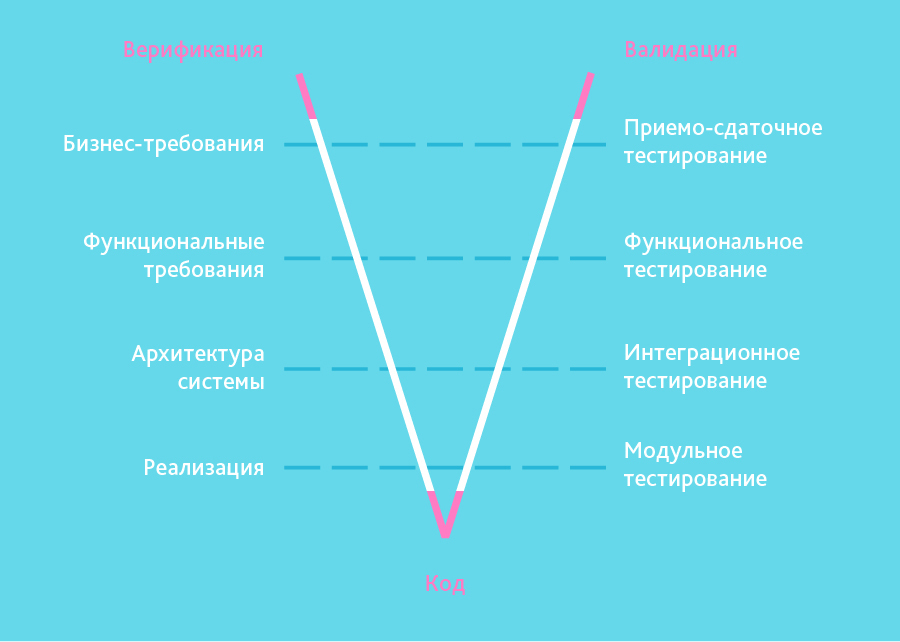
**История**

Концепция V-образной модели была разработана Германией и США в конце 1980-х годов независимо друг от друга:

* Немецкая V-модель была разработана аэрокосмической компанией IABG в [Оттобрунне](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D1%82%D0%BE%D0%B1%D1%80%D1%83%D0%BD%D0%BD" \o "Оттобрунн) рядом с [Мюнхеном](https://ru.wikipedia.org/wiki/%D0%9C%D1%8E%D0%BD%D1%85%D0%B5%D0%BD) в содействии с Федеральным департаментом по закупке вооружений в [Кобленце](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B1%D0%BB%D0%B5%D0%BD%D1%86), для Министерства обороны Германии. Модель была принята немецкой федеральной администрацией для гражданских нужд летом 1992[[1]](https://ru.wikipedia.org/wiki/V-Model#cite_note-V-Model_lyfecycle_process_model-1).
* Американская V-Model (VEE) была разработана [национальным советом по системной инженерии](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B6%D0%B4%D1%83%D0%BD%D0%B0%D1%80%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D1%81%D0%BE%D0%B2%D0%B5%D1%82_%D0%BF%D0%BE_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%BD%D0%BE%D0%B9_%D0%B8%D0%BD%D0%B6%D0%B5%D0%BD%D0%B5%D1%80%D0%B8%D0%B8) (международным — с 1995 года) для спутниковых систем, включая оборудование, программное обеспечение и взаимодействие с пользователями[[2]](https://ru.wikipedia.org/wiki/V-Model#cite_note-NCOSE_First_Annyal_Symposium-2).

Современной версией V-Model является V-Model XT, которая была утверждена в феврале [2005 года](https://ru.wikipedia.org/wiki/2005_%D0%B3%D0%BE%D0%B4). V-модель используется для управления процессом разработки [программного обеспечения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) для немецкой федеральной администрации. Сейчас она является стандартом для немецких правительственных и оборонных проектов, а также для производителей ПО в Германии. V-Model представляет собой скорее набор стандартов в области проектов, касающихся разработки новых продуктов. Эта модель во многом схожа с [PRINCE2](https://ru.wikipedia.org/wiki/PRINCE2) и описывает методы как для проектного управления, так и для системного развития.

**Основные принципы**



V-Model процесса разработки ИС.

Основной принцип V-образной модели заключается в том, что детализация проекта возрастает при движении слева направо, одновременно с течением времени, и ни то, ни другое не может повернуть вспять. Итерации в проекте производятся по горизонтали, между левой и правой сторонами буквы.

Применительно к разработке [информационных систем](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) V-Model — вариация [каскадной модели](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B2%D0%BE%D0%B4%D0%BE%D0%BF%D0%B0%D0%B4%D0%B0), в которой задачи разработки идут сверху вниз по левой стороне буквы V, а задачи тестирования — вверх по правой стороне буквы V. Внутри V проводятся горизонтальные линии, показывающие, как результаты каждой из фаз разработки влияют на развитие системы тестирования на каждой из фаз тестирования. Модель базируется на том, что приёмо-сдаточные испытания основываются, прежде всего, на требованиях, системное тестирование — на требованиях и архитектуре, комплексное тестирование — на требованиях, архитектуре и интерфейсах, а компонентное тестирование — на требованиях, архитектуре, интерфейсах и алгоритмах[[4]](https://ru.wikipedia.org/wiki/V-Model#cite_note-Economicus-4).

**Цели**

V-модель обеспечивает поддержку в планировании и реализации проекта. В ходе проекта ставятся следующие задачи:

* **Минимизация рисков:** V-образная модель делает проект более прозрачным и повышает качество контроля проекта путём стандартизации промежуточных целей и описания соответствующих им результатов и ответственных лиц. Это позволяет выявлять отклонения в проекте и риски на ранних стадиях и улучшает качество управления проектов, уменьшая риски.
* **Повышение и гарантии качества:** V-Model — стандартизованная модель разработки, что позволяет добиться от проекта результатов желаемого качества. Промежуточные результаты могут быть проверены на ранних стадиях. Универсальное документирование облегчает читаемость, понятность и проверяемость.
* **Уменьшение общей стоимости проекта:** Ресурсы на разработку, производство, управление и поддержку могут быть заранее просчитаны и проконтролированы. Получаемые результаты также универсальны и легко прогнозируются. Это уменьшает затраты на последующие стадии и проекты.
* **Повышение качества коммуникации между участниками проекта:** Универсальное описание всех элементов и условий облегчает взаимопонимание всех участников проекта. Таким образом, уменьшаются неточности в понимании между пользователем, покупателем, поставщиком и разработчиком[[5]](https://ru.wikipedia.org/wiki/V-Model#cite_note-Objectives_of_the_V-Model-5).

**Достоинства**

* Пользователи V-Model участвуют в разработке и поддержке V-модели. Комитет по контролю за изменениями поддерживает проект и собирается раз в год для обработки всех полученных запросов на внесение изменений в V-Model[[6]](https://ru.wikipedia.org/wiki/V-Model" \l "cite_note-Further_development_of_the_V-Model-6).
* На старте любого проекта V-образная модель может быть адаптирована под этот проект, так как эта модель не зависит от типов организаций и проектов[[7]](https://ru.wikipedia.org/wiki/V-Model#cite_note-V-Model_tailoring-7).
* V-model позволяет разбить деятельность на отдельные шаги, каждый из которых будет включать в себя необходимые для него действия, инструкции к ним, рекомендации и подробное объяснение деятельности[[8]](https://ru.wikipedia.org/wiki/V-Model#cite_note-activity-8).

**Ограничения**

Следующие моменты не учитываются в V-модели, но могут быть рассмотрены отдельно, либо возможно адаптировать модель под них:

* Не регулируется размещение контрактов на обслуживание.
* Организация и выполнение управления, обслуживания, ремонта и утилизации системы не учитываются в V-модели. Однако, планирование и подготовка к этим операциям моделью рассматриваются.
* V-образная модель больше касается разработки программного обеспечения в проекте, чем всей организации процесса[[9]](https://ru.wikipedia.org/wiki/V-Model#cite_note-limits-9).

## Методологии

[1 Гибкая методология разработки Agile](#Agile)

* + 1. [XP](#XP)
    2. Scrum
    3. FDD

2 Cleanroom

3 OpenUP

4 [RAD](#Rad)

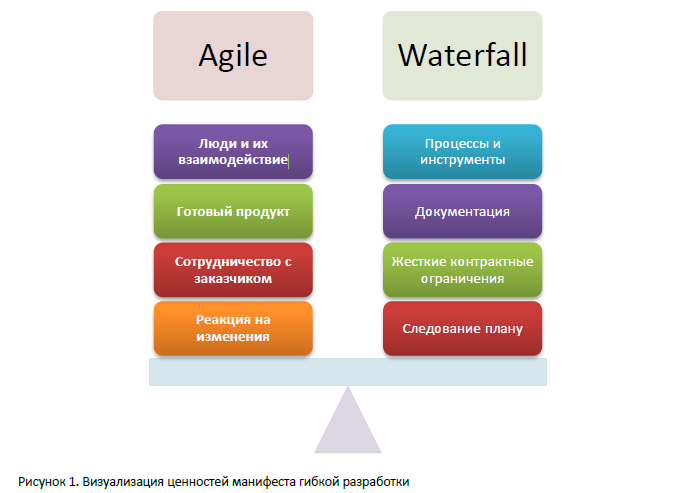
5 RUP

6 MSF

7 DSDM

8 TDD

Гибкая методология разработки (от англ. - Agile software development) - манифест, определяющий способ мышления и содержащий основные ценности и принципы, на которых базируется несколько подходов (фреймворков, от англ. framework — каркас, структура) например такие, как: XP, Lean, Scrum, FDD и др.)[5]



[6]

Экстремальное программирование (XP) – это упрощенная методология организации разработки программ для небольших и средних по размеру команд разработчиков, занимающихся созданием программного продукта в условиях неясных или быстро меняющихся требований.

Цели XP

Основными целями XP являются повышение доверия заказчика к программному продукту путем предоставления реальных доказательств успешности развития процесса разработки и резкое сокращение сроков разработки продукта. При этом XP сосредоточено на минимизации ошибок на ранних стадиях разработки. Это позволяет добиться максимальной скорости выпуска готового продукта и даёт возможность говорить о прогнозируемости работы. Практически все приемы XP направлены на повышение качества программного продукта.

Принципы XP

Основными принципами являются:

* Итеративность. Разработка ведется короткими итерациями при наличии активной взаимосвязи с заказчиком. Итерации как таковые предлагается делать короткими, рекомендуемая длительность – 2-3 недели и не более 1 месяца. За одну итерацию группа программистов обязана реализовать несколько свойств системы, каждое из которых описывается в пользовательской истории. Пользовательские истории (ПИ) в данном случае являются начальной информацией, на основании которой создается модуль. Они отличаются от вариантов использования (ВИ). Описание ПИ короткое – 1-2 абзаца, тогда как ВИ обычно описываются достаточно подробно, с основным и альтернативными потоками, и дополняются моделью. ПИ пишутся самими пользователями, которые в XP являются частью команды, в отличие от ВИ, которые описывает системный аналитик. Отсутствие формализации описания входных данных проекта в XP стремятся компенсировать за счет активного включения в процесс разработки заказчика как полноправного члена команды.
* Простота решений. Принимается первое простейшее рабочее решение. Экстремальность метода связана с высокой степенью риска решения, обусловленного поверхностностью анализа и жестким временным графиком. Реализуется минимальный набор главных функций системы на первой и каждой последующей итерации; функциональность расширяется на каждой итерации.
* Интенсивная разработка малыми группами (не больше 10 человек) и парное программирование (когда два программиста вместе создают код на одном общем рабочем месте), активное общение в группе и между группами. Все это нацелено на как можно более раннее обнаружение проблем (как ошибок, так и срыва сроков). Парное программирование направлено на решение задачи стабилизации проекта. При применении XP методологии высок риск потери кода по причине ухода программиста, не выдержавшего интенсивного графика работы. В этом случае второй программист из пары играет роль «наследника» кода. Немаловажно и то, как именно распределены группы в рабочем пространстве – в XP используется открытое рабочее пространство, которое предполагает быстрый и свободный доступ всех ко всем.
* Обратная связь с заказчиком, представитель которого фактически вовлечен в процесс разработки.
* Достаточная степень смелости и желание идти на риск.

Приемы XP (практики)

Обычно XP характеризуют набором из 12 правил (практик), которые необходимо выполнять для достижения хорошего результата. Ни одна из практик не является принципиально новой, но в XP они собраны вместе.

* + 1. Планирование процесса. Вся команда разработчиков собирается вместе, принимается коллективное решение о том, какие свойства системы будут реализованы в ближайшей итерации. Трудоемкость реализации каждого свойства определяется самими программистами.
    2. Тесное взаимодействие с заказчиком. Представитель заказчика должен быть членом XP-команды. Он пишет ПИ, выбирает истории, которые будут реализованы в конкретной итерации, и отвечает на вопросы, касающиеся бизнеса. Представитель заказчика должен быть экспертом в автоматизируемой предметной области. Необходимо наличие постоянное обратной связи с представителем заказчика.
    3. Общесистемные правила именования. Хорошие системные правила именования предполагают простоту именования классов и переменных. Команда разработчиков должна иметь единые правила именования.
    4. Простая архитектура. Любое свойство системы должно быть реализовано как можно проще. Программисты в XP-команде работают под девизом: «Ничего лишнего!». Принимается первое простейшее работающее решение, реализуется необходимый уровень функциональности на данный момент. Тем самым экономится время программиста.
    5. Рефакторинг. Это оптимизация существующего кода с целью его упрощения, Такая работа должна вестись постоянно. Сохраняя код прозрачным и определяя его элементы всего один раз, программисты сокращают число ошибок, которые впоследствии придется устранять. При реализации каждого нового свойства системы программист должен подумать над тем, можно ли упростить существующий код и как это поможет реализовать новое свойство. Кроме того, нельзя совмещать рефакторинг с дизайном: если создается новый код, рефакторинг следует отложить.
    6. Парное программирование. Все программисты должны работать в парах: один пишет код, другой смотрит. Таким образом, необходимо размещать группу программистов в одном месте. XP наиболее успешно работает в нераспределенных коллективах программистов и пользователей.
    7. 40-часовая рабочая неделя. Программист не должен работать более 8 часов в день. Необходимость сверхурочной работы – это четкий индикатор проблемы на данном конкретном направлении разработки. Поиск причин сверхурочной работы и их скорейшее устранение – одно из основных правил.
    8. Коллективное владение кодом. Каждый программист в коллективе должен иметь доступ к коду любой части системы и право вносить изменения в любой код. Обязательное правило: если программист внес изменения и система после этого работает некорректно, то именно этот программист должен исправить ошибки.
    9. Единые стандарты кодирования. Стандарты кодирования нужны для обеспечения других практик: коллективного владения кодом, парного программирования и рефакторинга. Без единого стандарта выполнять эти практики как минимум сложнее, а в реальности вообще невозможно: группа будет работать в режиме постоянной нехватки времени. Команда работает над проектом продолжительное время. Люди приходят и уходят. Никто не кодирует в одиночку и код принадлежит всем. Всегда будут моменты, когда необходимо будет понять и скорректировать чужой код. Разработчики будут удалять дублирующий код, анализировать и улучшать чужие классы и т. п. Со временем нельзя будет сказать, кто автор конкретного класса. Следовательно, все должны подчиняться общим стандартам кодирования – форматирование кода, именование классов, переменных, констант, стиль комментариев. Вышесказанное означает, что все члены команды должны договориться об общих стандартах кодирования. Неважно каких, но все обязаны им подчиняются.
    10. Небольшие релизы. Минимальная итерация – один день, максимальная – месяц; чем чаще осуществляются релизы, тем больше недостатков системы будет выявлено. Первые релизы помогают выявить недостатки на самых ранних стадиях, далее функциональность системы расширяется на основании ПИ. Поскольку пользователь включается в процесс разработки начиная с первого релиза, то он оценивает систему и выдает пользовательскую историю и замечания. На основании этого определяется следующая итерация, то есть, каким будет новый релиз. В XP все направлено на обеспечение непрерывной обратной связи с пользователями.
    11. Непрерывная интеграция. Интеграция новых частей системы должна происходить как можно чаще, как минимум раз в несколько часов. Основное правило интеграции следующее: интеграцию можно производить, если все тесты проходят успешно. Если тесты не проходят, то программист должен либо внести исправления и тогда интегрировать составные части системы, либо вообще не интегрировать их. Правило это – жесткое и однозначное. Если в созданной части системы имеется хотя бы одна ошибка, то интеграцию производить нельзя. Частая интеграция позволяет быстрее получить готовую систему, вместо того чтобы тратить на сборку неделю.
    12. Тестирование. В отличие от большинства остальных методологий тестирование в XP – одно из важнейших составляющих. Экстремальный подход предполагает, что тесты пишутся до написания кода. Каждый модуль обязан иметь unit test – тест данного модуля. Таким образом, в XP осуществляется регрессионное тестирование, «неухудшение качества» при добавлении функциональности. Большинство ошибок исправляются на стадии кодирования. Тесты пишут сами программисты, любой из них имеет право написать тест для любого модуля. Еще один важный принцип: тест определяет код, а не наоборот (test-driven development), то есть кусок кода кладется в хранилище тогда и только тогда, когда все тесты прошли успешно, в противном случае данное изменение кода отвергается.

Процесс XP является неформальным, но требует высокого уровня самодисциплины. Если это правило не выполняется, то XP мгновенно превращается в хаотичный и неконтролируемый процесс. XP не требует от программистов написания множества отчетов и построения массы моделей. В XP каждый программист считается квалифицированным работником, который профессионально и с большой ответственностью относится к своим обязанностям. Если в команде этого нет, то внедрять XP абсолютно бессмысленно – лучше для начала заняться перестройкой команды. Риск разработки снижается только в команде, которой XP подходит идеально, во всех остальных случаях XP – это процесс разработки с наиболее высокой степенью риска, поскольку другие методы снижения коммерческих рисков, кроме человеческого фактора, в XP просто отсутствуют.[7]

RAD Model (rapid application development model или быстрая разработка приложений)

RAD-модель — разновидность инкрементной модели. В RAD-модели компоненты или функции разрабатываются несколькими высококвалифицированными командами параллельно, будто несколько мини-проектов. Временные рамки одного цикла жестко ограничены. Созданные модули затем интегрируются в один рабочий прототип. Синергия позволяет очень быстро предоставить клиенту для обозрения что-то рабочее с целью получения обратной связи и внесения изменений.



Модель быстрой разработки приложений включает следующие фазы:

Бизнес-моделирование: определение списка информационных потоков между различными подразделениями.

Моделирование данных: информация, собранная на предыдущем этапе, используется для определения объектов и иных сущностей, необходимых для циркуляции информации.

Моделирование процесса: информационные потоки связывают объекты для достижения целей разработки.

Сборка приложения: используются средства автоматической сборки для преобразования моделей системы автоматического проектирования в код.

Тестирование: тестируются новые компоненты и интерфейсы.

Когда используется RAD-модель?

Может использоваться только при наличии высококвалифицированных и узкоспециализированных архитекторов. Бюджет проекта большой, чтобы оплатить этих специалистов вместе со стоимостью готовых инструментов автоматизированной сборки. RAD-модель может быть выбрана при уверенном знании целевого бизнеса и необходимости срочного производства системы в течение 2-3 месяцев.

1. Технология разработки программного обеспечения Конспект лекций Составитель И.И. Савенко.
2. Https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81\_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8\_%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.
3. Https://studopedia.ru/11\_201235\_modeli-zhiznennogo-tsikla-po.html.
4. Https://studopedia.ru/19\_240055\_protsessi-zhiznennogo-tsikla-po.html.
5. Http://fb.ru/article/342838/chto-takoe-agile-perevod-sferyi-primeneniya-gibkaya-metodologiya-razrabotki.
6. Борис\_Вольфсон\_Гибкие\_методологии.
7. http://www.informicus.ru/Default.aspx?id=95&SECTION=6.