Разработка программного обеспечения – это комплексный процесс, на ход которого влияют различные факторы. Для систематизации и описания каждого элемента потребовалась бы целая книга, однако важно выделить наиболее значимые части этого процесса.

Обычно, под разработкой подразумевают модель, однако это не единственное, что нужно знать. В нашей статье мы расскажем, что такое разработка ПО, через какие этапы она проходит, и разберем наиболее актуальные модели этого процесса.

**Понятие разработки программного обеспечения**

В первую очередь, необходимо дать определение понятию разработки программного обеспечения.

Программное обеспечение (ПО) — это исполняемый код, который осуществляет те или иные вычислительные операции. ПО является совокупностью элементов, в которую входит исполняемый программный код, связанные библиотеки и документация. Если оно создается в целях выполнения конкретных задач, то речь уже идёт о программном продукте (ПП).

Понятие разработки программного обеспечения

Ещё одним важным понятием, которое необходимо рассмотреть в рамках этой темы, является инжиниринг. Данная область представляет собой разработку продуктов с применением конкретной научной методологии.

Программная инженерия — это отдельная область деятельности, внутри которой разрабатываются программные продукты. При этом используются максимально конкретизированные научные методы и принципы. Конечной целью является создание высококачественного и полезного программного продукта.

По определению IEEE разработка программного обеспечения представляет собой применение систематического, дисциплинированного, количественного подхода к разработке, а также дальнейшее использование и обслуживание полученного результата.

**Методы структурного анализа для проектирования ПО**

Структурные методы составляют дисциплину системного анализа и проектирования. Благодаря таким методам появляется возможность устранить различные затруднения, связанные со спецификой больших систем. Достигается это за счёт их дифференцирования на составные части, которые еще называют «черными ящиками», а также иерархической организации таких «черных ящиков».

Практическая польза дифференциации состоит в том, что при использовании полученных частей необязательно понимать принцип их работы. Пользователю достаточно лишь знать их входы и выходы, а также назначение. Проще говоря, необходимо понимать, какие именно задачи должен выполнять тот или иной «черный ящик».

Методы структурного анализа для проектирования ПО

Исходя из всего этого, следует, что на первом этапе процесса упрощения сложной системы ее разделяют на несколько «черных ящиков». Однако деление должно соответствовать нескольким основным критериям:

* У каждого «черного ящика» должна быть одна единственная функция.
* Функции этих «ящиков» должны быть просты для понимания, даже если их сложно реализовать на практике.
* Взаимосвязь между элементами системы должна создаваться только в том случае, если взаимосвязаны их функции. Скажем, в бухгалтерии один из таких «черных ящиков» нужен для определения размера общей заработной платы работника, а другой — для определения размера налогов. Очевидно, что между ними должна быть связь. Ведь чтобы высчитать размер налогов, необходимо знать размер зарплаты.
* Любые взаимосвязи между «черными ящиками» должны быть максимально простыми. Благодаря этому они становятся независимыми друг от друга.

Ещё один фундаментальный аспект в области структурных методов — идея иерархии. Чтобы разобраться в сложной системе, нужно не только дифференцировать ее, но ещё и обеспечить грамотную организацию полученных частей. Как раз с этой целью и применяются иерархические структуры.

Если задуматься, то любая сложная система в нашем мире, будь то элементарная частица или целая галактика, обязательно устроена в определенной иерархии. Если сложную систему разрабатывает сам человек, то он использует этот природный принцип в своей сфере деятельности.

Например, каждая компания имеет директора, заместителей по направлениям, иерархию руководителей подразделений, рядовых служащих. Помимо этого, структурные методы часто применяют визуальное моделирование, которое необходимо для простоты понимания сложных структур.

Структурный анализ — это способ изучения системы. В первую очередь, производится ее общий обзор, а затем выполняется детализация полученной информации. В конечном итоге исследователи получают иерархическую структуру с большим числом уровней.

Функциональная декомпозиция — важнейший метод дифференциации на уровни абстракций в рамках структурного анализа. Декомпозиция представляет собой разделение целого на части. В данном случае речь идёт о разбиении системы на функциональные подсистемы, которые затем делятся на подфункции. Последние, в свою очередь, разделяются на задачи, а те — на конкретные процедуры.

Вместе с тем система все еще будет являться целостной, а все ее составляющие — связаны между собой. Если система разрабатывается «снизу-вверх» (от конкретных задач к общей системе), то утрачивается ее целостное представление. Кроме того, появляются трудности связанные с описанием информационного взаимодействия отдельных элементов.

Методы структурного анализа для проектирования ПО

В процессе структурного анализа и проектирования применяется множество моделей, которые описывают:

* функциональную структуру системы;
* последовательность производимых операций;
* передачу данных между функциональными процессами;
* отношения между данными.

**Для вас подарок! В свободном доступе**

Выделим несколько самых часто встречаемых моделей из первых трёх категорий:

* функциональная модель SADT (Structured Analysis and Design Technique);
* модель IDEF3;
* DFD (Data Flow Diagrams) — диаграммы потоков данных. Модель «сущность — связь» (ERM — Entity-Relationship Model), которая описывает отношения между данными. Обычно применяется в структурном анализе и проектировании. При этом она является подмножеством объектной модели предметной области.

**Этапы разработки программного обеспечения**

**Первая стадия работы над ПО — подготовка**

Основная задача, которую необходимо выполнить на данном этапе, заключается в формировании концепции будущей системы на основе требований заказчика. Ориентируясь на эту концепцию, разработчики дают оценку тому, насколько проект востребован и реализуем. Если решение о привлечении исполнителя принимается на основании проведенного конкурса, то речь идет об этапе подготовки потенциального сотрудника к этому конкурсу (включая формирование всех документов).

Вполне очевидно, что нет никакого смысла затрачивать время и деньги на проект, который является потенциально невостребованным и нереализуемым. В этом случае завершение проекта является наиболее рациональным решением.

Бывают ситуации, при которых необходима определенная итеративная работа с заказчиком по корректировке концепции проекта вплоть до момента, когда будет достигнуто достаточное соотношение требований заказчика и затрат исполнителя, либо когда будет принято решение о завершении разработки.

Если в основе проекта лежит реализуемая концепция, то наступает этап разработки требований. Данная стадия предполагает определение явных и неявных потребностей заказчика. Зачастую заказчики не имеют четкого представления о своих нуждах. В некоторых ситуациях их нужды не соотносятся с реальными возможностями разработчиков. Иногда потребности заказчиков имеют внутренние противоречия.

Именно для устранения таких проблем и нужен этап разработки требований. Необходимо максимально конкретизировать потребности заказчика и выявить его скрытые нужды. Кроме того, на данной стадии устраняются противоречия между требованиями, создаётся целостное техническое решение и производится анализ его реализуемости.

Конкретизация требований нередко влечёт за собой корректировку концепции проекта. Однако в некоторых ситуациях не получается найти эффективное техническое решение, и тогда проект либо закрывают, либо замораживают до появления выгодных условий.

Если же решение удалась найти, то исполнитель переходит на этап разработки архитектуры будущей системы. Главная задача данной стадии — определение верхнеуровневой логической и физической архитектуры, которая способна всецело закрыть потребности заказчика. В процессе разработки архитектуры выполняется рецензирование и уточнение концепции, требований и предварительного технического решения. Это позволяет снизить самые выраженные риски.

По окончании проектирования архитектуры следует еще раз проверить проект с целью выяснить, сможет ли исполнитель реализовать концепцию. На этапе разработки архитектуры рекомендуется убрать лишние и громоздкие функции. Такая оптимизация нередко помогает вписаться в оптимальные параметры проекта.

Однако иногда необходимо гораздо более серьезное урезание функциональной составляющей будущей системы. Но даже если сложится ситуация, при которой работы над проектом будут приостановлены, это все равно лучше, чем продолжение разработки.

Этапы разработки программного обеспечения

Если же результат оказался положительным, и была сформирована благоприятная архитектура системы, наступает этап реализации и поставки. При этом реализация может выполняться как в один, так и в несколько этапов. Если речь идёт о небольшом проекте, то можно ограничиться лишь одним шагом. Но когда проект является крупномасштабным, подсистемы внутри разрабатываемой системы становятся более зависимыми.

В таких случаях реализация подразделяется на определенное количество стадий. Причем делается это таким образом, чтобы по завершении каждой стадии разработчики получали готовый к поставке результат. Самые важные функции следует разрабатывать на начальных этапах, а менее важные — на последующих стадиях. Благодаря такому подходу самые опасные для системы ошибки будут устранены еще в самом начале, что повысит стабильность основы системы.

**Следующий этап — опытная эксплуатация**

Главная задача данной стадии — проверка качества работы системы в реальных условиях. Проверка чаще всего состоит из измерения количественных метрик, с помощью которых определяется качество продукта. Сначала испытываются функциональные показатели качества, а после этого — нефункциональные. Если в ходе проверки выявляются какие-либо расхождения, то исполнитель вносит коррективы в системный код.

Когда систему удается правильно настроить, ее вводят в эксплуатацию. Обычно исполнитель некоторое время сопровождает разработанный им продукт (как минимум во время гарантийного срока). При обнаружении тех или иных ошибок система корректируется. Пользователям и обслуживающему персоналу заказчика должна своевременно оказываться поддержка в виде консультаций.

Рано или поздно система потеряет свою актуальность для заказчика. С этого момента можно говорит об этапе ее вывода из эксплуатации. Однако для программного обеспечения, которое разрабатывается под заказ, этот этап может и не наступить. Дело в том, что заказчик, опираясь на свои эксклюзивные права, может не допустить исполнителя к дальнейшему сопровождению и настройке системы ещё до потери ее актуальности.

**Конечный этап любого проекта — завершение**

На этой стадии производится анализ результатов и внесение корректировок в процесс разработки программного обеспечения с опорой на полученный опыт. Кроме того, осуществляется пополнение базы знаний разработчиков новыми решениями, которые доказали свою эффективность, а также различными предостережениями и новыми компонентами. В дальнейшем все это должно применяться при разработке других проектов.

**Вспомогательные процессы при разработке ПО**

В рамках разработки программного обеспечения можно выделить несколько вспомогательных процессов.

* **Документирование**. Исполнитель формирует документацию и руководства пользователя к создаваемому программному продукту, как во время разработки, так и после. Такие документы позволяют программистам разбираться в структуре и коде даже по прошествии длительного времени после их создания. При этом документация помогает пользователям взаимодействовать с системой.
* **Управление конфигурацией**. Речь идет о работах по управлению наборами создаваемых компонентов программного обеспечения, а также по управлению версиями программного продукта.
* **Обеспечение качества**. Данный процесс необходим для обеспечения соответствия ПП предварительным требованиям к разработке и нормам организации исполнителя и заказчика.
* **Верификация**. Позволяет обнаружить ошибки, которые были допущены при разработке ПО. Кроме того, благодаря верификации можно выявить несоответствия между разрабатываемым ПО и сформированной архитектурой.
* **Аттестация**. Она необходима для подтверждения соответствия получаемых величин принятым стандартам. Иными словами, выходные данные должны иметь погрешность, которая удовлетворяет всем требованиям и нормам.
* **Совместная оценка**. Данный процесс направлен на контроль и проверку состояния персонала и создаваемого ПП. Осуществляется заказчиком и исполнителем в течение всего проекта.
* **Аудит**. Необходим для независимой оценки текущей ситуации, характеристики проекта, документации и различных отчетов. Данный процесс позволяет сравнить реальное положение дел с договором и документами, в которых прописаны требования. Аудит может проводиться как одной, так и двумя сторонами.
* **Разрешение проблем**. Устраняются ошибки, которые были обнаружены на этапах контроля и оценки.

**Факторы, влияющие на разработку ПО**

Перечислим факторы, которые непосредственно влияют на результат разработки ПО:

* **Классы решаемых задач**. Определяют смысловое содержание создаваемых программ.
* **Применяемые методологии**. Они задают особенности организационного и технического выполнения базовых этапов создания ПО.
* **Методы и парадигмы программирования**. От них зависят стили кодирования и архитектуры виртуальных машин;
* **Аппаратные и системные программные средства**. Они являются виртуальными и физическими ресурсами, благодаря которым становится возможным применение ПО.

Соотношение данных факторов формирует разнообразие вариантов организации разработки. Выделим базовые составляющие этого процесса.

Основная цель разработки программного обеспечения — создание программы, которая сможет выполнять определенную задачу и удовлетворять имеющимся стандартам. Решаемую задачу описывают набором формальных и неформальных (эмпирических) моделей. Они определяют осуществляемые в программе процессы и применяемые при этом данные.

Модель задачи представляет собой комплекс специализированных моделей, которые описывают те или иные нюансы решаемой задачи, отражаемые в создаваемой программе.

Специализированная модель необходима для описания конкретных параметров исследуемого явления. Она позволяет сосредоточиться на частных характеристиках.

Создаваемая программа должна выполнять функции, которые нужны для решения задачи в определенном исполнителе (вычислительной системе). Для отражения его особенностей используется модель исполнителя.

Факторы, влияющие на разработку ПО

Модель исполнителя представляет собой набор специализированных моделей, которые описывают организацию и поведение вычислительной системы, производящей выполнение программы.

Разрабатываемая программа выступает в качестве отображения модели решаемой задачи на модель исполнителя. Уровень сложности программирования зависит от числа таких специализированных моделей, описывающих задачу, а также их размера и семантического отличия от специализированных моделей исполнителя.

Кроме того, трудоемкость процесса разработки определяется параметрами модели исполнителя, которая описывает требования к уровню абстракции создаваемой программы и ее схожестью с архитектурой реального вычислителя.

**Модели разработки программного обеспечения**

Существует несколько видов разработки программного обеспечения, которые основываются на разных моделях. Перечислим 5 самых распространенных из них.

**Waterfall (каскадная модель, или «водопад»)**

В данном случае разработка выполняется в несколько этапов. Причем каждый следующий этап может начаться лишь после завершения предыдущего. При грамотном использовании каскадная модель является самой скоростной и простой. Ее начали применять еще в 1970-х.

Преимущества:

* Простота контроля. Заказчик будет всегда понимать, что в данный момент делают исполнители, сможет регулировать сроки и бюджет.
* Возможность расчёта стоимости проекта ещё на начальной стадии. Каждый нюанс прописывается во время стадии согласования договора.
* Отсутствие необходимости привлечения очень опытных тестировщиков. Специалисты смогут базироваться на подробной технической документации.

Недостатки:

* Тестирование осуществляется лишь на заключительных стадиях создания ПО. Исходя из этого, если при разработке были допущены ошибки, то на их устранение может уйти много времени и средств. Дело в том, что неполадки будут выявлены уже после написания кода и документации.
* Заказчик может рассмотреть продукт только на заключительных этапах его создания. Таким образом, обратная связь реализуется лишь под конец разработки. Вполне вероятно, что заказчик останется недовольным.
* Модель предполагает написание большого количества технической документации. Это снижает скорость выполнения работ, ведь разработчикам приходится выносить и согласовывать множество решений.

Waterfall предназначена для создания проектов в медицинской и космической сферах. В данных областях уже имеется крупная база данных (включая СНиПы и спецификации). Благодаря этим документам можно гораздо быстрее формировать требования к будущему продукту.

Важнейшая цель в процессе работы с «водопадом» заключается в скрупулезном описании требований к разработке. Необходимо избежать ситуации, при которой на стадии тестирования будет выявлена серьезная ошибка.

Модели разработки программного обеспечения

**V-образная модель (разработка через тестирование)**

Данную модель можно назвать улучшенной версией «водопада». Заказчик совместно с командой разработчиков формирует требования к системе и описывает, каким образом будет выполняться ее тестирование на каждой стадии. V-образную модель начали использовать в 1980-х.

Недостатки:

* Как и при использовании каскадной модели, если во время создания архитектуры были допущены ошибки, то будет довольно сложно их устранить.

Разработка через тестирование является оптимальным вариантом для проектов, в которых нужна повышенная надежность. Скажем, при создании подушек безопасности для автомобилей или систем наблюдения за пациентами в медицинских учреждениях.

**Incremental Model (инкрементная модель)**

Английское слово increment можно перевести как «приращение». Данную модель начали использовать ещё в 1930-х. В качестве примера возьмём разработку социальной сети.

Заказчику необходимо создать соцсеть. Он сформировал подробное техзадание. Разработчики предложили сначала создать основные функции в виде страницы с личной информацией и чата. После этого будет проводиться тестирование на реальных пользователях.

Заказчик оценивает продукт и принимает решение о его выпуске. В том случае, если заказчик и пользователи довольны результатом, то дальнейшая работа осуществляется по частям.

Разработчики одновременно организовывают функциональность для загрузки фотографий, обмена документами, прослушивания музыки и прочих операций, которые предварительно согласовываются с заказчиком. Шаг за шагом продукт становится все более совершенным, становясь все более похожим на сформированный эталон.

Incremental Model

Преимущества:

* Отсутствие необходимости больших материальных вложений на начальной стадии. Заказчику нужно лишь оплатить разработку базового функционала. После этого он получает продукт и может его выпустить. Решение о продолжении разработки будет приниматься на основе обратной связи с реальными клиентами.
* Возможность своевременного получения обратной связи для быстрого обновления техзадания. Благодаря такому подходу вероятность получить невостребованный продукт сводится к минимуму.
* Меньшая цена ошибки. При выявлении каких-либо проблем в архитектуре, их можно исправить за меньшую стоимость в сравнении с двумя предыдущими моделями.

Недостатки:

* Каждая из команд разработчиков занимается созданием отдельных функций. Это может привести к несогласованной реализации интерфейса ПП. Во избежание такой ситуации следует ещё на стадии обсуждения технического задания точно определить конечный результат.
* Программисты могут замедлять процесс создания продукта, откладывая настройку основных функций и излишне заостряясь на мелких деталях. Таким образом, необходимо сосредоточиться на управлении разработкой программного обеспечения. По этой причине менеджеру проекта необходимо осуществлять строгий контроль над действиями каждой команды.

Такая модель лучше всего подойдёт при работе с проектами, для которых техническое задание сформировано ещё на начальных этапах, а сам ПП должен в скором времени быть выпущен на рынок.

**Iterative Model (итеративная модель)**

Данная технология разработки программного обеспечения подразумевает, что заказчик может не разбираться в том, какой именно продукт ему нужен. Иными словами, от него не требуется скрупулезно прописывать техническое задание.

Преимущества:

* Возможность оперативно получить обратную связь за счет выпуска минимального продукта. Таким образом, разработчики могут сосредоточиться на фундаментальных функциях программного обеспечения и совершенствовать их, опираясь на стандарты рынка и отзывы пользователей.

Благодаря непрекращающемуся тестированию самими пользователями, программисты могут своевременно выявлять и нивелировать различные ошибки.

Недостатки:

* Применение на старте баз данных или серверов. Проблема заключается в том, что базы данных довольно сложно масштабировать, а серверы зачастую не выдерживают нагрузку. Из-за этого может сложиться ситуация, при которой нужно будет переделывать весомую часть продукта.
* Отсутствие фиксированного бюджета и сроков. Заказчик не имеет четкого представления о том, каким должен быть итоговый результат и в какое время необходимо завершить создание продукта.

Данная модель будет предпочтительна в том случае, если предполагается работа над крупномасштабным проектом с нечеткими требованиями. Кроме того, итеративный вариант подойдёт для задач с инновационным подходом, когда заказчик не может знать, что получится в конечном итоге.

**Spiral Model (спиральная модель)**

При применении спиральной модели заказчик и исполнители производят тщательный анализ рисков проекта и реализуют его итерациями. Каждая следующая стадия базируется на предыдущей. При этом в конце каждого цикла итераций необходимо принять решение относительно того, будет ли осуществляться разработка дальше.

Модель начали применять ещё в 1988 году. Она схожа с инкрементным вариантом, однако здесь упор делается именно на оценку всевозможных рисков. Каждый новый цикл спирали все больше усложняет процесс.

Преимущества:

* Сосредоточение внимания на анализе рисков.

Недостатки:

* Вероятность того, что разработка слишком зациклится на самой начальной стадии.
* Повышенная стоимость и длительность разработки.

**Гибкие подходы к разработке программного обеспечения**

На базе семейства итеративных моделей был создан крайне распространенный на данный момент вариант разработки — Agile. Это скорее является подходом, нежели целостной методологией. Дело в том, что внутри проекта на различных стадиях допускается использование как итерационных, так и каскадных моделей.

Гибкие подходы к разработке программного обеспечения

Суть данного подхода заключается в дифференцировании процесса разработки на несколько отдельных задач. Программисты могут выполнять эти задачи с высоким уровнем независимости друг от друга. Каждый день организовываются встречи команды (Scrum), в рамках которых проговаривается нынешнее состояние проекта. Разработку дифференцируют на несколько стадий-спринтов (Sprint). Во время прохождения этих спринтов разработчики должны выполнить поставленные цели.

Данный подход полезен в том случае, если заказчик имеет множество идей, но на старте работ ещё не знает, какая часть из них действительно будет актуальна. Помимо этого, у заказчика могут появляться новые идеи прямо в процессе реализации проекта. Применение Agile также имеет смысл при работе с крупными проектами, которые рассчитаны на длительный жизненный цикл. Такие ПП необходимо беспрестанно адаптировать к изменяющимся рыночным условиям.

Преимущества:

* Создание программного обеспечения можно начать без детализированного плана. Необходимо лишь сформулировать несколько общих идей.
* Заказчик может контролировать даже самые мелкие изменения продукта и корректировать его прямо в процессе разработки. Это минимизирует риск возникновения проблем на заключительных стадиях.
* Разработка не требует столь больших финансовых вложений. Кроме того, благодаря коротким спринтам отпадает необходимость в постоянном приспособлении проекта к изменениям рынка.

Недостатки:

* По причине отсутствия четкого плана может сложиться ситуация, при которой будет довольно проблематично дать адекватную оценку бюджету и времени, которое потребуется на реализацию проекта.
* Большой риск краха на старте. Из-за этого необходима гибкость бюджета.

В рамках данного подхода применяется целый ряд всевозможных техник, методологий и приемов. Выделим некоторые из них:

* **Scrum**. Участники проекта находятся в постоянном взаимодействии и обсуждают текущий прогресс.
* **Kanban**. Используется виртуальная доска с задачами. Последовательность выполнения таких задач не определяется.
* **RUP**. Наличие четких стадий планирования, уточнения и построения новых итерация программного обеспечения.
* **Extreme Programming**. Частые обновления версии продукта и отыскивание наиболее скоростных решений.

Гибкие подходы к разработке программного обеспечения

Каждый из вышеописанных методов предполагает готовность к внесению корректив и взаимодействию с заказчиком. Во главе угла ставится разработка полезного программного обеспечения и самоорганизация участников проекта, тогда как ведение документации и формальные обязанности сторон отходят на второй план.

Говоря о гибких методологиях, следует отдельно упомянуть так называемую бережливую разработку ПО Lean. Ее целью является увеличение уровня эффективности создания продукта и повышение результативности всех рабочих процессов. Иными словами, разработка организуется таким образом, чтобы на реализацию проекта ушло меньше денег и времени.

Важнейшей задачей в данном случае является собрание опытной команды, а также ее последующее обучение. Во избежание различных проблем необходимо принимать наиболее рациональные решения, а также своевременно обсуждать текущее состояние проекта с заказчиком.

**Навыки и умения разработчика ПО**

Разработчик ПО является специалистом в области IT, который создает всевозможные программы для компьютера.

Он может работать над корпоративным софтом, видеоигрой, программой для ПК и многим другим, пользуясь различными средствами разработки программного обеспечения.

Такой специалист должен обладать следующими навыками:

* знание как минимум одного языка программирования;
* понимание принципов ООП, алгоритмом и структур данных;
* умение работать с ОС, сетевыми протоколами и методами обмена информацией по сети;
* владение инструментами тестирования и отладки кода.

Фрондендеру необходимо уметь:

* создавать динамичный, интерактивный интерфейс по макету;
* работать с деталями и знать нюансы поставленной задачи, чтобы обеспечить удобство эксплуатации продукта;
* использовать принципы адаптивной верстки. Это позволяет создать мультиплатформенный продукт.

Бэкендер выполняет следующие задачи:

* разрабатывает бэкенд-программы на одном из языков;
* взаимодействует с файловой системой, алгоритмами поиска и сортировки;
* выполняет настройку интеграции с базами данных, формирует запросы;
* участвует в обеспечении сетевой безопасности и организует защиту программного обеспечения от различных вирусов и атак.

Full stack представляет собой программиста широкого профиля. Он может выполнять все задачи связанные с созданием ПО, включая формирование клиентской и серверной части продукта. Ему необходимо обладать следующими умениями:

* знание нескольких языков программирования, распространенных библиотек и фреймворков;
* навыки работы в системе управления версиями Git, применение для сборки и развертывания приложения Docker или Kubernetes;
* понимание шаблонов проектирования и владение гибкими методологиями (скажем, Agile).

Следует отметить, что система разработки программного обеспечения состоит из множества процессов, во время которых зачастую необходимо применять различные подходы, техники и методологии. Каждый вариант разработки обладает своими плюсами и минусами, а также наиболее подходящей областью применения.