Тестирование ПО. Уровень 1

**Место тестирования в процессе разработки ПО**

Цикл разработки ПО. Цикл тестирования ПО. Требования и их атрибуты

Оглавление

[Жизненный цикл разработки ПО](#_ct1x1a5jokng)

[Модели и методологии разработки ПО](#_xrar1bvfmnf4)

[Модели разработки](#_7payhnt3svg5)

[Водопадная модель разработки](#_6mpdyz5gd6tr)

[V-образная модель разработки](#_kzz7m4c8sswc)

[Итерационная инкрементальная модель](#_1ix99noxqek7)

[Спиральная модель](#_dnaepvyr9o9r)

[Методологии разработки](#_fg2y7otdkq41)

[Scrum](#_cd0owx9yddq7)

[Kanban](#_b67xoqnw6esr)

[Жизненный цикл тестирования](#_vzy1i0kwvxtb)

[Работа с требованиями](#_agqkqyscbyvm)

[Что такое требования?](#_e6jxvpeomdbj)

[Виды и уровни требований](#_3mdq94ovcvns)

[Бизнес-требования](#_m7rbk427h29s)

[Пользовательские требования](#_9lujlhz4p60f)

[Детальные требования](#_ape78abf4hjx)

[Функциональные требования](#_9qu6wxtn4xky)

[Нефункциональные требования](#_a78r9saft7pm)

[Что не является требованиями?](#_jyqhp7fm9mfr)

[Атрибуты требований](#_odhn59xm5jez)

[Корректность](#_pj19z5tnl5uw)

[Проверяемость](#_klupcal6ue03)

[Непротиворечивость](#_kjzwb6yv34x)

[Атомарность](#_g0bb195zbysx)

[Актуальность](#_2qkoxpbjqt1u)

[Приоритетность](#_s9gnj6685vgm)

[Модифицируемость](#_3rqju7xs2135)

[Прослеживаемость](#_muw025w5fy63)

[Выполнимость](#_un7viomjskck)

[Полнота](#_pbm1gtp5t58g)

[Недвусмысленность](#_gpg5chklqg1c)

[Глоссарий](#_9f1hh1isqmfk)

[Практическое задание](#_28gatnhpyjlb)

[Требования к выполненной работе](#_ojryxrhea6os)

[Дополнительные материалы](#_30auu7nd1sa)

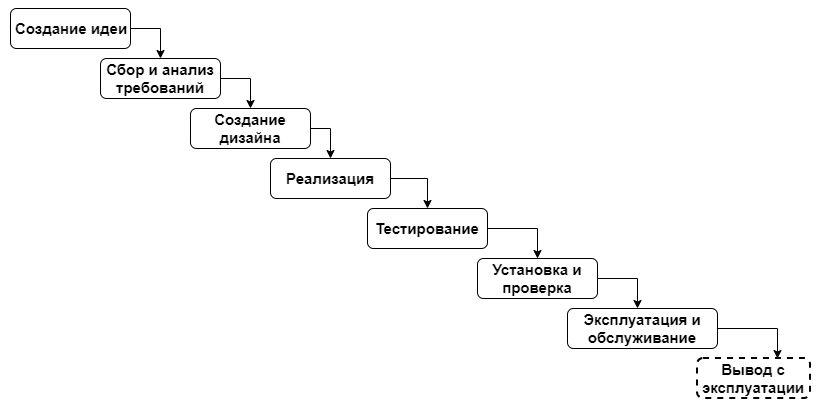
# **Жизненный цикл разработки ПО**

В классическом представлении, каждый разрабатываемый продукт проходит определенные этапы или стадии разработки, которые в целом объединены в жизненный цикл разработки ПО.

В соответствии с ISTQB, **жизненный цикл программного обеспечения** — период времени, который начинается, когда программный продукт задуман, и заканчивается, когда программное обеспечение больше не доступно для использования. Жизненный цикл программного обеспечения обычно включает в себя этап концепции (то есть рождение идеи), этап составления/написания требований, этап проектирования, этап внедрения, этап тестирования, этап установки и проверки, этап эксплуатации и обслуживания, а иногда и этап вывода из эксплуатации. Обратите внимание, что эти фазы могут перекрываться или выполняться итеративно.

Если немного короче, то **жизненный цикл разработки ПО (Software lifecycle)** — этапы, которые проходит система в процессе ее создания и дальнейшего использования (от момента создания до конца разработки и внедрения)

Схематично этапы\фазы жизненного цикла разработки ПО (далее жизненный цикл ПО) можно представить так:



Основные этапы:

* Создание концепции.
* Создание требований.
* Составление дизайна продукта.
* Этап реализации.
* Этап тестирования.
* Установка и проверка.
* Этап эксплуатации и обслуживания.
* Иногда включается этап вывода из эксплуатации.

Вопросы, касающийся обсуждения и рассмотрения жизненного цикла, часто задают во время собеседований.

# Модели и методологии разработки ПО

Часто в разных источниках информации можно встретить такие два понятия, как модель жизненного цикла разработки ПО (далее модель разработки) и методология разработки ПО (далее методология разработки). Часто их объединяют под одним термином\понятием — **модель разработки ПО**.

В уроке мы рассмотрим модель и методологию разработки как два разных понятия.

**Модель жизненного цикла ПО** описывает, какие этапы проходит ПО от рождения идеи до завершения использования, и что происходит с ПО на этих этапах. Мы с вами рассмотрели это на примере.

**Методология разработки ПО** — набор методов по управлению разработкой ПО, практических правил и техник разработки.

Когда говорят о методологиях, чаще всего подразумевают гибкие методологии, такие как Scrum или Kanban, а также говорят про гибкую разработку ПО.

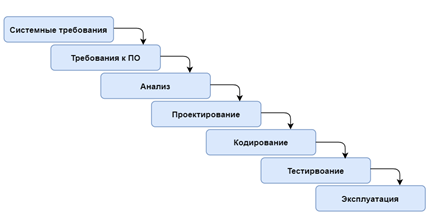
В ISTQB используется такое понятие, как **гибкая разработка** — группа методологий разработки ПО, основанная на итеративной поэтапной разработке, где требования и решения развиваются посредством общения/взаимоотношения между самоорганизованными кросс-функциональными командами.

## Модели разработки

Есть несколько основных моделей, которые до сих пор существуют и используются — водопадная модель, V-модель, инкрементная, итеративная и спиральная модели.

### Водопадная модель разработки

Модель характеризуется чётким переходом между этапами. Вы не можете начать следующий этап, не закончив первый. Но, перейдя на следующий этап, вы не можете вернуться назад, либо возврат очень сложен. Тестирование в этой модели начинается поздно. Это один из больших минусов такой модели.



Эта модель была создана в 1970 году. Отличительная характеристика — строгая последовательность этапов. Переход на последующий этап происходит после полного завершения предыдущего. Также для данной модели разработки характерно наличие детальной (исчерпывающей) документации, чёткое понимание сроков, в которые нужно получить готовый продукт. Также для этой модели характерно, что тестирование наступает на одной из последних фаз (этапов), что однозначно недостаток этой модели.

Преимущества модели:

1. у каждой стадии есть чёткий проверяемый результат;
2. в каждый момент времени команда выполняет один вид работы;
3. проста в изучении;
4. легко контролировать;
5. может использоваться технически слабо подготовленными командами или неопытным персоналом;
6. прогнозирование стоимости.

Недостатки модели:

1. недостаточная гибкость и адаптируемость;
2. позднее получение результата;
3. документация может быть избыточной;
4. позднее обнаружение конструктивных ошибок;
5. позднее представление результатов заказчику;
6. сложность (иногда невозможность) возврата к предыдущим этапам;
7. отсутствие обратной связи от пользователей;
8. сложность изменений и сильное влияние этих изменений на сроки и стоимость проекта.

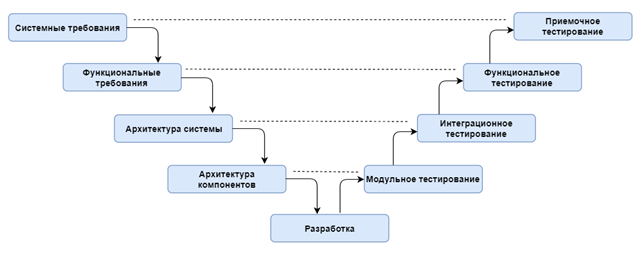
Использование:

* Крупные проекты со стабильными требованиями (космическая отрасль, медицинское ПО).
* Позитивный опыт разработки аналогичных систем.
* Создание новой версии ранее разработанного продукта, когда вносимые изменения определены и управляемы.
* Перенос уже существующего продукта на новую платформу.

### V-образная модель разработки

Модель была создана в конце 1980-х годов. Это усовершенствованная водопадная модель разработки. На этапе составления бизнес-требований сразу же составляется набор тестов для приемочного тестирования. Этапу написания функциональных требований соответствует этап написания функциональных тестов. Этапу написания архитектуры системы соответствует этап написания интеграционных тестов. Этапу написания архитектуры компонентов соответствует этап написания модульных тестов.

Отличительная характеристика — последовательность этапов. Тестирование происходит на начальных этапах.



Преимущества модели:

1. у каждой стадии есть чёткий проверяемый результат;
2. простота в использовании;
3. планирование и проектирование тестирования на ранних этапах разработки;
4. верификация и аттестация всех внешних и внутренних полученных данных, а не только конечного продукта;
5. тестирование начинается с первого же этапа.

Недостатки модели:

1. недостаточная гибкость и адаптируемость;
2. документация может быть избыточной;
3. отсутствует раннее прототипирование;
4. сложно вносить изменения;
5. нет действий, направленных на анализ рисков;
6. сложность устранения проблем, пропущенных на предыдущих этапах.

V-модель применима к системам, которым особенно важно бесперебойное функционирование. Это могут быть прикладные программы в клиниках для наблюдения за пациентами, интегрированное ПО для механизмов управления аварийными подушками безопасности в транспортных средствах.

Эта модель направлена на тщательную проверку и тестирование продукта, находящегося на первоначальных стадиях проектирования.

Использование:

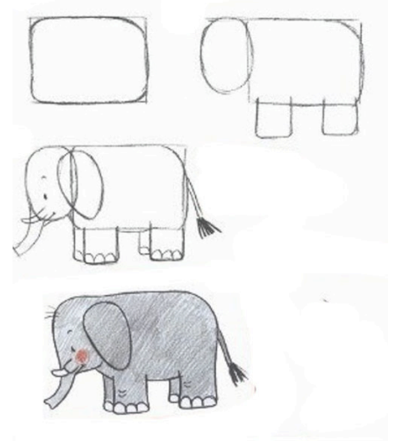
* Для управления разработкой ПО немецкой администрацией.
* Стандарт для немецких правительственных и оборонных проектов.
* Системы, в которых требуется высокая надежность (например, прикладные программы для наблюдения за пациентами в клиниках).
* Встроенное ПО для устройств управления аварийными подушками безопасности в автомобилях.

### Итерационная инкрементальная модель

Итерационная инкрементальная модель — фундаментальная основа современного подхода к разработке ПО. Мы её будем рассматривать как одну модель, которой присуща двойственность. Бывает, что рассматривают как две разные модели — итерационную и инкрементальную.

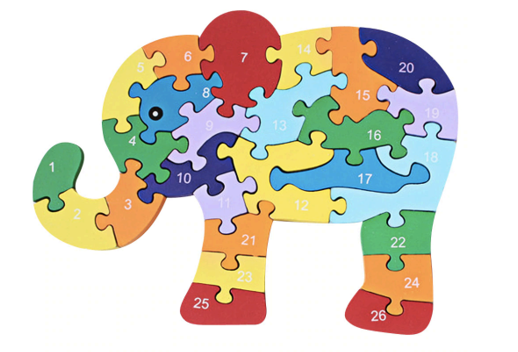
Если рассматривать по отдельности, ей присуща определенная двойственность: с точки зрения жизненного цикла модель — итерационная, т. к. подразумевает многократное повторение одних и тех же стадий.

Визуально это можно представить следующим образом:



С точки зрения развития продукта (приращения его полезных функций) модель — инкрементальная.

Визуально это можно представить следующим образом:





Модель появилась в 30-е годы прошлого века. Её ключевая особенность — разбиение проекта на относительно небольшие промежутки (итерации), каждый из которых в общем случае может включать в себя все классические стадии, присущие водопадной и v-образной моделям. Итог итерации — приращение (инкремент) функциональности продукта, выраженное в промежуточном билде (build).

Преимущества модели:

1. позволяет уменьшить затраты на первоначальных этапах разработки (до достижения уровня исходной производительности);
2. ускоряет создание функционирующей системы;
3. позволяет получать отзывы заказчика в процессе разработки;
4. снижает риск неудачи при изменении требований;
5. раннее получение прототипа, который можно показывать заказчику;
6. декомпозиция проекта на управляемые итерации.

Недостатки данной модели:

1. не всегда возможно определить полную функциональность ПО в начале разработки;
2. Т. к. модули разрабатываются в разное время, необходимо четко определить интерфейсы;
3. сложность приоритизации инкрементов;
4. может возникать тенденция к откладыванию сложных решений на будущее.

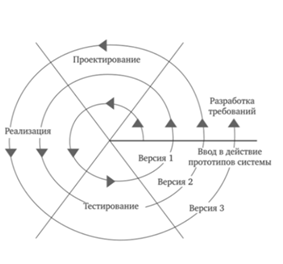
Использование:

Проекты, в которых требования сформулированы заранее и необходима быстрая поставка продукта на рынок.

### Спиральная модель

Спиральная модель представляет собой частный случай итерационной инкрементальной модели, в котором особое внимание уделяется управлению рисками, в особенности влияющими на организацию процесса разработки проекта и контрольные точки.

Схематично суть спиральной модели представлена на рисунке.



Модель начали использовать с 1988 года. Она характеризуется особым вниманием на управление рисками. Также для модели характерно, что каждый виток спирали — готовая часть продукта, и в конце каждого витка принимается решение о продолжении проекта. Также каждая последующая стадия основывается на предыдущей.

Главная же задача — как можно быстрее показать пользователям системы работоспособный продукт, тем самым активизируя процесс уточнения и дополнения требований. С точки зрения тестирования и управления качеством, повышенное внимание рискам — ощутимое преимущество при использовании спиральной модели для разработки концептуальных проектов, в которых требования естественным образом сложны и нестабильны (могут многократно меняться по ходу выполнения проекта).

Модель не подходит для малых проектов. Модель уместна для сложных и дорогих проектов, таких как разработка системы документооборота для банка — когда каждый следующий шаг требует большего анализа для оценки последствий, чем программирование само по себе. Или проекты с высоким риском, или исследовательские проекты, или проекты, в которых изначально неизвестно, возможна ли их реализация в текущих условиях.

Преимущества модели:

1. высокий уровень анализа рисков;
2. активное участие заказчиков, особенно на стадии планирования;
3. совершенствование процесса на каждом витке.

Недостатки модели:

1. высокая стоимость разработки;
2. сложно определить момент перехода на следующий виток спирали;
3. ложная структура, трудная для понимания;
4. спираль может продолжаться бесконечно.

Использование:

Сложные проекты, связанные с системами документооборота в банке.

## Методологии разработки

Гибкие методологии представляют собой совокупность различных подходов к разработке ПО и базируются на т. н. Agile-манифесте:

«Мы постоянно открываем для себя более совершенные методы разработки программного обеспечения, занимаясь разработкой непосредственно и помогая в этом другим. Благодаря проделанной работе мы смогли осознать, что:

* Люди и взаимодействие важнее процессов и инструментов.
* Работающий продукт важнее исчерпывающей документации.
* Сотрудничество с заказчиком важнее согласования условий контракта.
* Готовность к изменениям важнее следования первоначальному плану

То есть, не отрицая важности того, что справа, всё-таки ценится то, что слева.»

Выделяют 12 основополагающих принципов Agile-манифеста:

1. Наивысший приоритет для нас — удовлетворение потребностей заказчика благодаря регулярной и ранней поставке ценного программного обеспечения.
2. Изменение требований приветствуется, даже на поздних стадиях разработки. Agile-процессы позволяют использовать изменения для обеспечения заказчику конкурентного преимущества.
3. Работающий продукт следует выпускать как можно чаще, с периодичностью от пары недель до пары месяцев.
4. На протяжении всего проекта разработчики и представители бизнеса должны ежедневно работать вместе.
5. Над проектом должны работать мотивированные профессионалы. Чтобы работа была сделана, создайте условия, обеспечьте поддержку и полностью доверьтесь им.
6. Непосредственное общение — наиболее практичный и эффективный способ обмена информацией как с самой командой, так и внутри команды.
7. Работающий продукт — основной показатель прогресса.
8. Инвесторы, разработчики и пользователи должны иметь возможность поддерживать постоянный ритм бесконечно. Agile помогает наладить такой устойчивый процесс разработки.
9. Постоянное внимание к техническому совершенству и качеству проектирования повышает гибкость проекта.
10. Простота — искусство минимизации лишней работы — крайне необходима.
11. Самые лучшие требования, архитектурные и технические решения рождаются у самоорганизующихся команд.
12. Команда должна систематически анализировать возможные способы улучшения эффективности и соответственно корректировать стиль своей работы.

В основу гибких методологий были заложены подходы, которые логически развивают и продолжают то, что было создано и опробовано в традиционных моделях разработки. Основной эффект был достигнут за счёт снижения бюрократической составляющей и максимальной адаптации процесса разработки ПО к мгновенным изменениям рынка и требований заказчика.

В основе гибких методологиях лежат непродолжительные отрезки времени (итерации, или sprint), во время которых выполняется определенный (ограниченный) список задач (sprint backlog), ежедневные встречи (Stand-up meeting) и регулярные собрания (планирование спринта, итоги спринта). На ежедневных совещаниях участники команды обсуждают:

* отчёт о проделанной работе с момента последнего Stand-up или meeting;
* список задач, которые сотрудник должен выполнить до следующего собрания;
* проблемы или сложности, с которыми столкнулся кто-либо из членов команды.

Методологии подходят для больших или нацеленных на длительный жизненный цикл проектов, постоянно адаптируемых к условиям рынка. Соответственно, в процессе реализации требования изменяются.

Преимущества модели:

1. быстрая поставка рабочего функционала;
2. взаимодействие с заказчиком;
3. возможность получения быстрой обратной связи от конечных пользователей;
4. адаптируемость к изменению внешних условий;
5. возможность корректировки требований в процессе разработки.

Недостатки модели:

1. сложно применима в сложных проектах;
2. опасность возникновения неучтенных рисков;
3. может быть недостаточно документации.

Использование:

Подходит для больших или нацеленных на длительный жизненный цикл проектов, постоянно адаптируемых к условиям рынка.

Наиболее популярные методологии:

1. Scrum.
2. Kanban.
3. XP (Extreme Programming).
4. Lean.
5. FDD (Feature Driven Development).
6. TDD (Test Driven Development).
7. Cleanroom Software Engineering.
8. OpenUp.
9. MSF (Microsoft Solutions Framework).
10. DSDM (Dynamic Systems Development Method).

Коротко разберём две часто встречающиеся на проектах методологии.

### Scrum

Сердце Scrum — Sprint, период времени в один месяц или менее, в течение которого создается новая часть приложения (отдельный модуль или новые функции), пригодная для использования. Спринты имеют одинаковую продолжительность на протяжении всего процесса разработки.

Новый спринт начинается сразу после завершения предыдущего.

### Kanban

Kanban-доска — инструмент управления гибкими проектами, предназначенный для визуализации работы, ограничения незавершённой работы и максимизации эффективности.

Доска использует карточки, колонки и постоянное улучшение с целью помочь командам выполнить правильный объем работы.

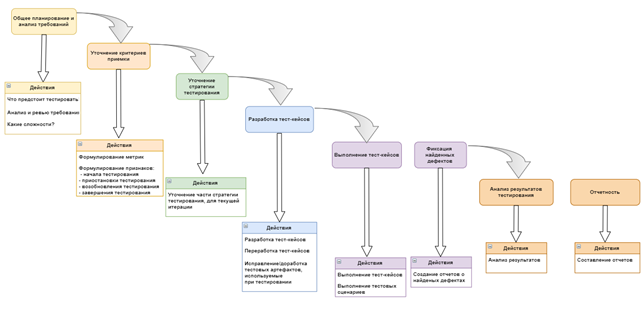
# Жизненный цикл тестирования

В тестировании существует свой жизненный цикл. Каждому из этапов присущи свои активности.

Итак, жизненный цикл тестирования проходит следующие этапы:

1. общее планирование и анализ требований,
2. уточнение критериев приемки,
3. уточнение стратегии тестирования,
4. разработка тест-кейсов,
5. выполнение тест-кейсов,
6. фиксация найденных дефектов,
7. анализ результатов тестирования,
8. отчётность.

Стадии 5 и 6 тесно связаны друг с другом, так же, как и стадии 7 и 8. Поэтому фактически они выполняются параллельно.



**На этапе 1** (общее планирование и анализ требований) отвечают на такие вопросы, как:

* что нам предстоит тестировать;
* как много будет работы;
* какие есть сложности;
* всё ли необходимое у нас есть, и т. п.

Как правило, получить ответы на эти вопросы невозможно без анализа требований, т. к. именно требования — первичный источник ответов.

**На этапе 2** (уточнение критериев приемки) проводится формулирование и уточнение метрик и признаков возможности или необходимости начала тестирования, приостановки тестирования, завершения или даже прекращения тестирования.

**На этапе 3** (уточнение стратегии тестирования) рассматриваются и уточняются те части стратегии тестирования, которые актуальны для текущей итерации.

**На этапе 4** (разработка тест-кейсов) проводится разработка, пересмотр, уточнение, доработка, переработка и прочие действия с тест-кейсами, наборами тест-кейсов, тестовыми сценариями и иными артефактами, которые будут использоваться при непосредственном выполнении тестирования.

**На этапе 5** (выполнение тест-кейсов) проводится выполнение (прохождение) тест-кейсов и поиск дефектов.

**На этапе 6** (фиксация найденных дефектов) фиксируются дефекты по факту их обнаружения в процессе выполнения тест-кейсов.

Этапы 5 и 6 объединяются, поскольку нахождение и фиксация дефектов происходит во время прохождения тест-кейсов.

Зачастую после выполнения всех тест-кейсов и написания всех отчётов о найденных дефектах проводится явно выделенная стадия уточнения, на которой все отчёты о дефектах рассматриваются повторно с целью формирования единого понимания проблемы и уточнения таких характеристик дефекта, как важность и срочность.

**Этапе 7** (анализ результатов тестирования) и **этап 8** (отчётность) объединяют.

Здесь формулируемые на стадии анализа результатов выводы напрямую зависят от плана тестирования, критериев приемки и уточненной стратегии, полученных на стадиях 1, 2 и 3.

# Работа с требованиями

## Что такое требования?

Вспомним еще раз, что такое требования.

**Требование (requirement)** — описание того, какие функции и с соблюдением каких условий должно выполнять приложение в процессе решения полезной для пользователя задачи.

Требования — отправная точка для тестировщиков и тестирования, поскольку:

1. Позволяют понять, что и с соблюдением каких условий система должна делать.
2. Предоставляют возможность оценить масштаб работ.
3. Они — основа для формирования плана проекта (в том числе плана тестирования).
4. Помогают предотвращать или разрешать конфликтные ситуации.
5. Упрощают расстановку приоритетов в наборе задач.
6. Позволяют объективно оценить степень прогресса в разработке проекта.

Проверка (или тестирование) требований — получение подтверждения от заинтересованных сторон и проектной команды, что требования полные, точные, совместимые. Другими словами, согласование и утверждение сформированных требований.

Почему мы должны это делать?

1. уменьшить количество переделок и изменений;
2. снизить риски;
3. ознакомить команду с задачами и согласовать действия.

Кто тестирует?

* тестировщики,
* аналитики,
* разработчики,
* менеджеры.

Ниже графически представлен рост стоимости исправления ошибки в зависимости от того, на каком этапе она была обнаружена.



Статистика показывает, что больше половины дефектов находятся в требованиях.

## Виды и уровни требований

Есть два подхода к выделению требований.

В соответствии с первым подходом, требования подразделяют на две большие группы — функциональные и нефункциональные.

В соответствии со вторым подходом, требования разделяют на бизнес-требования, требования пользователей (или пользовательские требования) и детальные требования.



### Бизнес-требования

Бизнес-требования (business requirements) выражают цель, ради которой разрабатывается продукт, и отвечают на вопросы:

1. Зачем он нужен?
2. Какая от него ожидается польза?
3. Как заказчик с его помощью будет получать прибыль?

Результат выявления требований — общее видение: документ, представленный простым текстом и таблицами. Нет детализации поведения системы и иных технических характеристик. Могут быть определены приоритеты решаемых бизнес-задач, риски и т. п.

Примеры бизнес-требований:

* Нужен инструмент, в реальном времени отображающий наиболее выгодный курс покупки и продажи валюты.
* Необходимо в два-три раза повысить количество заявок, обрабатываемых одним оператором за смену.
* Нужно автоматизировать процесс выписки товарно-транспортных накладных на основе договоров.

### Пользовательские требования

Пользовательские требования описывают задачи, которые пользователь может выполнять с помощью разрабатываемой системы. Требования этого уровня могут быть использованы для оценки объема работ, стоимости проекта и времени разработки.

Примеры пользовательских требований:

* При первом входе пользователя в систему должно отображаться лицензионное соглашение.
* Администратор должен иметь возможность просматривать список всех пользователей, работающих в данный момент в системе.
* При первом сохранении новой статьи система должна выдавать запрос на сохранение в виде черновика или публикацию.

### Детальные требования

Детальные требования включают в себя несколько категорий требования:

* Системные требования.
* Функциональные требования.
* Ограничения.
* Требования к внешнему виду и интерфейсу приложения.

Они определяют свойства, которыми должна обладать система. Показывают, что ожидает пользователь от системы.

В детальных требованиях указывается и перечисляется как должны действовать система или как должна отвечать на то или иное действие пользователя. Описывают требования к взаимодействию отдельных подсистем программного обеспечения, как программных, так и аппаратных. Могут описываться требования к базам данных (описывать их структуру и их особенности).

Примеры детальных требований:

* Система должна обеспечивать возможность открытия формы заказа путем считывания сканером штрих-кода, напечатанного на сопроводительном документе.
* Система должна обеспечивать возможность выбора заказа.
* Система должна обеспечивать возможность поиска заказа.
* Для установки приложения операционная система должна быть Windows Server 2012R2.
* Для установки приложения сервер должен иметь не менее 16 Гб оперативной памяти (ОЗУ).

### Функциональные требования

Функциональные требования описывают поведение системы, т. е., её действия.

Примеры функциональные требований:

* В процессе инсталляции приложение должно проверять остаток свободного места на целевом носителе.
* Система должна автоматически выполнять резервное копирование данных ежедневно в указанный момент времени.
* Электронный адрес пользователя, вводимый при регистрации, должен быть проверен на соответствие требованиям RFC822.

### Нефункциональные требования

Нефункциональные требования описывают свойства системы, которыми она должна обладать при реализации своего поведения.

Примеры нефункциональных требований:

* При одновременной непрерывной работе с системой 1000 пользователей, минимальное время между возникновением сбоев должно быть более или равно 100 часов.
* Ни при каких условиях общий объем используемой приложением памяти не может превышать 2 ГБ.
* Размер шрифта для любой надписи на экране должен поддерживать настройку в диапазоне от 5 до 15 пунктов.

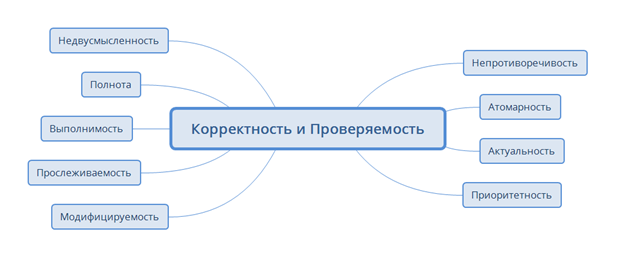
## Что не является требованиями?

Требованиями не являются:

1. Соображения относительно управления проектом.
2. Соображения относительно проведения тестирования.
3. Соображения о применении системы.
4. Соображения относительно реализации требований в коде.
5. Устные утверждения или результаты переговоров в коридорах.

## Атрибуты требований

У требований есть свои атрибуты, при соблюдении которых можно утверждать, что требования качественные и точные.



Как правило, к атрибутам требований относят:

1. Корректность.
2. Проверяемость.
3. Непротиворечивость.
4. Атомарность.
5. Актуальность.
6. Приоритетность.
7. Модифицируемость.
8. Прослеживаемость.
9. Выполнимость.
10. Полнота.
11. Недвусмысленность.

### Корректность

Каждое требование должно точно описывать желаемую функциональность.

Ошибки корректности:

* Ошибки в тексте (Copy-Paste, опечатки).
* Устаревшие требования.
* Технически невозможные функции.
* Ложные факты.

### Проверяемость

Требование должно быть сформулировано так, чтобы существовали способы однозначной проверки, выполнено оно или нет.

Ошибки проверяемости:

* Незаконченность.
* Двусмысленность.
* Неполнота.
* Некорректность.

### Непротиворечивость

Требование не должно содержать внутренних противоречий и противоречий другим требованиям и документам.

Ошибки непротиворечивости:

* Противоречия внутри одного требования.
* Противоречия между разными требованиями.
* Если два требования вступают в конфликт, то как минимум одно из них некорректно.

### Атомарность

Требование является атомарным, если его нельзя разбить на отдельные требования без потери завершенности, и оно описывает одну и только одну ситуацию.

Ошибка атомарности: в одном требовании содержатся фактически несколько независимых требований или ситуаций.

### Актуальность

Требование должно быть актуальным к текущей версии выпуска программного продукта, то есть оно не устарело с течением времени.

Ошибка актуальности: требование перестало быть актуальным по отношению к текущей версии продукта.

### Приоритетность

Приоритет требования представляет собой количественную оценку степени значимости (важности) требования.

Ошибки приоритетности:

* Все требования одинаково важны.
* 90% требований имеют наивысший приоритет.

### Модифицируемость

Это свойство характеризует простоту внесения изменений в отдельные требования и в их набор.

Ошибки модифицируемости:

* Требования представлены в неудобной для обработки форме.
* Присутствует избыточность.
* Слишком сложная структура (либо совсем отсутствует).

### **Прослеживаемость**

У каждого требования должен быть уникальный идентификатор, по которому на это требование можно сослаться.

Ошибки прослеживаемости:

* Требования не пронумерованы, не структурированы, не имеют оглавления, работающих перекрестных ссылок.
* Набор требований неполный, с явными пробелами.
* Ссылки на старые /удаленные документы.

### Выполнимость

Требование может быть реализовано в пределах проекта.

Ошибки выполнимости:

* Требование не может быть реализовано из-за архитектурных особенностей реализации приложения;
* Требование не учитывает особенности используемых технологий (протоколов передачи данных и т .д.).

### Полнота

Каждое требование должно содержать всю информацию, необходимую разработчику, чтобы правильно спроектировать и реализовать требуемую функциональность.

Ошибки полноты:

* Отсутствуют нефункциональные составляющие требования.
* Возникают предположения (из-за соображений «это и так всем понятно»).
* Слишком общие фразы.
* Что-то не дописано (или TBD).

### Недвусмысленность

Требование описано без неочевидных аббревиатур и расплывчатых формулировок и допускает только однозначное объективное понимание.

К ошибкам недвусмысленности относятся:

* Использование терминов или фраз, допускающих субъективное толкование: удобный, быстрый, простой, улучшенный, гибкий, союз «И».
* Использование неочевидных или двусмысленных аббревиатур без расшифровки.

# Глоссарий

**Жизненный цикл программного обеспечения** — период времени, который начинается, когда программный продукт задуман, и заканчивается, когда программное обеспечение больше не доступно для использования. Жизненный цикл программного обеспечения обычно включает в себя этап концепции (то есть рождение идеи), этап составления/написания требований, этап проектирования, этап внедрения, этап тестирования, этап установки и проверки, этап эксплуатации и обслуживания, а иногда и этап вывода из эксплуатации. Обратите внимание, что эти фазы могут перекрываться или выполняться итеративно.

**Гибкая разработка** — группа методологий разработки ПО, основанная на итеративной поэтапной разработке, где требования и решения развиваются посредством общения/взаимоотношения между самоорганизованными кросс-функциональными командами.

# Практическое задание

**Используя приложенную таблицу, выполните следующие задания:**

1. Прочитать в книге Святослава Куликова (Software Testing — Base Course (Svyatoslav Kulikov)) страницы 18-64.
2. Зарегистрироваться на Trello. Создать доску в Trello.
3. Пройти первое задание (Testing Challenge #1 — web testing) на сайте<http://testingchallenges.thetestingmap.org/>.
4. \* Посмотреть [вебинар по моделям и методологиям](https://geekbrains.ru/events/1250).

## Требования к выполненной работе

1. Домашнее задание должно быть выполнено в приложенной форме.
2. Расширение файла должно быть формата Excel (XLS, XLSX).
3. Название файла не менять, вместо ФИО укажите свои фамилию и имя.

# Дополнительные материалы

1. [Руководство по использованию Scrum](https://www.scrum.org/resources/scrum-guide).
2. [Принципы, на которых базируется Agile-манифест](https://agilemanifesto.org/principles.html).
3. [The Scrum Guide.](https://www.scrumguides.org/scrum-guide.html)
4. [Scrum vs Kanban: в чем разница и что выбрать?](https://habr.com/ru/company/hygger/blog/351048/)
5. [Что должен уметь начинающий тестировщик](http://testbase.ru/).
6. [Чит-лист регистрации от Алексея Лупана](http://wiki.software-testing.ru/%D0%A7%D0%B8%D1%82-%D0%BB%D0%B8%D1%81%D1%82_%D1%80%D0%B5%D0%B3%D0%B8%D1%81%D1%82%D1%80%D0%B0%D1%86%D0%B8%D0%B8_%D0%BE%D1%82_%D0%90%D0%BB%D0%B5%D0%BA%D1%81%D0%B5%D1%8F_%D0%9B%D1%83%D0%BF%D0%B0%D0%BD%D0%B0).
7. [Чит-лист по Web UI контролам от Игоря Любина](http://wiki.software-testing.ru/%D0%A7%D0%B8%D1%82-%D0%BB%D0%B8%D1%81%D1%82_%D0%BF%D0%BE_Web_UI_%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D0%B0%D0%BC_%D0%BE%D1%82_%D0%98%D0%B3%D0%BE%D1%80%D1%8F_%D0%9B%D1%8E%D0%B1%D0%B8%D0%BD%D0%B0).

**Используемые источники**

Для подготовки методического пособия мы использовали эти ресурсы:

1. Software Testing — Base Course (Svyatoslav Kulikov).
2. [ISTQB Glossary.](https://glossary.istqb.org/search)
3. [Manifesto for Agile Software Development](https://agilemanifesto.org/).
4. [The Scrum Guide](https://www.scrum.org/resources/scrum-guide).
5. [Agile-манифест разработки программного обеспечения](https://agilemanifesto.org/iso/ru/manifesto.html).
6. [Требования к программному обеспечению](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B5%D0%B1%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%BC%D1%83_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8E).