

# Лекция 2

- Жизненный цикл ПО
- Модели разработки ПО
- Жизненный цикл тестирования
- Классификация методов тестирования

# ЖИЗНЕННЫЙ ЦИКЛ ПО

---

Это период времени, который начинается с

момента принятия решения о необходимости  
создания программного продукта

и заканчивается

в момент его полного изъятия из эксплуатации

***Software Development Life Cycle (SDLC)*** – это жизненный цикл программного обеспечения из набора типовых этапов

1. Планирование.
2. Анализ
3. Проектирование
4. Разработка
5. Тестирование и развертывание
6. Поддержка и сопровождение



# МОДЕЛИ РАЗРАБОТКИ ПО

---

Модель разработки ПО – это структура, систематизирующая различные виды проектной деятельности, их взаимодействие и последовательность в процессе разработки ПО

Выбор той или иной модели зависит от масштаба и сложности проекта, предметной области, доступных ресурсов и множества других факторов.

- ✓ Водопадная модель
- ✓ V-модель
- ✓ Итерационная модель
- ✓ Спиральная модель

# ВОДОПАДНАЯ МОДЕЛЬ

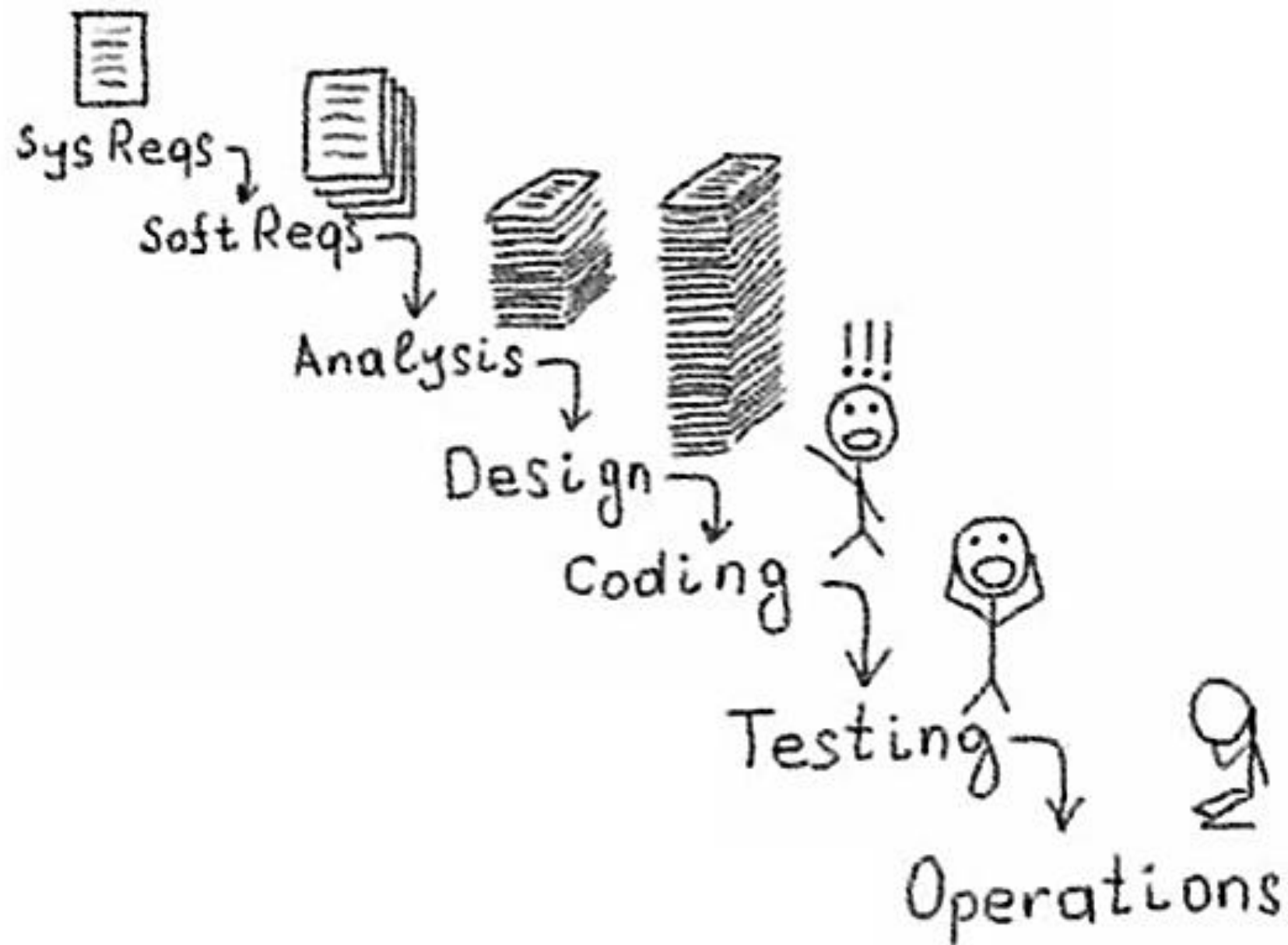
Водопадная модель (waterfall model) сейчас представляет скорее исторический интерес, т.к. в современных проектах практически неприменима (исключением могут быть крупные проекты со стабильными требованиями).



С точки зрения тестирования эта модель плоха тем, что тестирование в явном виде появляется здесь лишь с середины развития проекта, достигая своего максимума в самом конце.

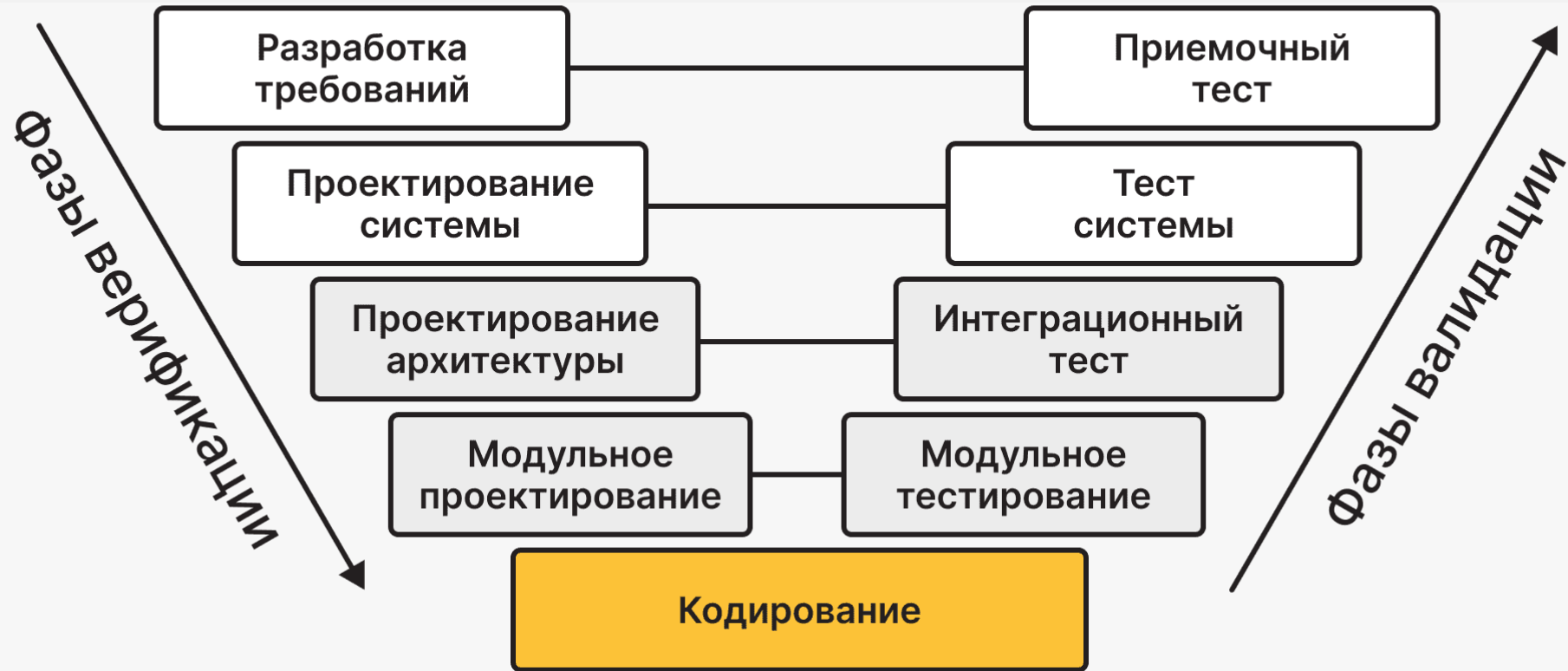
!!! Забавное описание истории развития и заката водопадной модели было создано Максимом Дорофеевым в виде слайдката «The Rise And Fall Of Waterfall», который можно посмотреть по ссылке [https://vk.com/video9468503\\_161602683](https://vk.com/video9468503_161602683)





Управление состоянием ИТ-систем

# Модель верификации и валидации (V&V)



Данная модель считается усовершенствованной версией Waterfall. V&V - это двухэтапный процесс:

- **Верификация** - это процесс проверки соответствия программного обеспечения **требованиям** пользователя.
- **Валидация** - это процесс проверки соответствия программного обеспечения **потребностям** пользователя.

## Чем валидация отличается от верификации?

Верификация — проводится практически всегда, выполняется методом проверки (сличения) характеристик продукции с заданными требованиями, результатом является вывод о соответствии (или несоответствии) продукции.

Валидация — проводится при необходимости, выполняется методом анализа заданных условий применения и оценки соответствия характеристик продукции этим требованиям, результатом является вывод о возможности применения продукции для конкретных условий.

Верификация отвечает на вопрос «Правильно ли записаны требования?», а валидация – на вопрос «А правильные ли требования записаны?».

или

Верификация отвечает на вопрос "Делаем ли мы продукт правильно?", а валидация- на вопрос "Делаем ли мы правильный продукт?"



## Сравнительная таблица

Верификация	Валидация
Делают ли разработчики продукт правильно	Правильный ли получился проект
Все ли функции реализованы	Насколько грамотно реализована функциональность
Предшествует валидации. Включает в себя полноценную проверку правильности написания.	Проводится после верификации. Это – оценка качества итогового проекта.
Испытания организовываются разработчиками	Испытания организованы тестировщиками
Тип анализа – статистический. Проводится сравнение с установленными требованиями к итоговому проекту.	Тип анализа – динамический. Проект проходит испытания по эксплуатации. Это помогает понять, насколько продукт соответствует действующим нормам.
Оценка объективна. Она базируется на соответствии определенным стандартам.	Оценка субъективна. Она является личной. Это – оценка, которую ставит каждый тестировщик.

# V-МОДЕЛЬ

Тестирование появляется на ранних стадиях развития проекта. Это позволяет минимизировать риски, а также обнаружить и устранить множество потенциальных проблем.



# ИТЕРАЦИОННАЯ ИНКРЕМЕНТАЛЬНАЯ МОДЕЛЬ



Итерационная инкрементальная модель является фундаментальной основой современного подхода к разработке ПО. Как следует из названия модели, ей свойственна определённая двойственность:

- с точки зрения жизненного цикла модель является итерационной, т.к. подразумевает многократное повторение одних и тех же стадий;
- с точки зрения развития продукта (приращения его полезных функций) модель является инкрементальной.

Ключевой особенностью данной модели является 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. Итогом итерации является приращение (инкремент) функциональности продукта, выраженное в промежуточном билде. Это позволяет гарантировать, **что и тестирование, и демонстрация продукта конечному заказчику (с получением обратной связи) будет активно применяться с самого начала и на протяжении всего времени разработки проекта.**

Во время каждой итерации тестирование проводится параллельно с разработкой. Тестировщики выполняют тесты, где основное внимание уделяется функциональному тестированию. Специалисты по тестированию оценивают качество программного обеспечения, выявляют области, требующие улучшения, и предлагают усовершенствования для обеспечения лучшей функциональности и удобства использования.

# СПИРАЛЬНАЯ МОДЕЛЬ

Проработка целей,  
альтернатив и  
ограничений

Анализ  
рисков и  
прототипиро  
вание



Планирование  
следующего  
цикла

Разработка  
(промежуточной  
версии) продукта

**Спиральная модель** представляет собой частный случай итерационной инкрементальной модели, в котором особое внимание уделяется управлению рисками.

С точки зрения тестирования и управления качеством повышенное внимание рискам является ощутимым преимуществом при использовании спиральной модели для разработки концептуальных проектов, в которых требования могут многократно меняться по ходу выполнения проекта.

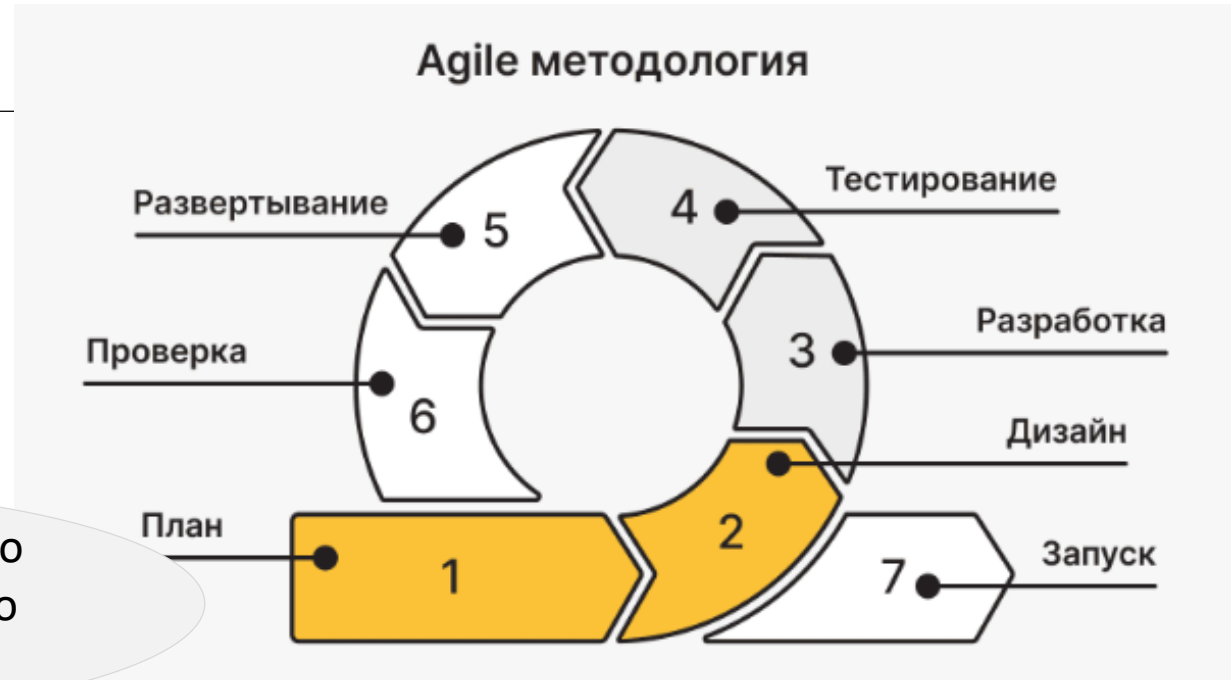
# AGILE методология

Гибкая модель (agile model) представляет собой совокупность различных подходов к разработке ПО и базируется на т.н. «agile-манифесте»

не отрицая важности того, что справа, больше ценим то, что слева.

## AGILE МАНИФЕСТ:

- ✓ **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.



**Методология Agile** особое внимание уделяет **совместной работе команды и обратной связи**. Agile-проекты делятся на небольшие, ограниченные по времени циклы, называемые спринтами. Каждый спринт обычно длится от одной до четырех недель, в течение которого разрабатывается, тестируется и реализуется часть функций ПО.

**Тестирование по методологии Agile** проводится **непрерывно в течение всего проекта**. Тестировщики тесно сотрудничают с разработчиками, заинтересованными сторонами бизнеса и владельцами продуктов для уточнения требований, доработки пользовательских сценариев использования и определения критериев приемки.

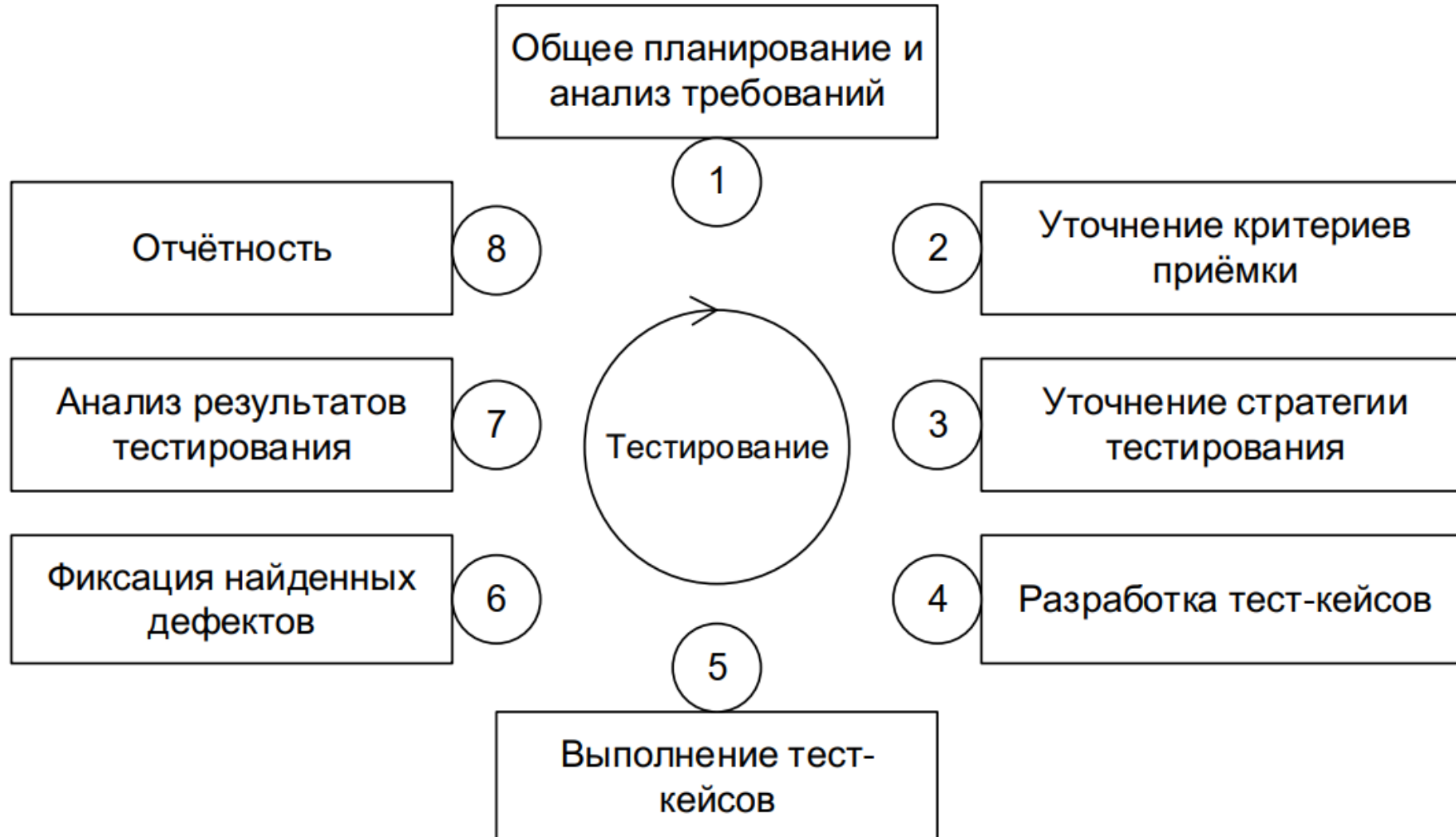
**Тестирование в модели Agile** отличается **гибкостью, адаптивностью и нацеленностью на сотрудничество**. Это позволяет ускорить цикл обратной связи и выявить дефекты на ранней стадии. Тем не менее, такая методология тестирования может привести к увеличению сроков поставки готового программного обеспечения.

# Сравнение моделей разработки ПО

Модель	Тестирование
Водопадная	С середины проекта.
V-образная	На переходах между стадиями
Итерационная инкрементальна	<ul style="list-style-type: none"><li>• В определённые моменты итераций.</li><li>• Повторное тестирование (после доработки) уже проверенного ранее.</li></ul>
Спиральная	
Гибкая	В определённые моменты итераций и в <b>любой необходимый момент.</b>



# Жизненный цикл тестирования



**Стадия 1 (общее планирование и анализ требований)** Что нам предстоит тестировать; как много будет работы; какие есть сложности; всё ли необходимое у нас есть и т.п.

**Стадия 2 (уточнение критериев приёмки)** позволяет сформулировать или уточнить метрики и признаки возможности или необходимости начала тестирования (entry criteria), приостановки (suspension criteria) и возобновления (resumption criteria) тестирования, завершения или прекращения тестирования (exit criteria).

**Стадия 3 (уточнение стратегии тестирования)** планирование, но уже на локальном уровне: рассматриваются и уточняются те части стратегии тестирования (test strategy), которые актуальны для текущей итерации.

**Стадия 4 (разработка тест-кейсов)** посвящена разработке, пересмотру, уточнению, доработке, переработке и прочим действиям с тест-кейсами, наборами тесткейсов, тестовыми сценариями и иными артефактами, которые будут использоваться при непосредственном выполнении тестирования.

**Стадия 5 (выполнение тест-кейсов)** и **стадия 6 (фиксация найденных дефектов)** тесно связаны между собой и фактически выполняются параллельно: дефекты фиксируются сразу по факту их обнаружения в процессе выполнения тест-кейсов. Однако зачастую 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.

**Стадия 7 (анализ результатов тестирования)** и **стадия 8 (отчётность)** также тесно связаны между собой и выполняются практически параллельно. Формулируемые на стадии анализа результатов выводы напрямую зависят от плана тестирования, критериев приёмки и уточнённой стратегии, полученных на стадиях 1, 2 и 3.

**Полученные выводы оформляются на стадии 8 и служат основой для стадий 1, 2 и 3 следующей итерации тестирования. Таким образом, цикл замыкается.**

# Классификация тестирования

Тестирование можно классифицировать по очень большому количеству признаков, на рисунке приведена **упрощенная классификация**





## **Статическое тестирование**

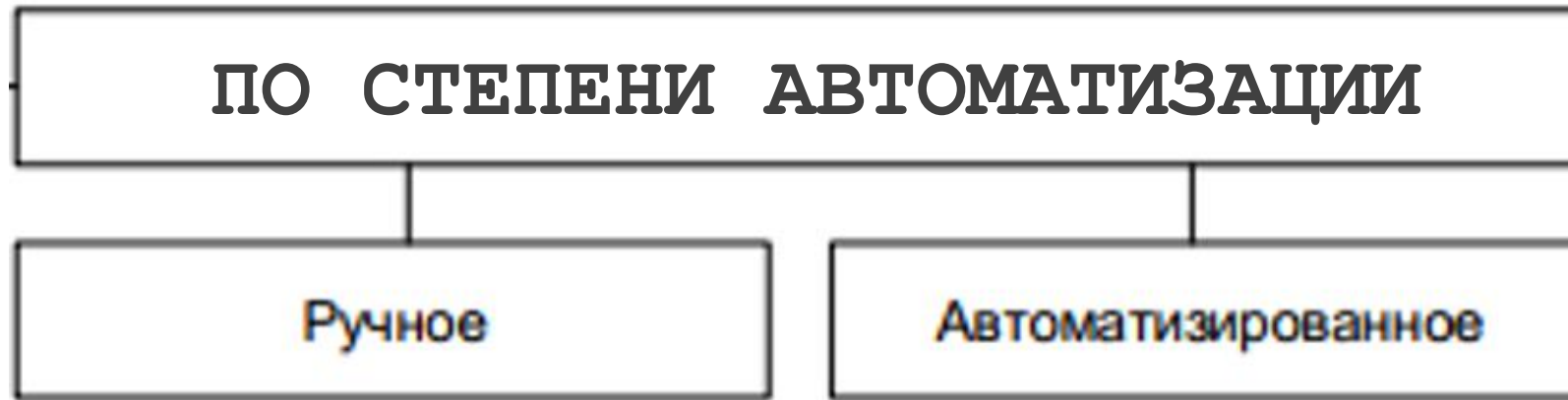
---

Тестирование требований  
Вычитка исходного кода

## **Динамическое тестирование**

---

Модульное тестирование  
Интеграционное тестирование  
Приемочное тестирование



## ПЛЮСЫ РУЧНОГО ТЕСТИРОВАНИЯ

- ✓ Отчет тестировщика;
- ✓ Обратная связь по UI;
- ✓ Низкая стоимость;
- ✓ Мгновенное начало тестирования;
- ✓ Возможность тестировать нетипичные сценарии.

## ПЛЮСЫ АВТО.ТЕСТИРОВАНИЯ

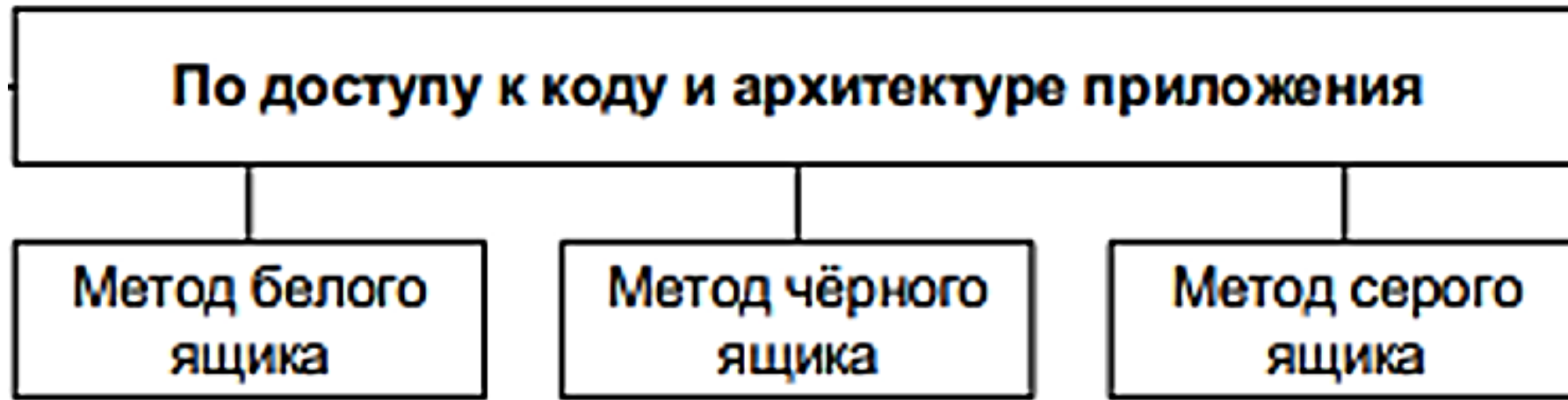
- ✓ Скорость работы;
- ✓ Отсутствие человеческого фактора;
- ✓ Затраты (при многократном использовании);
- ✓ Способность выполнять непосильные для человека тесты;
- ✓ Работа с большими объемами данных;
- ✓ Низкоуровневость;
- ✓ Многократное использование.

## МИНУСЫ РУЧНОГО ТЕСТИРОВАНИЯ

- ✓ Человеческий фактор;
- ✓ Трудозатраты и продолжительность;
- ✓ Отсутствие возможности моделирования большой нагрузки.

## МИНУСЫ АВТОМАТИЗИР. ТЕСТИРОВАНИЯ

- ✓ Необходимость в квалифицированном персонале;
- ✓ Большое количество средств;
- ✓ Низкая адаптивность;
- ✓ Отсутствие обратной связи;
- ✓ Невозможность протестировать дизайн;
- ✓ Относительно низкая надежность;
- ✓ Затраты (при однократном использовании).



## **МЕТОД «БЕЛОГО ЯЩИКА»**

---

Тестирование исходного кода ПО

## **МЕТОД «ЧЕРНОГО ЯЩИКА»**

---

Тестирование ПО через интерфейс  
(без доступа к исходному коду)

# МЕТОД «БЕЛОГО ЯЩИКА»

---

## Плюсы

- Упрощение диагностики
- Легко автоматизировать тесты
- Легко собирать данные
- Стимулирует разработчиков
- «Олдовость»

## Минусы

- Высокий порог входа
- Фокус только на имеющемся функционале
- Не учитывается среда выполнения
- Не учитывается непредсказуемость пользователей



# МЕТОД «ЧЕРНОГО ЯЩИКА»

---

## Плюсы

- Низкий порог входа
- Учитывается среда выполнения
- Учитывается непредсказуемость пользователей
- Раннее тестирование
- Тестирование требований
- Многократность использования

## Минусы

- Повторяемость
- Вероятность неполного покрытия
- Необходимость качественной документации
- Повышенная сложность
- Трудности с планированием
- Потенциальная дороговизна

# МЕТОД «СЕРОГО ЯЩИКА»

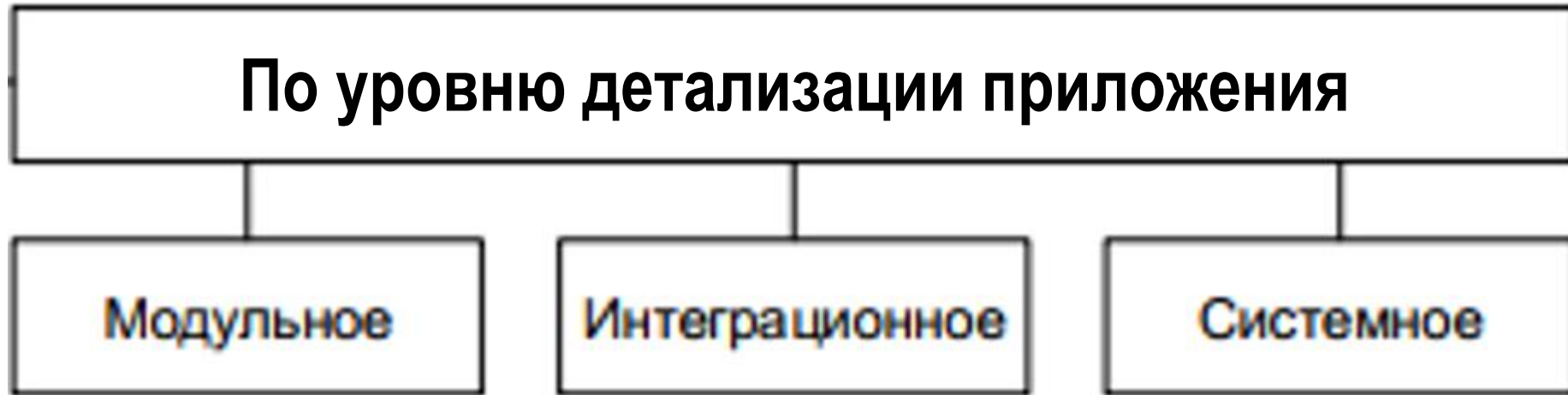
---

## Плюсы

- Комбинация методов Белого и Черного ящиков
- Возможность усложнения тестов
- Дает много времени на дебаг
- Легко убрать ненужные тесты

## Минусы

- Ограничен анализ кода
- Неполное покрытие тестами
- Ситуации ненужности тестировщиков



- ✓ **Модульное** (компонентное) тестирование — проверяются отдельные небольшие части приложения
- ✓ **Интеграционное** тестирование — проверяется взаимодействие между несколькими частями приложения.
- ✓ **Системное** тестирование — приложение проверяется как единое целое

Скидки

Рейтинг  
товаров

Оплата  
WebMoney

Отзывы о  
товарах

**КОМПОНЕНТНОЕ  
ТЕСТИРОВАНИЕ**

Рейтинг товаров +  
Каталог товаров

Скидки + Оплата

Рейтинг товаров +  
Поиск товаров


Оплата WebMoney +  
Профайл покупателя

**ИНТЕГРАЦИОННОЕ  
ТЕСТИРОВАНИЕ**

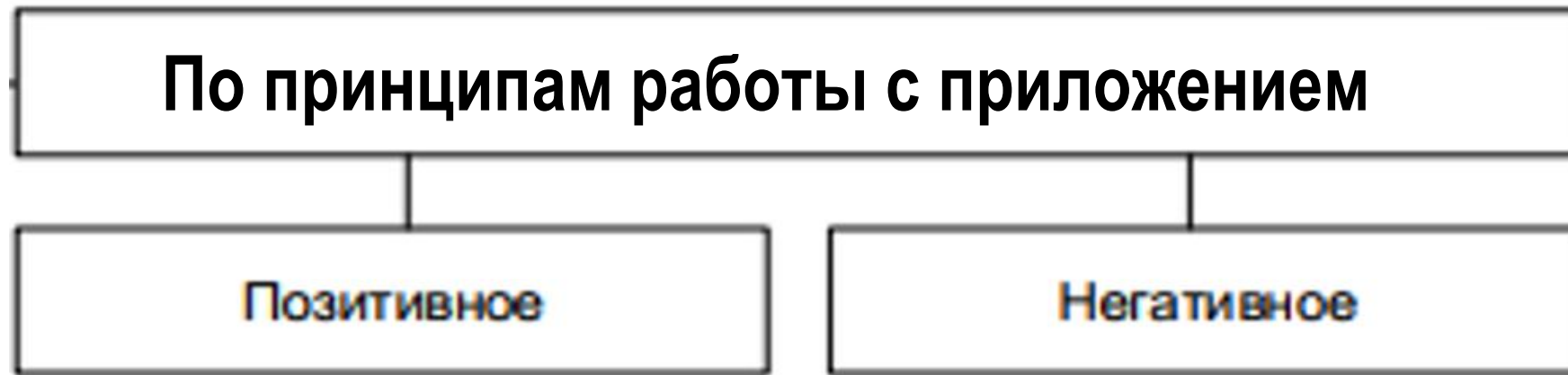
Каталог товаров + Поиск товаров + Оплата +  
Профайл покупателя + Скидки + Рейтинг товаров +  
Оплата WebMoney + Отзывы о товарах

**СИСТЕМНОЕ  
ТЕСТИРОВАНИЕ**



Это странное название происходит из истории XX века, а именно: крайне несовершенных электрических приборов того времени. Если при включении электроприбора был  виден дым (или был запах горелой проводки) – прибор *нерабочий*.

- ✓ **Дымовое тестирование** (smoke test) — проверка самой важной, самой ключевой функциональности, неработоспособность которой делает бессмысленной саму идею использования приложения.
- ✓ **Тестирование критического пути** (critical path test) — проверка функциональности, используемой типичными пользователями в типичной повседневной деятельности.
- ✓ **Расширенное тестирование** (extended test) — проверка всей (остальной) функциональности, заявленной в требованиях.



- ✓ **Позитивное тестирование** — все действия с приложением выполняются строго по инструкции без никаких недопустимых действий, некорректных данных и т.д.
- ✓ **Негативное тестирование** — в работе с приложением выполняются (некорректные) операции и используются данные, потенциально приводящие к ошибкам.

Бытует неверное понимание того, что негативные тест-кейсы должны заканчиваться возникновением сбоев и отказов в приложении. Нет, это не так. Ожидаемым результатом негативных тест-кейсов является именно корректное поведение приложения, а сами **негативные тест-кейсы** считаются пройденными успешно, если им не удалось «поломать» приложение.



Рассмотрели упрощенную классификацию, схема и описание более подробной классификации приведена в книге «Тестирование программного обеспечения» Святослава Куликова на стр.72

Но есть и другие авторы и источники, например:

<https://habr.com/ru/articles/770600/>

<https://testengineer.ru/vidy-tipy-testirovaniya/>

