

Лекция 2.

**Тестирование в контексте
жизненного цикла ПО**

**Жизненный цикл ПО
(Software Life Cycle)**

и

**жизненный цикл разработки ПО
(Software Development Life Cycle)**

Жизненный цикл ПО

Это **более широкое понятие**, охватывающее **всю историю существования ПО** — от появления идеи до полного вывода из эксплуатации. Он включает:

- Формирование бизнес-идеи или потребности
- Планирование и анализ требований
- Проектирование, разработку, тестирование
- Внедрение и эксплуатацию
- Поддержку, модернизацию, обновления
- Окончательное прекращение использования (депрекация)

Жизненный цикл разработки ПО

Это **часть** общего жизненного цикла ПО, сфокусированная **исключительно на процессе создания продукта**. SDLC обычно охватывает этапы:

1. Анализ требований
2. Проектирование
3. Реализация (кодирование)
4. Тестирование
5. Развертывание

Этапы жизненного цикла разработки ПО

1. Планирование и анализ требований

На этом этапе определяются цели проекта, выявляются потребности заказчика или пользователей, проводится технико-экономическое обоснование. Результатом обычно становится документ с требованиями к системе (Software Requirements Specification, SRS).

2. Проектирование

На основе требований архитекторы и разработчики проектируют архитектуру системы: выбирают технологии, определяют структуру модулей, интерфейсы, базы данных и т.д. Часто создаются диаграммы, технические спецификации и прототипы.

Этапы жизненного цикла разработки ПО

3. Реализация (кодирование)

Разработчики пишут исходный код в соответствии с проектной документацией. На этом этапе также проводится внутреннее (модульное) тестирование и применяются практики контроля качества кода (code review, статический анализ и др.).

4. Тестирование

Проводится проверка соответствия ПО заявленным требованиям. Включает различные виды тестирования: функциональное, интеграционное, нагрузочное, регрессионное и т.п. Обнаруженные дефекты исправляются.

Этапы жизненного цикла разработки ПО

5. Развертывание (внедрение)

Готовое ПО устанавливается в целевой среде. Может включать миграцию данных, обучение пользователей, настройку инфраструктуры и поэтапный запуск (например, через canary-релиз или blue-green deployment).

6. Сопровождение и поддержка

После запуска ПО продолжает развиваться: исправляются баги, добавляются новые функции, обеспечивается совместимость с новым окружением. Этот этап может длиться годами.

7. Вывод из эксплуатации

Когда система устаревает или заменяется новой, её эксплуатация прекращается. Данные могут быть архивированы, а функциональность передана другим системам.

Модели жизненного цикла разработки

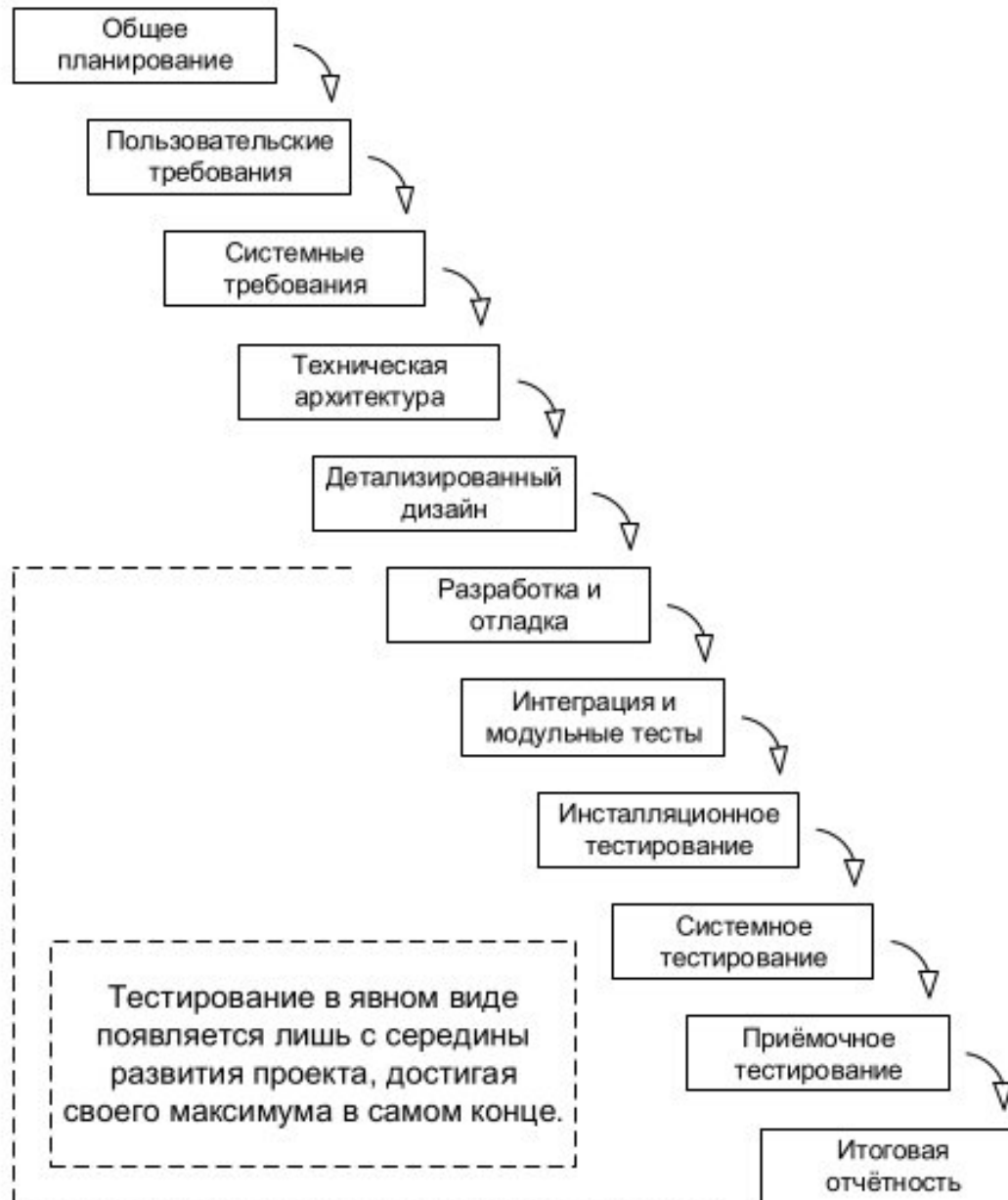
- **Каскадная (водопадная)** — линейная последовательность этапов.
- **Итеративная** — повторяющиеся циклы разработки с постепенным уточнением требований.
- **Спиральная** — сочетает итерации и оценку рисков.
- **Гибкие (Agile, Scrum, Kanban)** — ориентированы на частые релизы, гибкость и взаимодействие с заказчиком.

Выбор модели зависит от масштаба проекта, стабильности требований, команды и других факторов.



Место и роль тестирования в жизненном цикле ПО

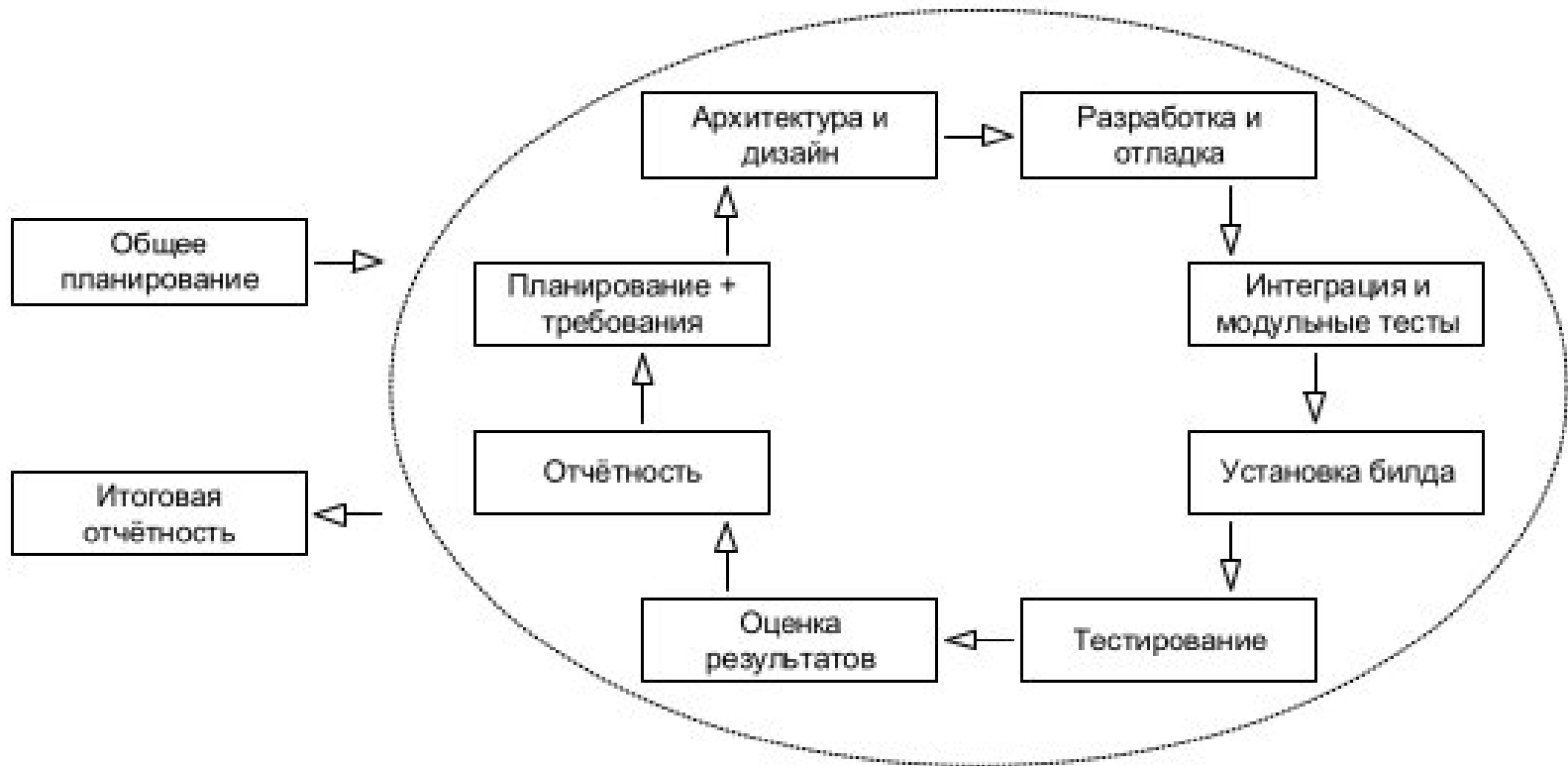
Водопадная (каскадная) модель



Водопадная (каскадная) модель

- **Характеристика:** линейная, последовательная модель. Каждый этап завершается полностью до начала следующего.
- **Место тестирования:**
Тестирование — отдельный, завершающий этап **после реализации (кодирования)**.
- **Роль:**
 - Основная цель — верификация и валидация готового продукта.
 - Тесты пишутся **после** завершения кода на основе спецификаций.
 - Обратная связь медленная: ошибки, найденные на этапе тестирования, могут быть очень дорогостоящими для исправления.
- **Особенности:**
 - Подходит для проектов с **чётко зафиксированными и стабильными требованиями**.
 - Тестирование часто проводится отдельной QA-командой.

Итеративная модель

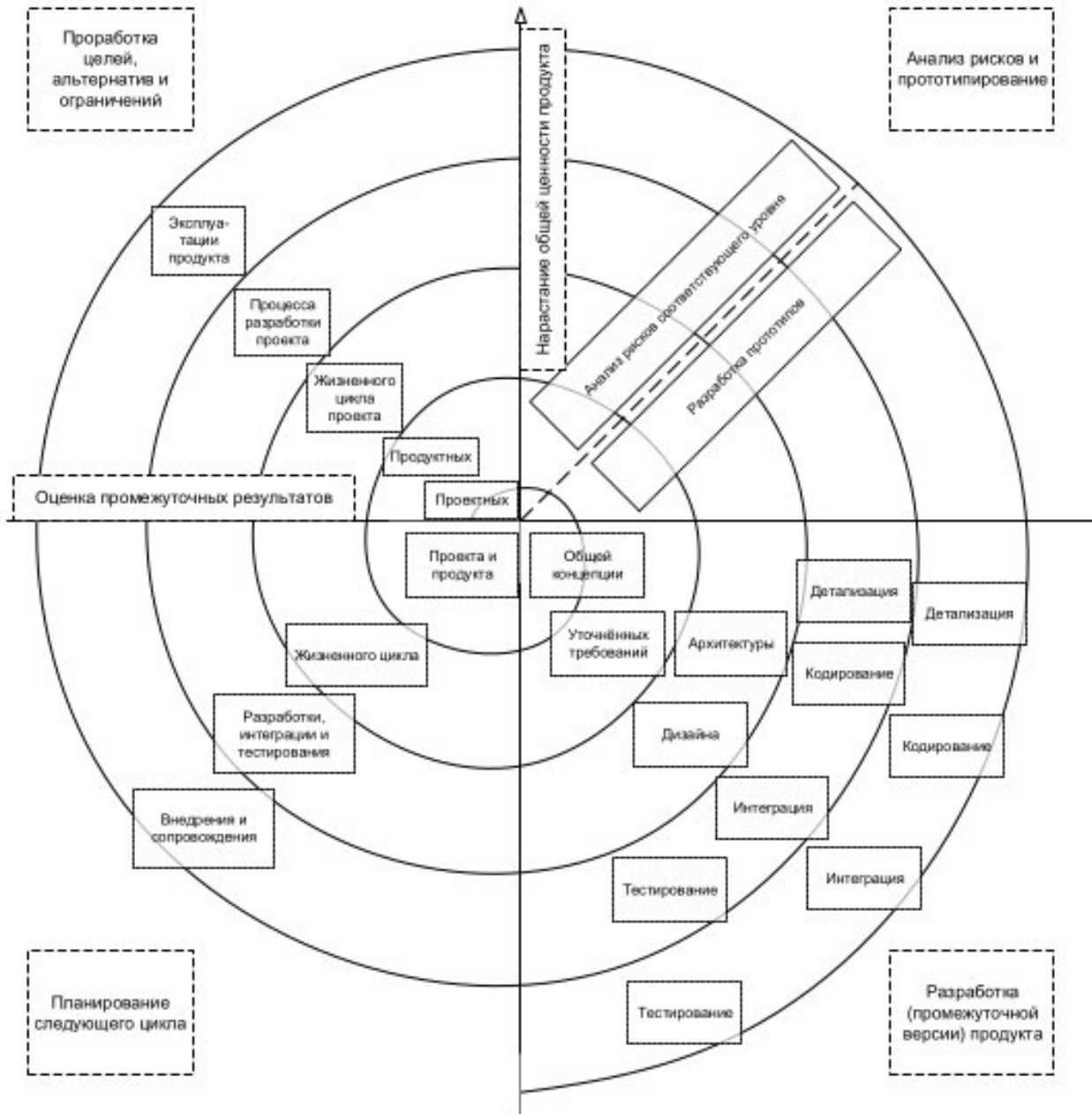


- **Характеристика:** разработка идёт циклами, каждый из которых добавляет функциональность.
- **Место тестирования:** Тестирование интегрировано в **каждую итерацию**, но может быть менее непрерывным, чем в Agile.

Итеративная модель

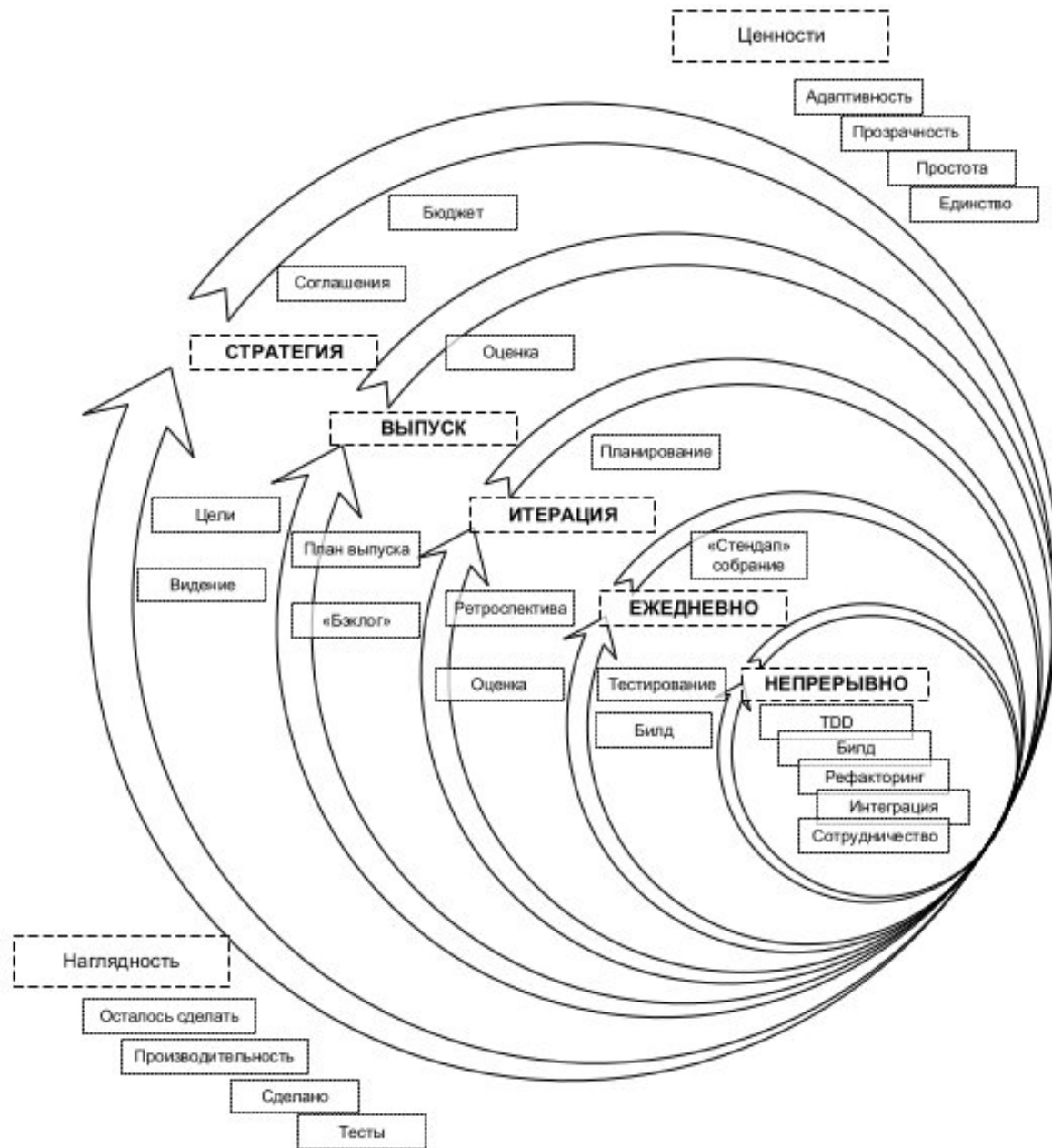
- **Характеристика:** разработка идёт циклами, каждый из которых добавляет функциональность.
- **Место тестирования:**
Тестирование интегрировано в **каждую итерацию**, но может быть менее непрерывным, чем в Agile.
- **Роль:**
 - Позволяет регулярно проверять работоспособность новых модулей.
 - Часто используется **интеграционное и системное тестирование** по мере наращивания функционала.
- **Особенности:**
 - Баланс между структурированностью и гибкостью.

Спиральная модель



- **Характеристика:** сочетает итерации с акцентом на управление рисками.
- **Место тестирования:**
Тестирование проводится **на каждой итерации (витке спирали)**.
- **Роль:**
 - Тестирование используется для **оценки рисков** и проверки жизнеспособности прототипов.
 - На ранних витках — фокус на архитектурных и критических рисках.
- **Особенности:**
 - Позволяет выявлять дефекты и риски **рано**, но требует высокой экспертизы.

Гибкие модели (agile, scrum, kanban)



Гибкие модели (Agile, Scrum, Kanban)

- **Характеристика:** итеративная и инкрементальная разработка, с акцентом на **непрерывную доставку ценности**.
- **Место тестирования:**
Тестирование — **неотъемлемая часть каждой итерации (спринта)**.
Часто практикуется **непрерывное тестирование (Continuous Testing)**.
- **Роль:**
 - Тестирование начинается **параллельно с разработкой** или даже раньше (в рамках BDD/TDD: Test-Driven Development / Behavior-Driven Development).
 - QA-инженеры работают **в одной команде с разработчиками**.
 - Цель — **быстрая обратная связь**, автоматизация регрессионных тестов, предотвращение дефектов.
- **Особенности:**
 - Высокий уровень **автоматизации тестов**.
 - "Сдвиг влево" (Shift-Left Testing): тестирование смещается **на ранние стадии** жизненного цикла.
 - Качество — **ответственность всей команды**, а не только QA.

- **Характеристика:** фокус на **непрерывной интеграции и доставке (CI/CD)**.
- **Место тестирования:**
Тесты запускаются **автоматически при каждом коммите** в pipeline.
- **Роль:**
 - Тестирование — **часть конвейера доставки ПО**.
 - Используются многоуровневые тесты: unit → integration → E2E → performance → security.
 - Быстрая откатываемость при падении тестов.
- **Особенности:**
 - Требуется **высокая степень автоматизации и надёжности тестов**.
 - Тестирование становится **инфраструктурной задачей**, интегрированной в DevOps-практики.

Тестирование в разных моделях

Методология	Когда тестируют?	Кто тестирует?	Насколько рано выявляются дефекты?	Автоматизация
Водопадная	После разработки	Отдельная QA-команда	Поздно, дорого исправлять	Низкая
Итеративная	В конце каждой итерации	QA + разработчики	Средне-рано	Средняя
Спиральная	На каждой итерации	Команда + аналитики рисков	Рано (в рамках итерации)	Средняя
Agile/Scrum	Непрерывно в спринте	Вся команда	Очень рано	Высокая
DevOps	Автоматически при каждом изменении	Dev/QA/Infra	Максимально рано	Очень высокая

Тестирование в разных моделях

Вывод

- В **традиционных моделях** (водопад, спираль) тестирование — **отдельная фаза или этап**, часто завершающий.
- В **современных гибких и DevOps-подходах** тестирование — **непрерывный, встроенный процесс**, начинающийся с проектирования и сопровождающий ПО на протяжении всего жизненного цикла.
- Современные практики стремятся к **раннему и постоянному тестированию**, чтобы **снизить стоимость исправления ошибок** и **повысить надёжность** поставляемого ПО.

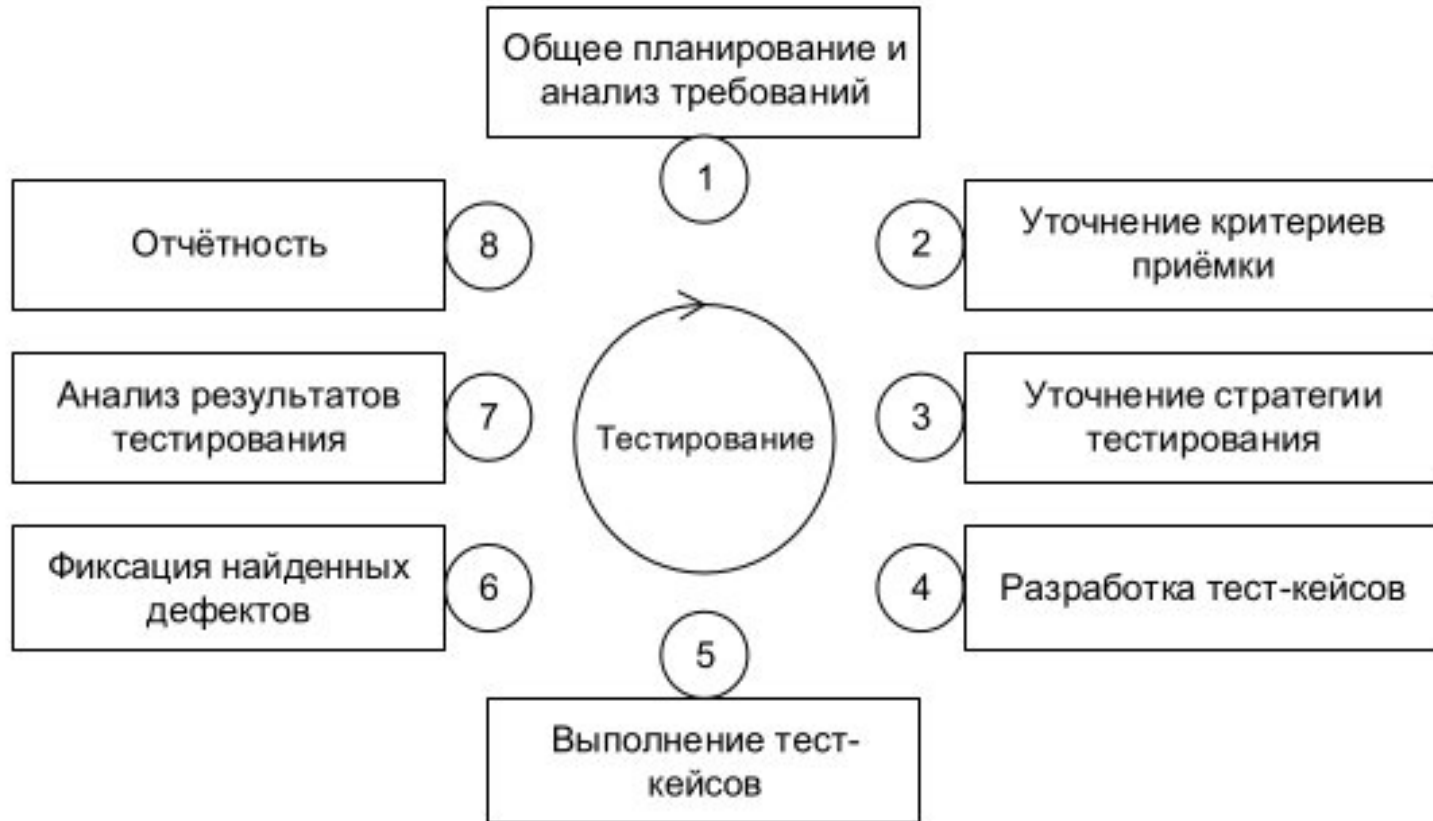
Если кратко:

В старых моделях тестируют, чтобы найти баги.

В новых — тестируют, чтобы их не допустить.

Жизненный цикл тестирования

Жизненный цикл тестирования



Жизненный цикл тестирования

- ♦ **Этап 1: Общее планирование и анализ требований**
- **Цель:** определить, **что нужно тестировать**, на основе бизнес-требований, технических спецификаций, пользовательских сценариев.
- **Что делается:**
 - Анализ функциональных и нефункциональных требований.
 - Определение целей тестирования.
 - Формирование предварительного плана тестирования.
- **Важно:** это **начальный этап**, от которого зависит вся дальнейшая работа. Без чёткого понимания требований невозможно спроектировать эффективные тесты.

Жизненный цикл тестирования

- **Этап 2: Уточнение критериев приемки**
- **Цель:** определить, **какие условия должны быть выполнены, чтобы продукт считался принятым** заказчиком или бизнесом.
- **Что делается:**
 - Формулировка критериев «готово» (acceptance criteria) для каждой функции.
 - Согласование с заинтересованными сторонами (стейкхолдерами).
- **Пример:** «Пользователь может войти в систему, если введён корректный email и пароль».

- ♦ **Этап 3: Уточнение стратегии тестирования**
- **Цель:** разработать **общую стратегию** — как будет проводиться тестирование.
- **Что делается:**
 - Выбор типов тестирования (функциональное, регрессионное, нагрузочное и т.д.).
 - Определение уровня автоматизации.
 - Распределение ресурсов, сроков, инструментов.
 - Планирование тестовой среды.
- **Результат:** документ **Test Strategy** — руководство для всей команды.

- **Этап 4: Разработка тест-кейсов**
- **Цель:** создать конкретные сценарии тестирования, которые позволят проверить соответствие ПО требованиям.
- **Что делается:**
 - Написание шагов теста, ожидаемых результатов.
 - Создание данных для тестов.
 - Классификация тест-кейсов по приоритетам и модулям.
- **Формат:** часто используется шаблон: ID, описание, предусловия, шаги, ожидаемый результат, фактический результат, статус.

Жизненный цикл тестирования

♦ **Этап 5: Выполнение тест-кейсов**

- **Цель:** провести тестирование в соответствии с подготовленными тест-кейсами.
- **Что делается:**
 - Запуск тестов (ручных или автоматизированных).
 - Фиксация результатов: пройден / не пройден.
 - Ведение журналов тестирования.
- **Важно:** этот этап — **центральный** в цикле, но он не существует сам по себе — он опирается на всё, что было сделано ранее.

- **Этап 6: Фиксация найденных дефектов**
- **Цель:** зафиксировать и задокументировать все обнаруженные ошибки.
- **Что делается:**
 - Создание дефектных отчётов (bug reports) с описанием, шагами воспроизведения, скриншотами/логами, приоритетом.
 - Передача дефектов разработчикам.
 - Отслеживание статуса исправления.
- **Инструменты:** Jira, Bugzilla, Trello, TestRail и др.

- ◆ **Этап 7: Анализ результатов тестирования**
- **Цель:** оценить качество ПО и эффективность самого процесса тестирования.
- **Что делается:**
 - Подсчёт количества найденных и закрытых дефектов.
 - Оценка покрытия требований тестами.
 - Анализ метрик: процент прохождения тестов, время на исправление багов, количество регрессий.
 - Формирование выводов: готов ли продукт к релизу?
- **Результат:** отчёт о качестве, рекомендации по улучшению.

- **Этап 8: Отчётность**

- **Цель:** информировать всех участников проекта о ходе и результатах тестирования.
- **Что делается:**
 - Подготовка итоговых отчётов для менеджеров, разработчиков, заказчиков.
 - Представление метрик, графиков, диаграмм.
 - Обсуждение рисков и рекомендаций.
- **Важно:** отчётность — ключевой элемент для принятия решений о релизе или необходимости доработок.

Жизненный цикл тестирования

Цикличность

Заметьте, что после этапа 8 (**Отчётность**) стрелка указывает снова на этап 1 — **Общее планирование и анализ требований**. Это означает, что:

Тестирование — не одноразовый процесс, а циклический и итеративный. После анализа результатов и отчётности могут быть выявлены новые требования, уточнения, изменения — и цикл начинается заново.

Это особенно актуально в Agile и DevOps, где тестирование повторяется в каждом спринте или при каждом релизе.

Жизненный цикл тестирования

Вывод

Данный график демонстрирует **полноценный жизненный цикл тестирования**, который включает:

- **Планирование и подготовку** (этапы 1–4),
- **Исполнение** (этап 5),
- **Обратную связь и анализ** (этапы 6–8).

Он подчёркивает, что качественное тестирование — это **системный процесс**, требующий внимания на всех этапах, а не просто запуск тестов перед релизом. Такой подход позволяет:

- ✓ Раннее выявление дефектов
- ✓ Повышение качества продукта
- ✓ Улучшение коммуникации между командами
- ✓ Обеспечение прозрачности и контроля над процессом