

# Тестирование программного обеспечения

Алексей Островский

Физико-технический учебно-научный центр НАН Украины

февраль 2015 г.

# Основные понятия тестирования

**Этапы возникновения сбоев** в программе:

- 1. программист совершает *ошибку* (error, mistake);
- 2. ошибка приводит к *дефекту* (defect, fault, bug) в исходном коде;
- 3. при определенных условиях исполнения дефект приводит к *сбою программы* (program failure).

**Тест** — набор входных данных и прочих условий (напр., характеристики операционной системы и оборудования), которые полностью определяют ход выполнения программы.

**Цель тестирования** — локализация и устранение дефектов, соответствующие всем сбоям программы, обнаруженным с помощью тестов.

Прогон *всех* тестов невозможен  
даже для простых систем.

⇒

Необходим отбор  
*информативных* тестов.

# Тестирование ПО

## Определение

**Тестирование ПО** — это процесс проверки готовой программы в *статике* (обзоры кода, инспекции и т. п.) и *динамике* (прогон программы на тестовых данных) с целью обеспечить ее соответствие заданным требованиям.

### Режимы тестирования:

	Валидация	Обнаружение дефектов
Цели	демонстрация разработчикам и заказчикам соответствия программного продукта требованиям	определение границ применимости ПО; поиск ситуаций, в которых поведение некорректно, нежелательно или не удовлетворяет спецификации.
Содержание	ожидаемые сценарии использования ПО	экстремальные сценарии использования.
Выбор тестов	исходя из спецификации системных требований	для отработки исключительных ситуаций.

# Виды тестирования

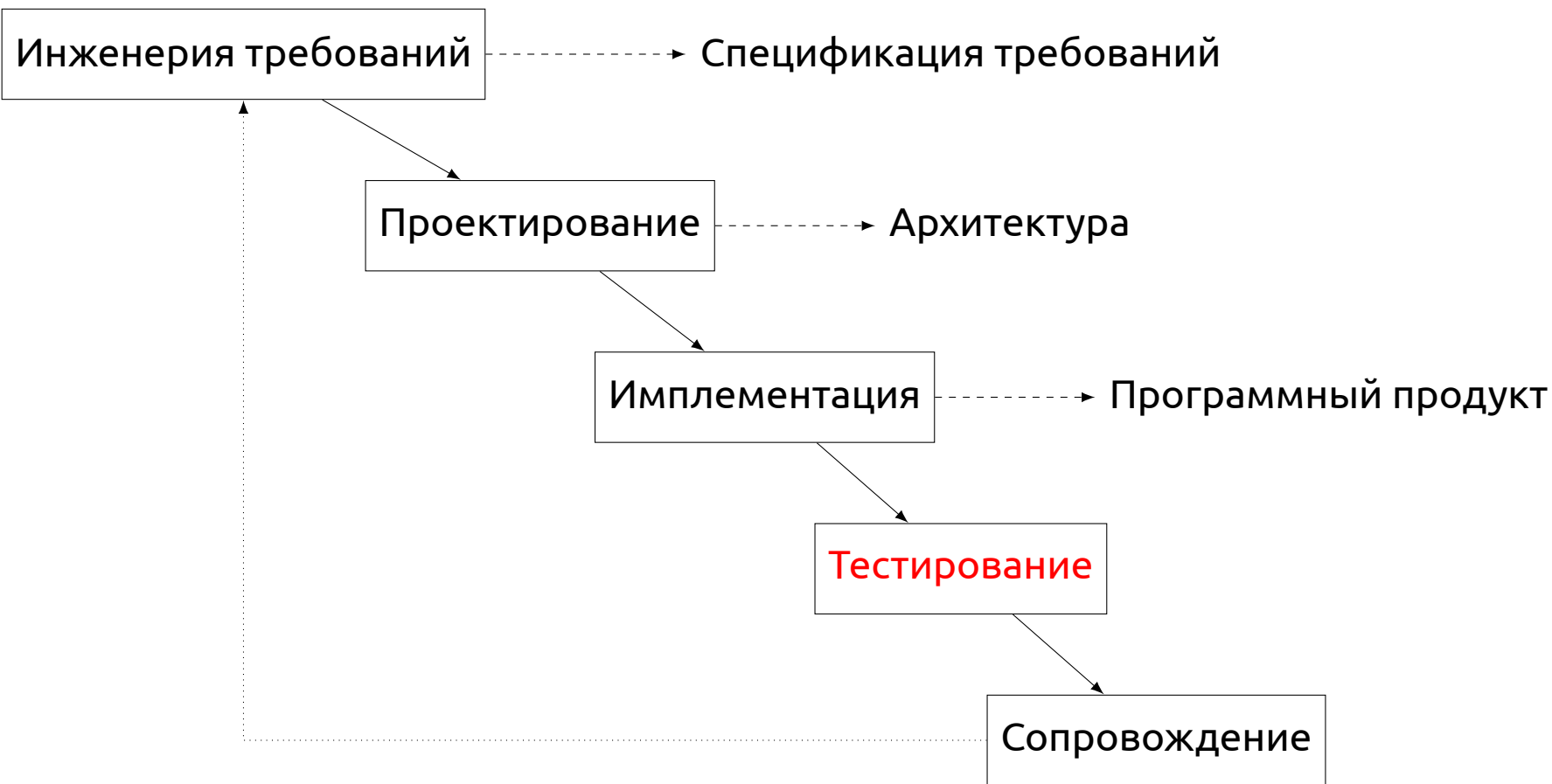
## По объекту тестирования:

- ▶ статические методы (процессы верификации и валидации — следующая лекция);
- ▶ динамические методы.

## С точки зрения при составлении тестов:

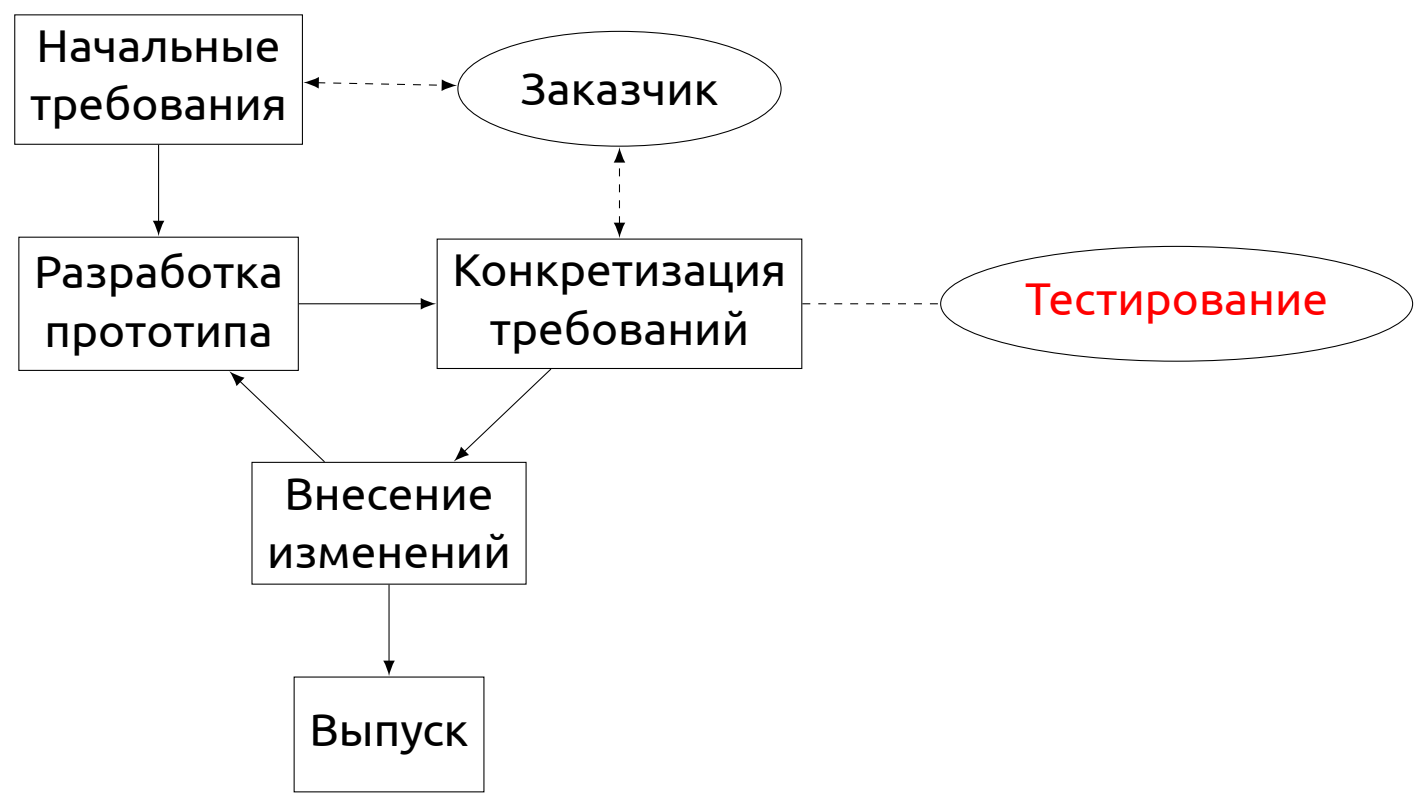
- ▶ **Белый ящик** (англ. *white box testing*), структурное тестирование — тестирование внутренних структур и операций ПО.
- ▶ **Черный ящик** (англ. *black box testing*) — тестирование функциональности, доступной конечному пользователю ПО.
- ▶ **Серый ящик** (англ. *gray box testing*) — тестирование ПО с частичным знанием о его внутренней структуре.

# Место тестирования в разработке ПО



В каскадной модели тестирование — отдельный этап разработки, следующий за написанием кода.

# Место тестирования в разработке ПО



Согласно эволюционной модели разработки ПО, тестирование неразрывно связано с конструированием.

# Процесс тестирования

1. Разработка тестовых примеров (англ. *test cases*).

**Результаты:** набор тестовых примеров.

2. Подготовка тестовых данных.

**Результаты:** данные для тестовых примеров.

3. Выполнение программы на тестовых данных.

**Результаты:** продукты выполнения программы.

4. Сравнение результатов с ожидаемыми.

**Результаты:** отчеты о тестировании.

Выполнение тестов и подготовка отчетов может быть *автоматизированным*.

# Фазы тестирования

№	Название	Цель	Ответственные
1	Тестирование при разработке ( <i>development testing</i> )	обнаружение дефектов	разработчики
2	Тестирование выпусков ( <i>release testing</i> )	проверка на соответствие требованиям заказчика	отдел тестирования
3	Пользовательское тестирование ( <i>user testing</i> )	выполнение в среде пользователя; уточнение требований.	пользователи; отдел маркетинга



# Тестирование при разработке

## Определение

**Тестирование при разработке** (англ. *development testing*) — совокупность всех видов тестирования, осуществляемых непосредственными разработчиками программного продукта.

### Уровни тестирования:

1. модульное тестирование (англ. *unit testing*) — проверка функциональности отдельных методов или классов;
2. компонентное тестирование (англ. *component testing*) — проверка программных интерфейсов составных компонентов;
3. системное тестирование (англ. *system testing*) — проверка системы в целом; тестирование взаимодействий между компонентами.

# Модульное тестирование

## Цель:

- ▶ проверка методов класса, реализующих отдельные системные требования, с различными входными данными;
- ▶ тестирование состояний, в которых может находиться объект.

## Этапы модульного теста

1. **подготовка:** инициализация системы, входных и ожидаемых выходных данных;
2. **вызов** тестируемых методов или объектов;
3. **сравнение** результатов вызова с ожидаемым.

## Инструменты:

- ▶ автоматизированные системы тестирования (xUnit);
- ▶ мок-объекты (упрощенные объекты, реализующие требуемые интерфейсы, напр., БД).

# Подбор модульных тестов

- ▶ Анализ граничных значений (англ. *boundary-value analysis*).

...	-2	-1	0	1	...	99	100	101	102	...
некорректно			корректно				некорректно			

Граничные значения для целочисленной величины, обозначающей процент.

- ▶ Комбинаторные методы: уникальные пары значений, ортогональные массивы.
- ▶ Выбор на основе математической модели тестируемой системы.
- ▶ Эвристический подход — тестирование тех наборов данных, где легче всего допустить ошибку.

# Пример — тестирование чисел Фибоначчи (Java / JUnit)

```
1  @Test
2  public void testBoundaryValues() {
3      String str = Fibonacci.get(0);
4      assertEquals("0", str);
5      str = Fibonacci.get(1);
6      assertEquals("1", str);
7  }
8
9  @Test
10 public void testSums() {
11     for (int n = 10; n < 10000; n++) {
12         BigInteger a = new BigInteger(Fibonacci.get(n)),
13             b = new BigInteger(Fibonacci.get(n - 1)),
14             c = new BigInteger(Fibonacci.get(n - 2));
15         assertEquals(b.plus(c), a);
16     }
17 }
```

# Тестирование компонентов

Тестирование компонентов  $\simeq$  проверка корректности их интерфейсов.

## Виды ошибок, связанных с интерфейсами:

- ▶ Неправильное использование (англ. *interface misuse*).

**Пример:** параметры некорректного типа или в неправильном порядке при вызове методов.

- ▶ Заблуждения об интерфейсе (англ. *interface misunderstanding*).

**Пример:** бинарный поиск в несортированном массиве.

- ▶ Ошибки синхронизации (англ. *timing errors*).

**Пример:** работа с разделяемой памятью или очередями сообщений с различной скоростью обработки данных разными компонентами.

# Подбор тестов компонентов

- ▶ Проверка граничных значений параметров методов при работе с внешними компонентами;
- ▶ корректность работы с нулевыми указателями (**null**) при передаче объектов;
- ▶ возникновение и обработка исключительных ситуаций для проверки правильного понимания спецификаций;
- ▶ стрессовое тестирование для систем обмена сообщениями;
- ▶ варьирование очередности вызовов при использовании разделяемой памяти.

**Инструменты:** статические анализаторы кода.

# Системное тестирование

## Определение

### Системное тестирование —

- ▶ проверка корректности взаимодействия между компонентами программной системы;
- ▶ проверка отсутствия нежелательных побочных эффектов, связанных с выполнением в единой среде.

### Подбор тестов:

- ▶ покрытие кода (каждая инструкция должна выполняться  $\geq 1$  раз);
- ▶ тестирование пользовательского ввода с корректными и некорректными данными;
- ▶ тестирование методов взаимодействия с системой;
- ▶ тестирование ожидаемых комбинаций методов.

# Разработка через тестирование

## Определение

**Разработка через тестирование** (англ. *test-driven development, TDD*) — модель разработки ПО, основанная на использовании автоматизированных тестовых вариантов для определения новой или усовершенствованной функциональности.

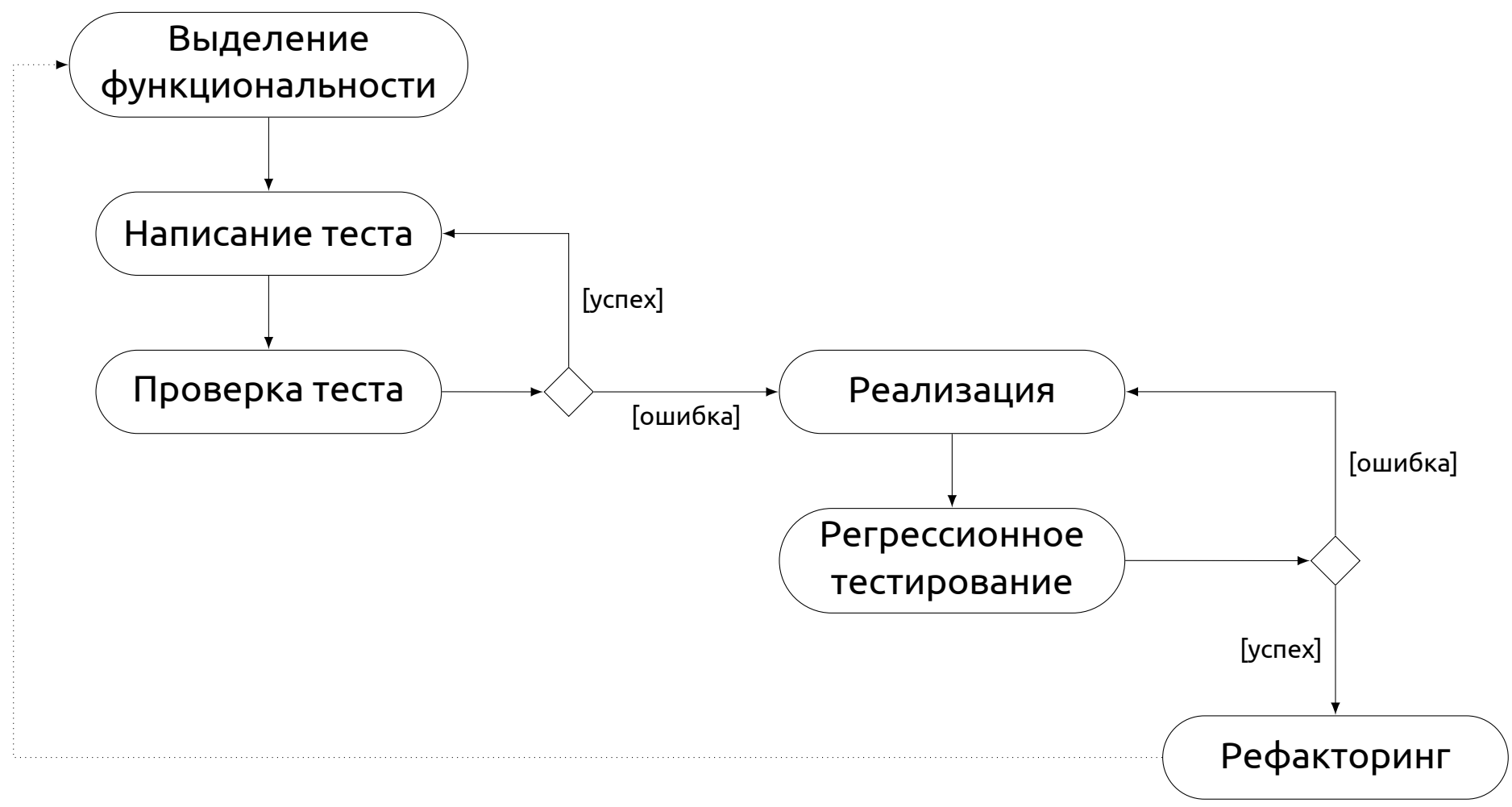
### Область применения TDD:

- ▶ гибкая методология разработки;
- ▶ экстремальное программирование.

**Инструменты:** автоматизированные системы тестирования (напр., xUnit).



# Процесс разработки в TDD



# Преимущества TDD

- ▶ Скрупулезное тестирование всего выполняемого кода; обнаружение дефектов на ранних стадиях разработки.
- ▶ Автоматизация регрессионного тестирования.
- ▶ Упрощение процесса отладки и локализации ошибок.
- ▶ Повышение модульности и гибкости кода; четкое определение интерфейсов.
- ▶ Тесты как форма документации программной системы.

# Недостатки TDD

- ▶ Невозможность применения для тестирования слабо формализованных (пользовательский интерфейс) или комплексных (работа с БД) требований.
- ▶ Ангажированность при написании тестов: тесты могут не покрывать возможные сценарии использования из-за неверной трактовки разработчиком требований к системе.
- ▶ Необходимость тщательного планирования разработки (тестам должно уделяться адекватное количество времени).
- ▶ Проблема тестирования скрытых (**private**) полей / методов.

# Тестирование выпусков

## Определение

**Тестирование выпусков** (англ. *release testing*) — проверка определенного выпуска программного продукта, предназначенного для использования вне отдела разработки.

**Отличия от системного тестирования на этапе разработки:**

	Системное тестирование	Тестирование выпуска
Ответственные за тестирование	разработчики	независимый отдел тестирования
Цель тестирования	определение и исправление ошибок	проверка соответствия требованиям
Методы тестирования	«белый ящик»	«черный ящик»

# Виды тестирования выпусков

- ▶ Тестирование отдельных пользовательских требований (1 требование = много тестов).
- ▶ Тестирование сценариев использования ПО ( $\simeq$  тестирование ожидаемых комбинаций требований к различным компонентам системы).
- ▶ Тестирование нефункциональных требований (производительность, отказоустойчивость, надежность, ...)

# Тестирование производительности

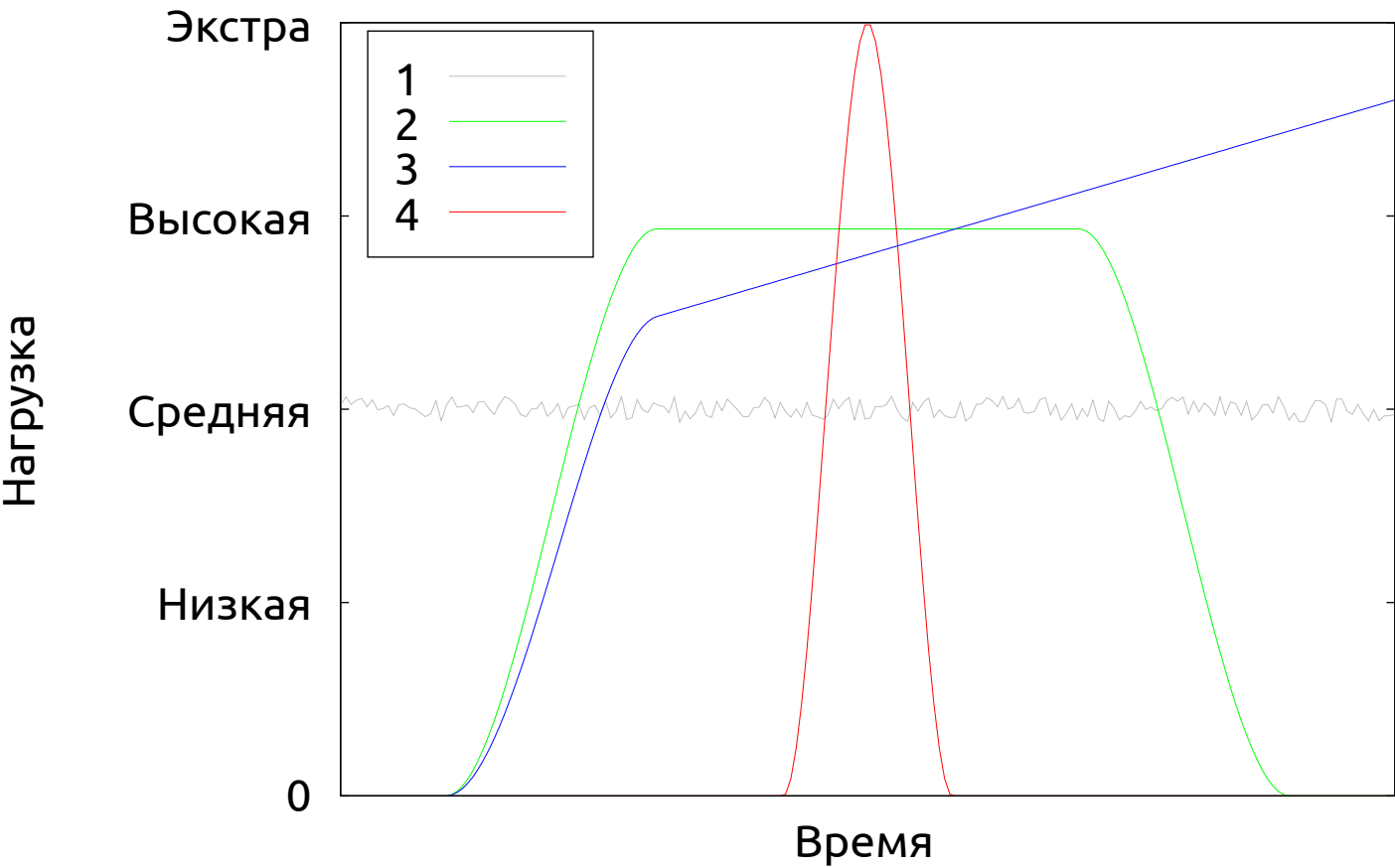
## Измеряемые характеристики:

- ▶ производительность (англ. *performance*);
- ▶ надежность (отказоустойчивость) (англ. *reliability*);
- ▶ масштабируемость (англ. *scalability*).

## Цели:

- ▶ проверка выполнения требований к производительности ПО;
- ▶ определение параметров системы для достижения максимальной производительности;
- ▶ выявление наиболее затратных по времени модулей программной системы;
- ▶ тестирование отказоустойчивости системы при превышении ожидаемой нагрузки.

# Тестирование производительности



Различные режимы тестирования производительности:

- 1

— нагрузочное тестирование (англ. *load testing*);
- 2

— стресс-тестирование (англ. *stress testing*);
- 3

— тестирование выносливости (англ. *soak testing*);
- 4

— импульсное тестирование (англ. *spike testing*).

# Пользовательское тестирование

## Определение

**Пользовательское тестирование** (англ. *user testing*) — оценка выполнения требований к программной системе с точки зрения конечных пользователей или заказчика.

### Виды пользовательского тестирования:

- ▶ Альфа-тестирование — тестирование системы в содействии с командой разработки в контролируемой среде.

**Цели:** разработка реалистичных тестов; конкретизация требований.

- ▶ Бета-тестирование — тестирование промежуточного выпуска ПО, доступного для определенного контингента пользователей.

**Цели:** определение работоспособности в различных условиях; продвижение ПО.

- ▶ Приемочное тестирование (англ. *acceptance testing*) — проверка системы заказчиком на ее готовность.


**Цели:** оплата стоимости разработки; развертывание системы.




# Выводы

1. Существует две основные цели тестирования — обнаружение ошибок и проверка (валидация) функциональности программного продукта.
2. Тестирование включает три фазы — тестирование во время разработки (*development testing*), тестирование выпусков (*release testing*) и пользовательское тестирование (*user testing*).
3. В классических моделях жизненного цикла тестирование следует за конструированием ПО; более современный подход состоит в опережающем написании тестов (TDD).
4. Автоматизированные системы тестирования (напр., xUnit) многократно повышают эффективность тестирования вносимых в ПО изменений.

# Материалы

 **Sommerville, Ian**  
Software Engineering.  
  
Pearson, 2011. — 790 p.

 **Лавріщева К. М.**  
Програмна інженерія (підручник).  
  
К., 2008. — 319 с.

Спасибо за внимание!