

# ТЕХНОПОЛИС |

## Лекция 2.1

# Стратегия поведенческого тестирования



# T | Типы тестирования

---

## Black Box

**Мы не знаем, как устроена тестируемая система.**

Техника тестирования, основанная на работе исключительно с внешними интерфейсами тестируемой системы.

## White Box

**Нам известны все детали реализации тестируемой программы.**

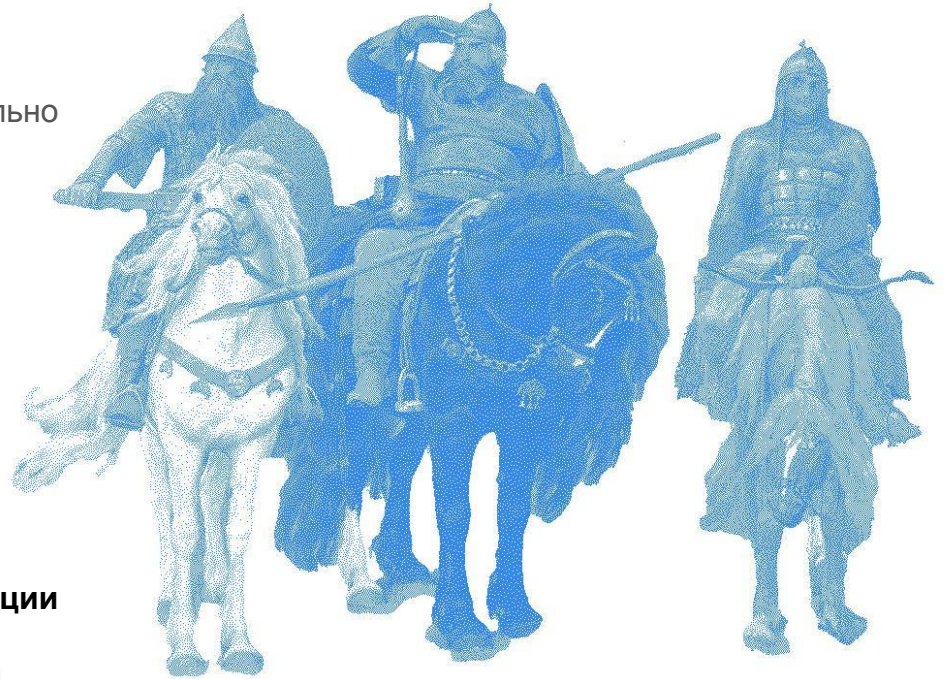
метод тестирования программного обеспечения, который предполагает, что внутренняя

## Grey Box

**Нам известны только некоторые особенности реализации тестируемой системы.**

метод тестирования программного обеспечения, который

предполагает, комбинацию White Box и Black Box подходов.



# **Т** | Зачем вообще нужны стратегии?

---

**Суть тестирования - поиск ошибок в реализации программы**

**Организация тестирования для выявления всех ошибок:**

1. Подготовить все возможные наборы входных данных, включая некорректные.
2. Выполнить программу на всех возможных перестановках и сочетаниях входных данных.
3. Проанализировать все выходные данные и установить, что каждый тестовый выходной набор соответствует правильному.

**Для реализации подобного тестирования потребовалось бы колоссальные временные, финансовые и человеческие ресурсы.**

# **Т** | Зачем вообще нужны стратегии?

---

- Простейшая программа, имеющая три входных параметра – символа латинского алфавита, имеет ~16 млн. комбинаций ввода (с учетом негативных тестов)
- Если тестировщик сможет проверять одну комбинацию в секунду, ему потребуется 190 суток непрерывной работы!
- А если программа работает с оперативной памятью, аппаратными приборами, сетью, базами данных, где результат работы на некотором наборе входных данных зависит от данных, поступивших в программу ранее?
- Исчерпывающее тестирование за разумное время невозможно!

# T | Зачем вообще нужны стратегии?

---

- Невозможно найти все ошибки в программе!
- Необходимо выбирать мизерное подмножество входных данных...
- ... Но так, чтобы найти как можно больше дефектов.
- Выбор должен делаться не случайно, а на основании некоторой стратегии

## Black-box

Поведенческое, функциональное

Неизвестно, как объект (программа) сконструирован внутри. Он как бы представляет собой черный ящик, о котором известна лишь информация о его входах и выходах – функциональные требования к программе. При этом нет никакой информации о том, как именно программа преобразует входные данные в выходные.

## White-box

Структурное

Процедура тестирования строится исходя из знания того, как объект (программа) сконструирован внутри. Объект – это прозрачный ящик, о котором известна не только информация о его входах и выходах, но, прежде всего, известен механизм преобразования входных данных в выходные.

# Т | Зачем вообще нужны стратегии?

## Регистрация в закрытый клуб тестировщиков

Вы тестировщик?

Нет ▾

Сколько вам лет?

Введите email

Хотите получать новостную рассылку?

Зарегистрироваться

Отмена

## Функциональные требования к тестируемой программе (Software Under Test, SUT):



Пользователь регистрируется, если:

1. Совершеннолетний
2. Тестировщик



3. Корректный e-мейл

Получает письмо, если

1. Успешно зарегистрирован в клуб
2. Поставил галочку «хочу получать»

## Как тестировать



# Т | Скриптовой процесс тестирования

---

—  
Тест-аналитик создаёт  
тестовый набор

Тест-дизайнер создаёт тест-  
кейсы и/или чек-листы

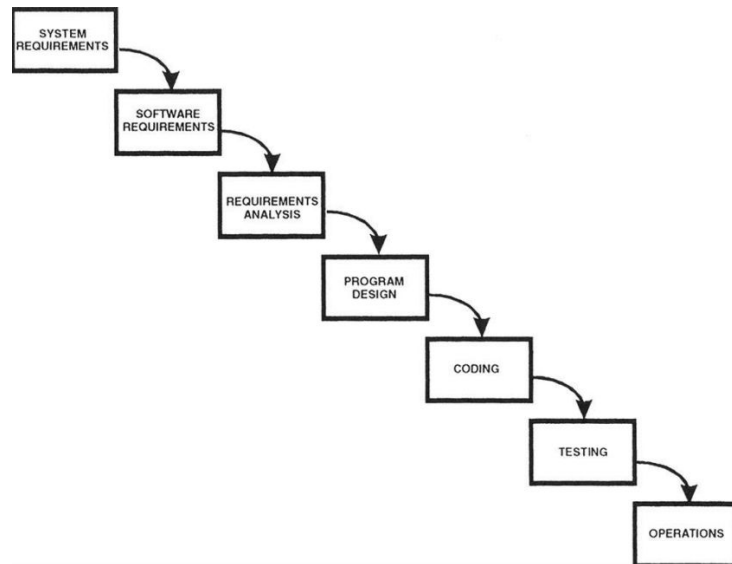
Тестировщик по ним  
тестирование и  
регистрирует найденные  
несоответствия



# T | Скриптовой процесс тестирования

---

Зародилось как компонента водопадного (waterfall) подхода к разработке ПО:



Скриптовое тестирование – пример философии «plan your work, work your plan»



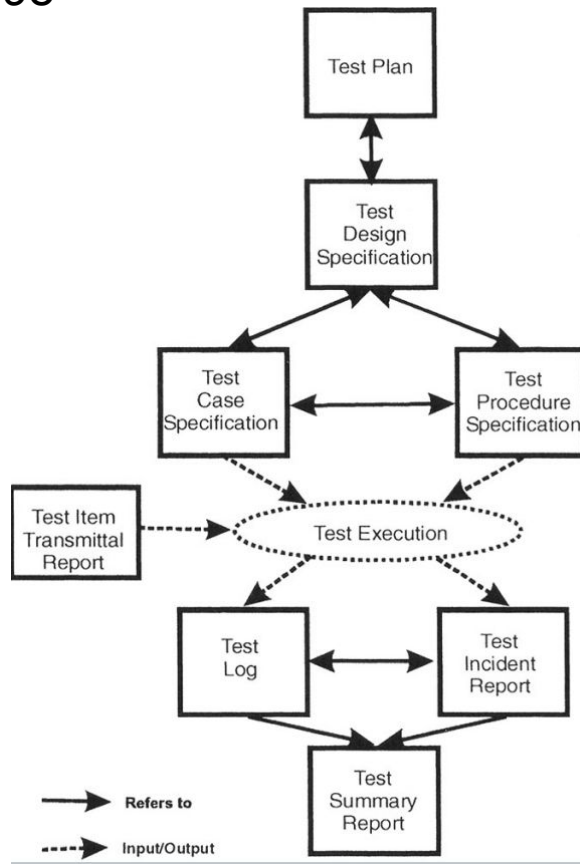
# T | Скриптовой процесс тестирования

Стандарт IEEE Std 829-1998

Артефакты и документы скриптового тестирования описаны в стандарте «IEEE Standard for Software Test Documentation.»

Стандарт выделяет 8 артефактов и документов:

- Test plan
- Test design specification
- Test case specification
- Test procedure specification
- Test item transmittal report (информация об итерации тестирования)
- Test log
- Test incident report
- Test summary report



# Т | Скриптовой процесс тестирования

---

## Достоинства скриптового тестирования

○ Разделение труда – планирование, тест-дизайн, реализация и выполнение тестов могут выполняться:

1. Разными специалистами с необходимыми навыками;
2. В разное время в цикле разработки ПО.

○ Тесты создаются на основе спецификаций, дизайна и кода – все важные части системы будут покрыты тестами.

○ Тесты документированы, они могут быть:

1. Легко поняты и воспроизведены;
2. Выполнены тестировщиками, не имеющими глубокого

знания системы.

○ Тесты детализированы – легче автоматизировать.

○ Тесты создаются на ранних стадиях цикла разработки – освобождается доп. время на этапе выполнения тестов.

# Т | Скриптовой процесс тестирования

---

## Недостатки скриптового тестирования

- Сильно зависит от качества требований к системе: полнота, последовательность, непротиворечивость и т.д.
  
- Не отличается гибкостью. Тесты соответствуют ранее определенным сценариям. Дефекты, не покрытые сценариями, могут быть пропущены.
  
- Дополнительные затраты времени на:
  1. Создание артефактов и документов.
  2. Поддержку в актуальном состоянии при изменении требований, ограничений и т.д.

## **Тестировщик**

Тестирует;

Исследует;

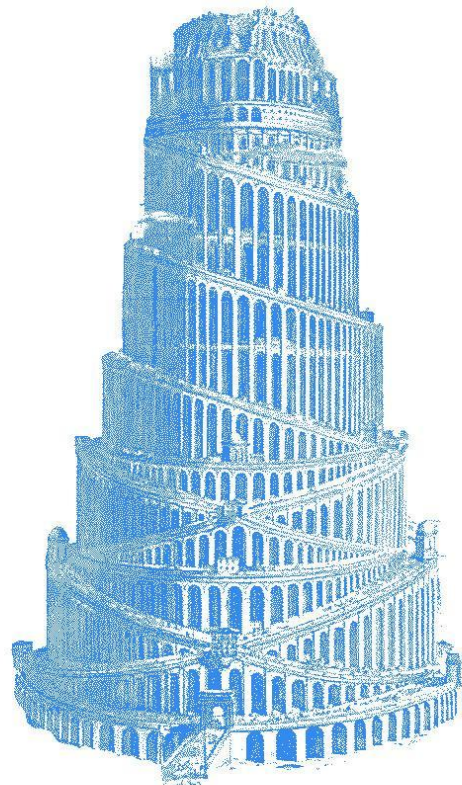
Узнаёт новое;

Генерирует идеи;

Тестирует;

Исследует;

Узнаёт новое...





# Т | Исследовательский процесс тестирования

---

## Достоинство исследовательского тестирования

- Следующий тест не определен заранее, а рождается в процессе в процессе анализа результатов выполнения предыдущих тестов.
- Полезно в условиях, когда необходимо быстро получить результат тестирования, а требования нечеткие или отсутствуют.
- Полезно на ранних этапах разработки, когда система нестабильна.
- Необходимо «прощупать почву» вокруг уже найденного дефекта.
- Полезно в дополнение к скриптовым тестам, когда те уже подвержены «парадоксу пестицида»

# Т | Исследовательский процесс тестирования

## Парадокс пестицида

Boris Beizer, “Software Testing Techniques”

В сельском хозяйстве: После обработки поля часть вредителей погибала. Но оставшиеся приспособились к яду и с высокой вероятностью выживут при последующих обработках

В тестировании: Эффективность неизменяемого набора тестов постепенно уменьшается по мере исправления дефектов, найденных этим набором. Тесты застаиваются



## Недостатки исследовательского тестирования

Не направлено на предотвращение дефектов. В скриптовом тестировании тесты создаются на этапе формирования требований и дизайна ПО, что позволяет находить дефекты раньше.

Требует высокой квалификации тестировщика. Качество зависит от тестировщика.

Нет документированных артефактов – каждая следующая итерация тестирования содержит новую фазу тест-анализа и тест-дизайна.

# Т | Хаотичный процесс тестирования

---

## Тестировщик

Нажимает кнопки

Нажимает кнопки

Нажимает кнопки

Нажимает кнопки

Нажимает кнопки

Нажимает кнопки

Нажимает кнопки

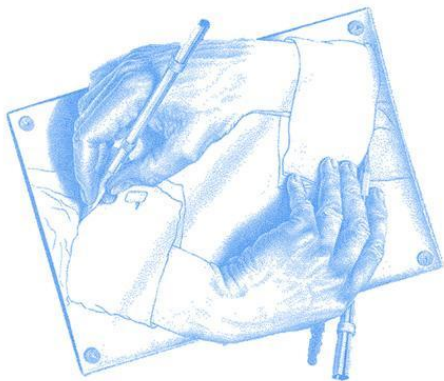
Нажимает кнопки

Нажимает кнопки

Нажимает кнопки







Определение того, ЧТО будет тестироваться  
(=тестовый анализ, создаём тестовый набор)

Определение того, КАК это будет тестироваться  
(=тест-дизайн)

# T | Проектирование тестов

## Регистрация в закрытый клуб тестировщиков

Вы тестировщик?

Сколько вам лет?

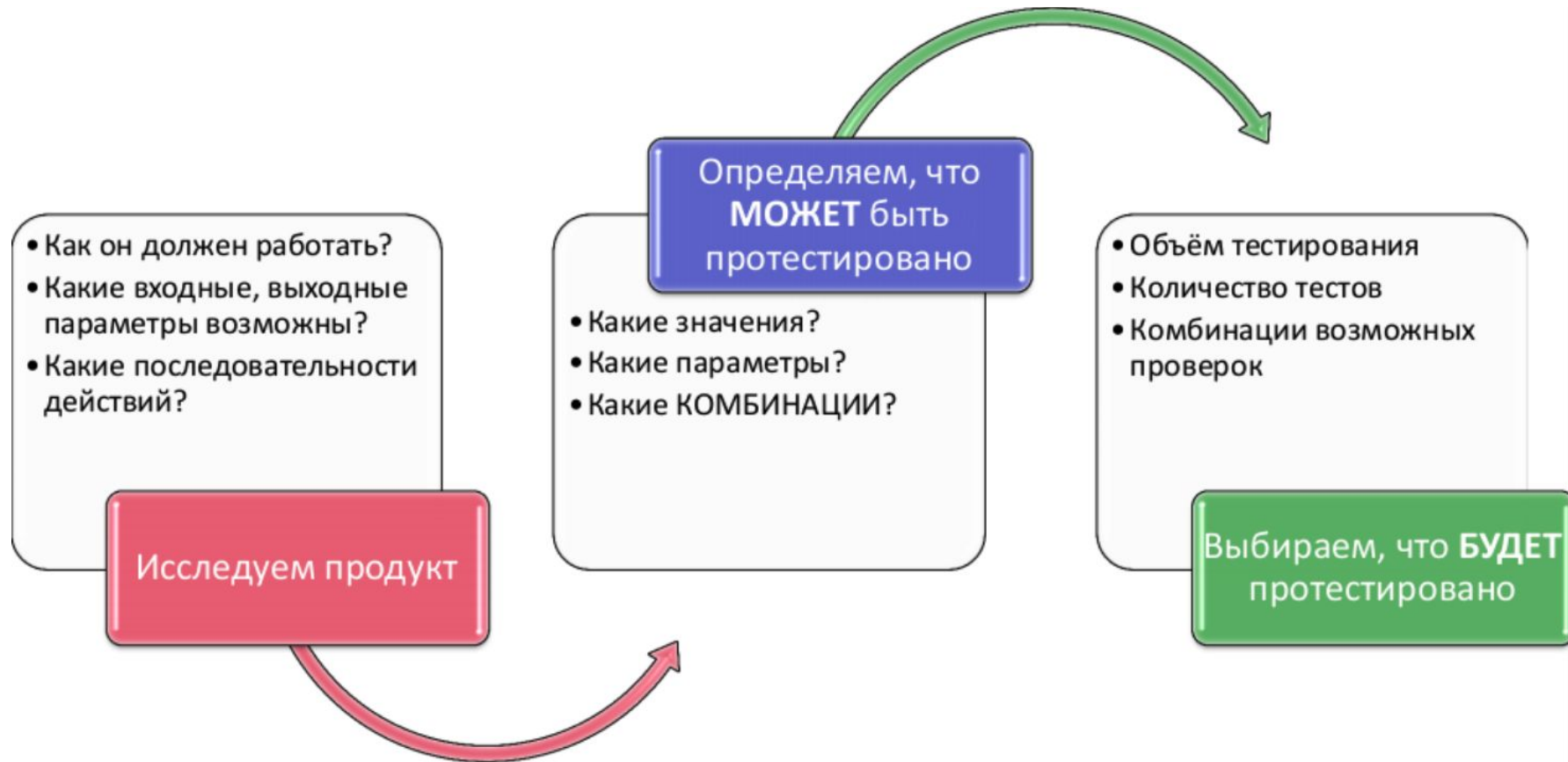
Введите email

Хотите получать новостную рассылку?

Как тестировать?  
Сколько тестов?  
Как протестировать  
БЫСТРО?  
Как протестировать  
ПОЛНОЦЕННО?

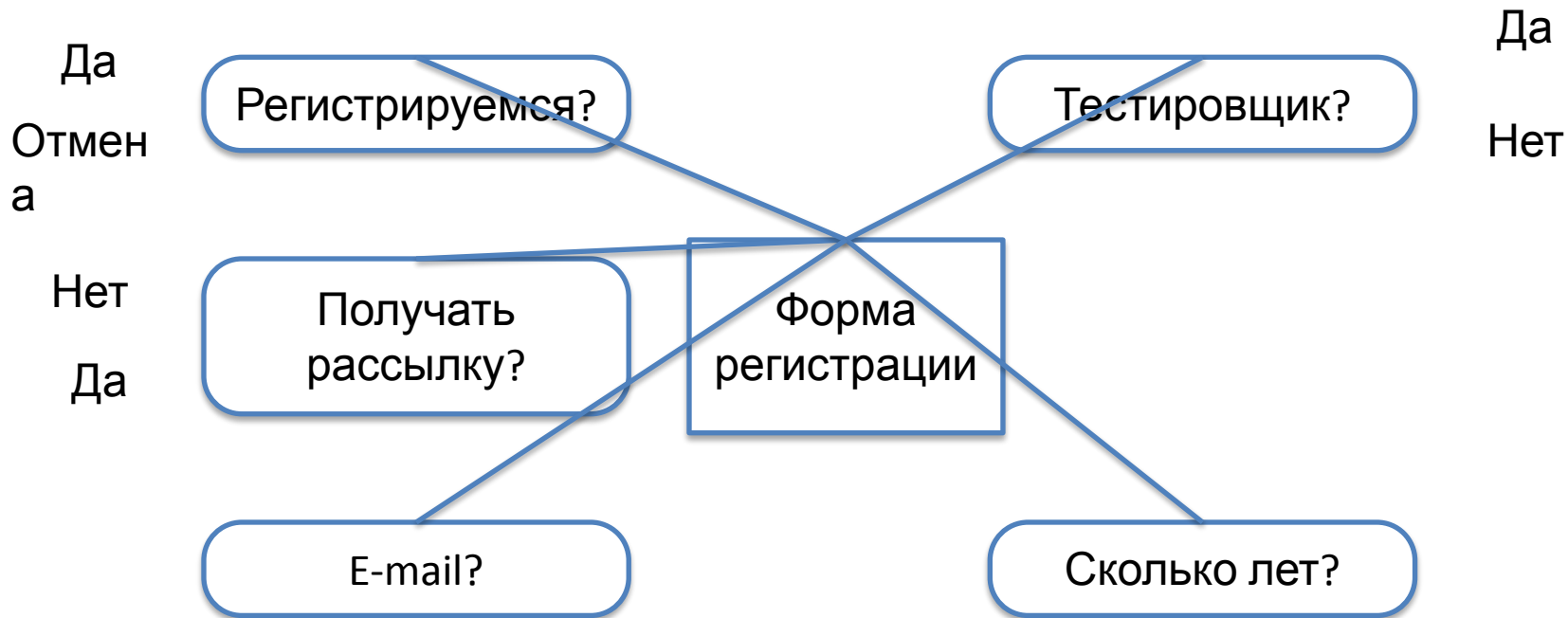
Проектирование тестов = компромисс между качеством тестового покрытия и скоростью тестирования

# T | Проектирование тестов



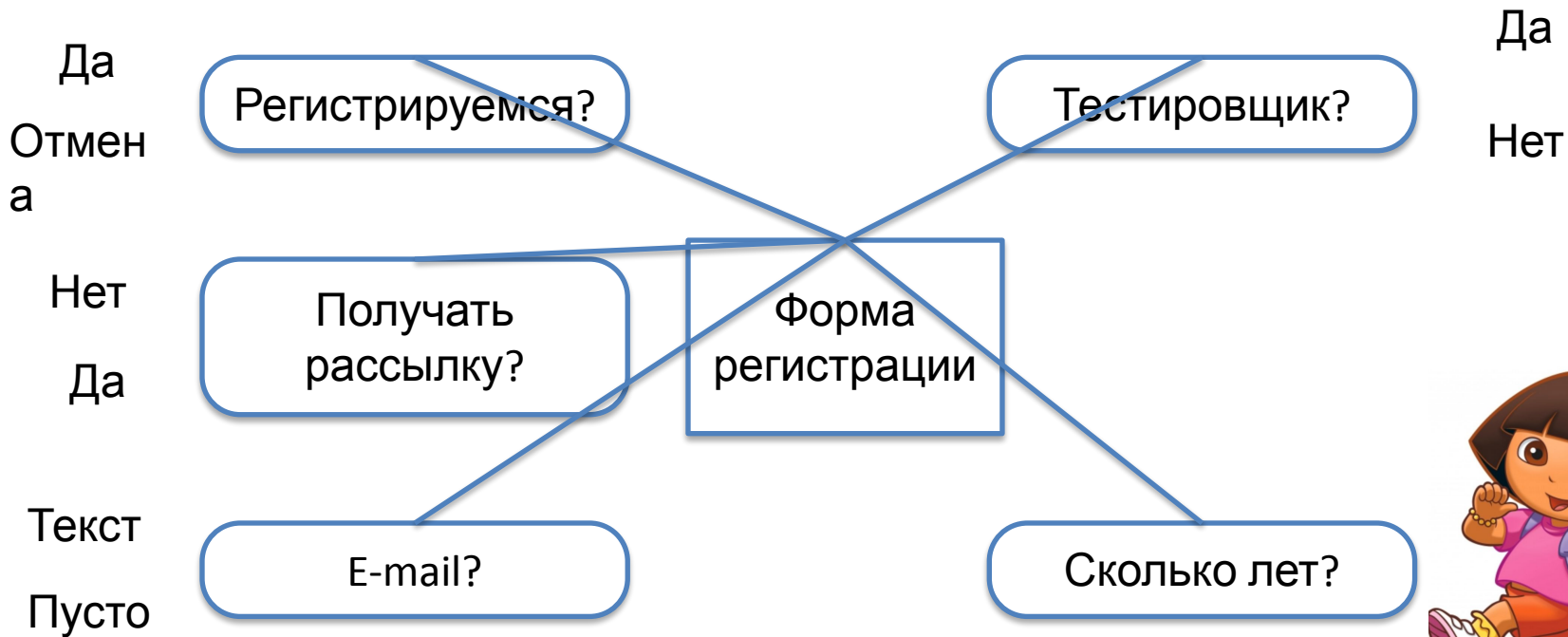
# T | Исследуем продукт

---

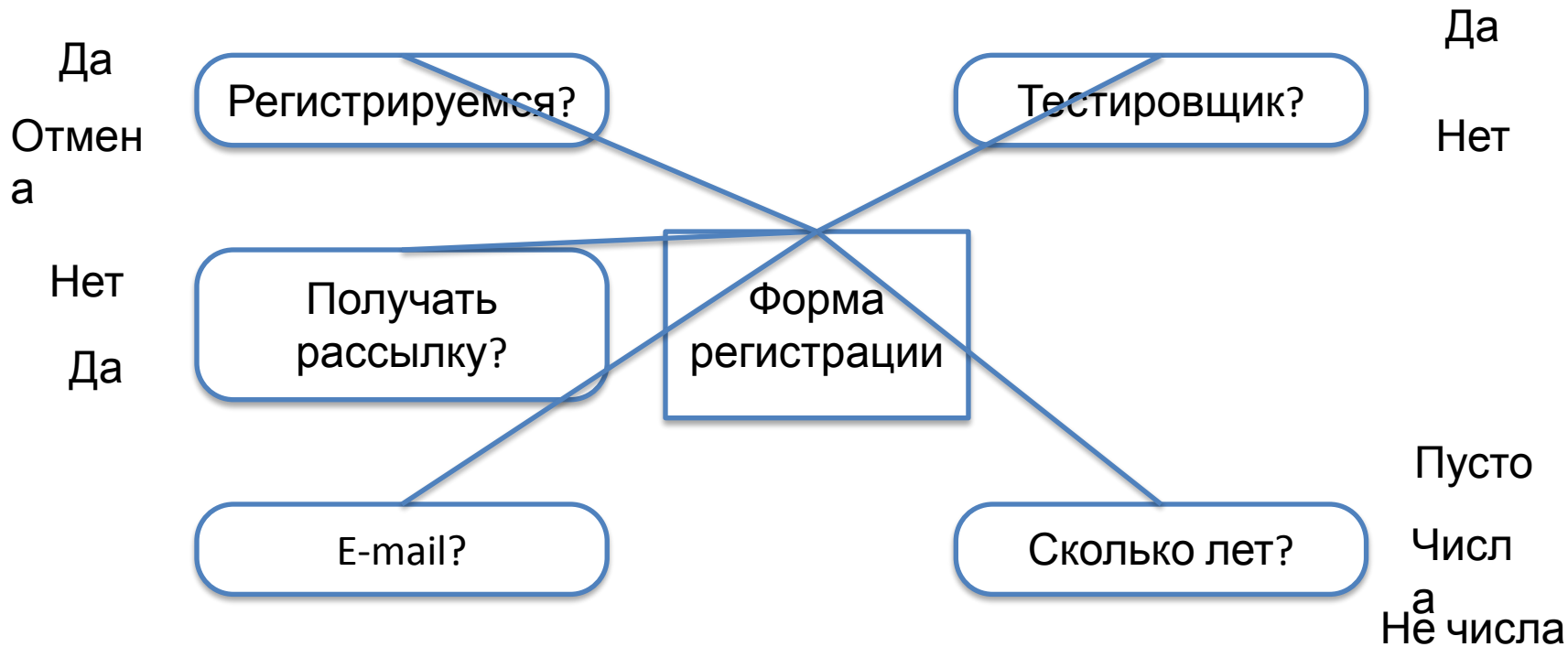




# Т | Исследуем продукт



# T | Исследуем продукт



# **T** | Разделение по категориям

---

Разделение по категориям основано на серии декомпозиций всего множества входных данных

Каждая декомпозиция зависит от характеристик этого множества

## **Основу метода составляют три шага:**

Создание набора категорий, описывающих свойства множества входных данных.

Разделение каждой категории на подмножества значений или диапазонов значений (классов эквивалентности), каждое из которых по специфике отличается от другого.

Определение условий, при которых одни подмножества влияют на другие.

# Т | Разделение по категориям

1. Выделяем категорию «содержимое поля ввода»:

Числа, Не числа, Ничего

2. Разделяем каждую категорию на подмножества:

2.1 для категории «Ничего» нет множества значений.

2.2 для категории «Не числа» нам неважно, какие «не числа» вводить. Все значения в одно множество.

2.3 Для категории «Числа»:

Сколько лет?

Пусто

Числ

Не числа

0

18

Неизвестный максимум





# Т | Анализ граничных значений

Какие грузовики  
Пройдут под мостом,  
а какие - нет?



# Т | Анализ граничных значений

Все грузовики ниже 4,5м

Все грузовики выше 4,5м



# Т | Анализ граничных значений

Если 4,5 пройдёт - то и 4,49 пройдёт

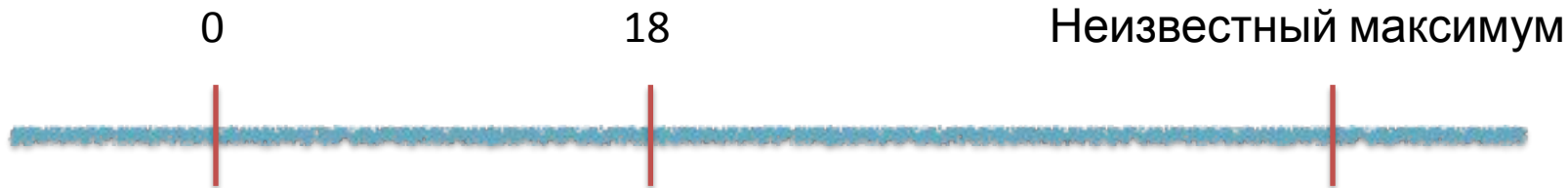
Если 4,51 не пройдёт - то и 5,5 не пройдёт

А если 3 пройдёт - то о чём это говорит?



# Т | Анализ граничных значений

---



Каков неизвестный максимум?

Число на 1 больше максимума образует границу еще одной категории.

Обычно для проверки «позитивных» классов берется еще одно типичное значение из каждого класса эквивалентности.

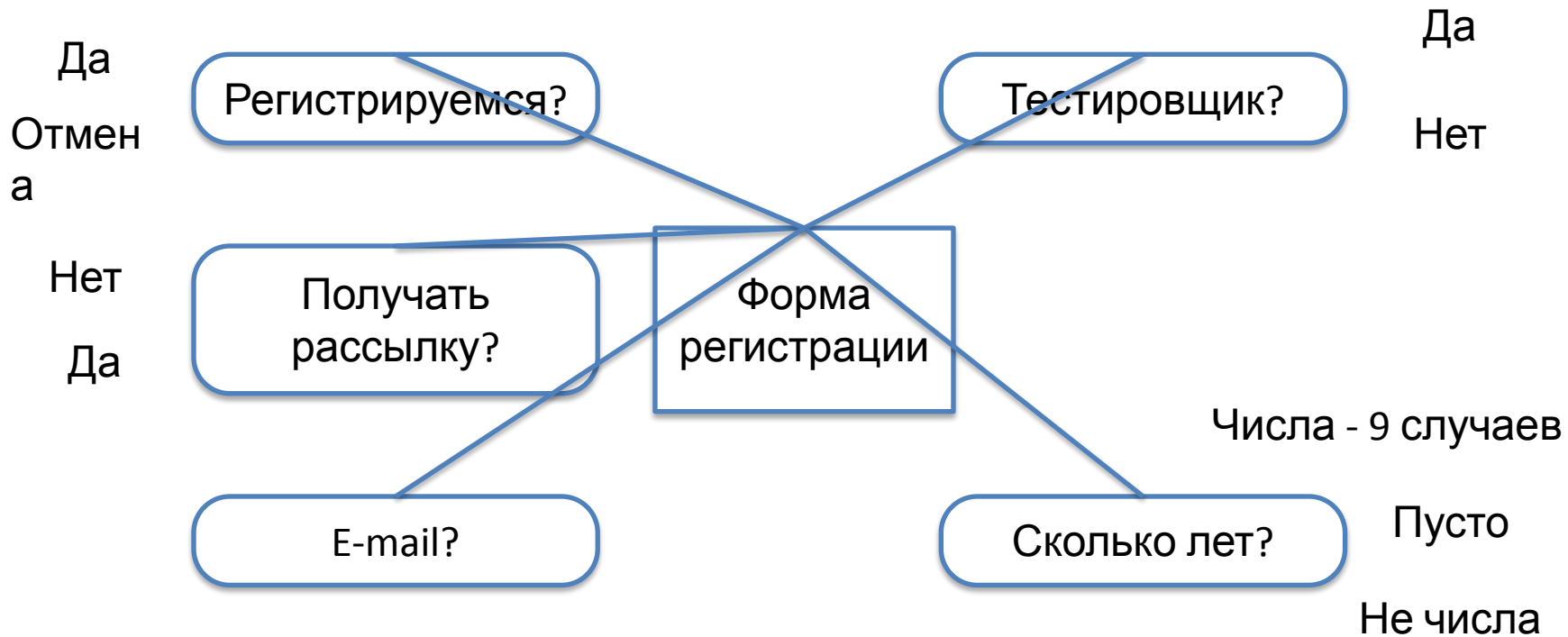
# Т | Анализ граничных значений

---

Имеем следующие тесты:

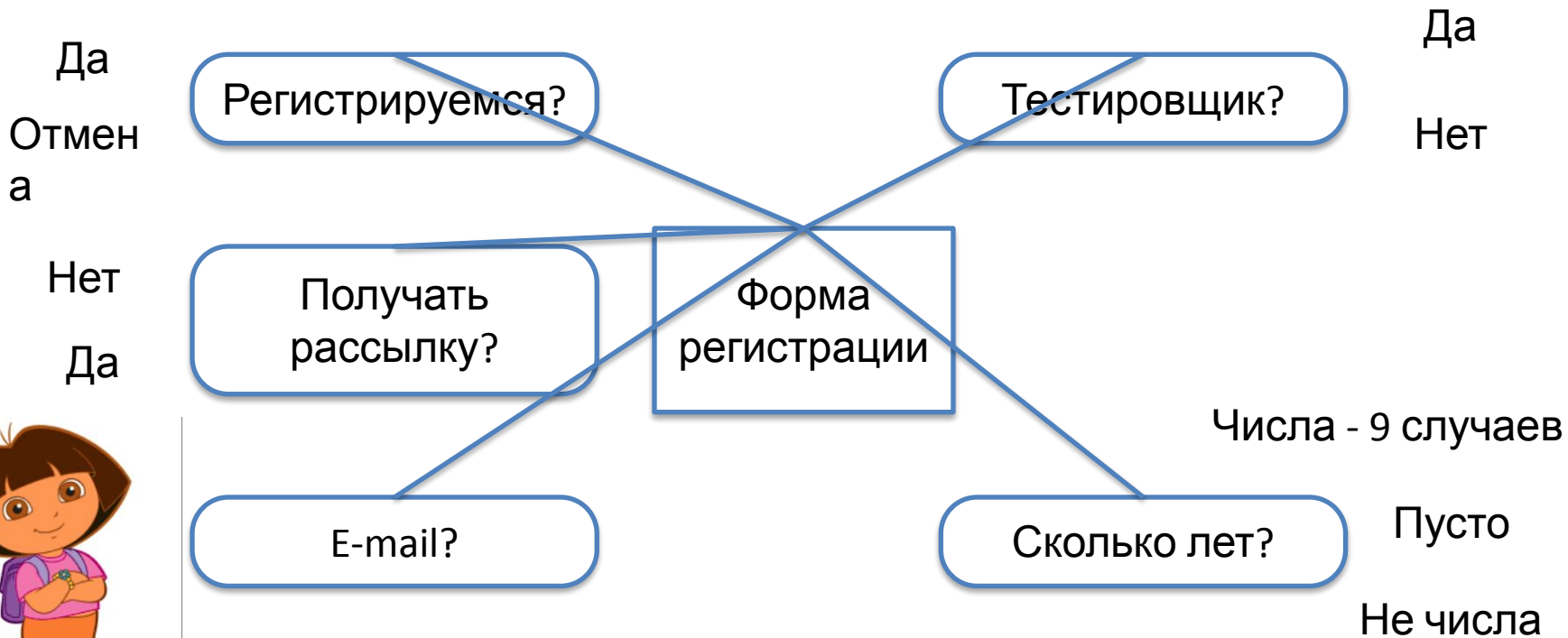
-1	Граница отрицательных чисел
0	Граница класса «несовершеннолетний»
17	Граница класса «несовершеннолетний»
18	Граница класса «совершеннолетний»
25	Типичное значение класса «совершеннолетний»
255	Граница класса «совершеннолетний»
256	Граница недопустимо больших значений
“строка”	Значение класса «Не число»
“”	Значение класса «Пусто»

# T | Исследуем продукт

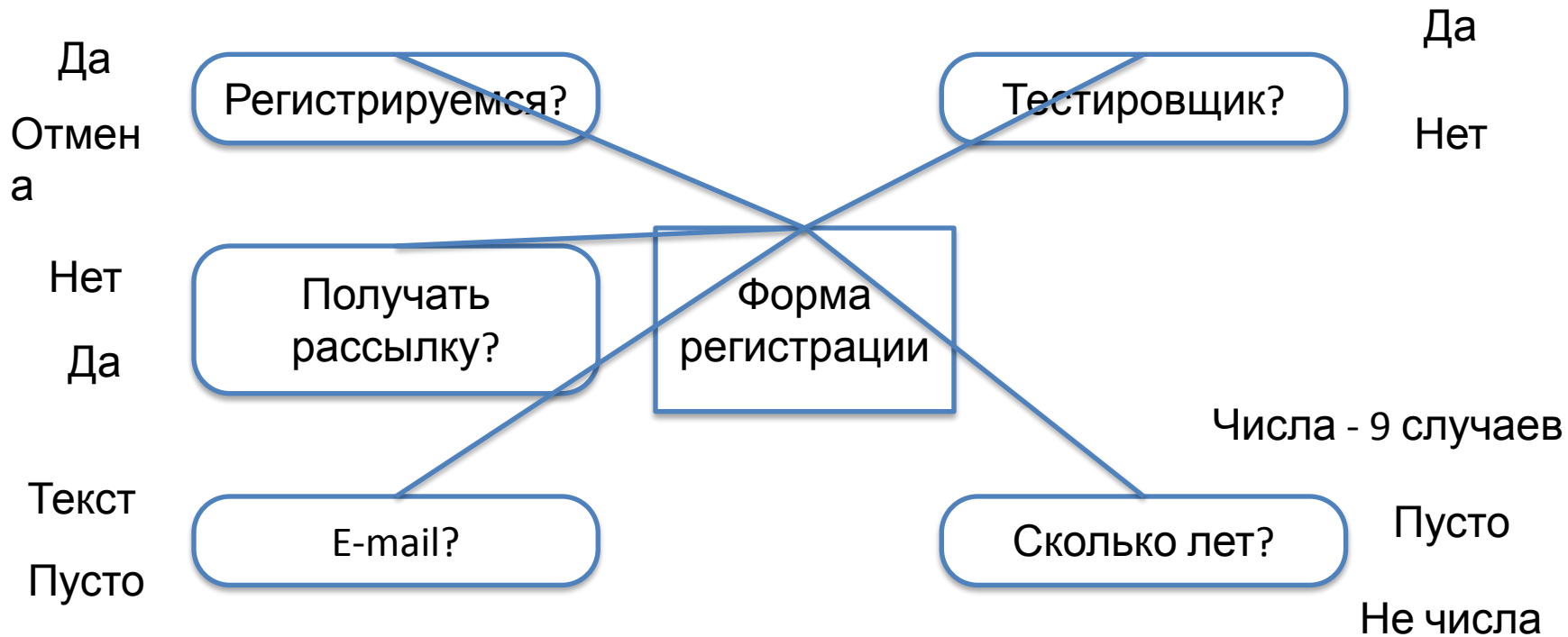




# Т | Исследуем продукт



# T | Исследуем продукт



А теперь спроектируем тесты для e-mail.

**Синтаксическое тестирование** – метод проверки для:

1. Командно-управляемого ПО
2. Элементов ПО, где требуется проверка корректности некоторого ввода.

Синтаксическое тестирование сводится к разбору строки и ответу на вопрос: «соответствует ли строка определенным для нее правилам?»

Правила для строки называются грамматикой

# Т | Форма Бэкуса-Наура

Форма Бэкуса-Наура (сокращенно БНФ) – формальный метод записи грамматики, в которой одни синтаксические элементы последовательно определяются через другие.

```
<вещ.число> ::= <целое> | <целое>.<хвост> 12 | 12.34
<целое> ::= 0 | <цифрабез0> | <цифрабез0><целое> 0 | 5 | 305
<хвост> := <цифра> | <цифра><хвост> <..>.5 | <..>.534
<цифрабез0> ::= 1|2|3|4|5|6|7|8|9
<цифра> ::= 0|1|2|3|4|5|6|7|8|9
```

БНФ состоит из символов(<нетерминалов>) и букв(терминалов). Начиная с одного символа, символы последовательно заменяются на последовательность букв и символов, пока не останутся одни буквы.

! Если остались символы, строка не соответствует грамматике.

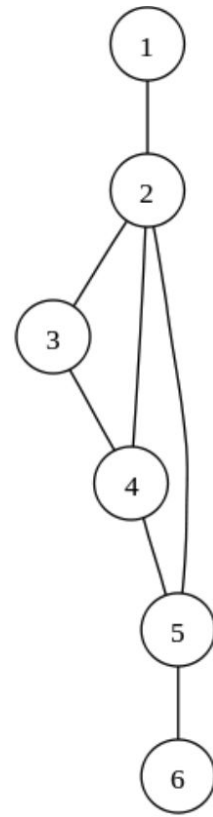
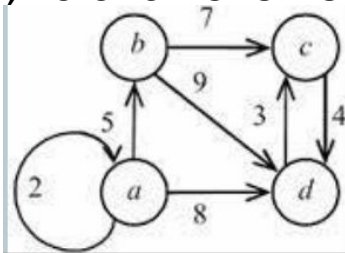
# Т | Минимум из теории графов

Граф – это объект, состоящий из вершин и связей между ними - ребер.

Графы можно записать разными способами. Наиболее наглядная – графическая.

Граф наз. ориентированным, если все его ребра имеют направления. Направленные ребра = дуги.

Если ребро или дуга помечена символом или числом, то она взвешена этим символом/числом.



# **Т** | Две разновидности синтаксического тестирования

---

## **Чистое:**

Построим граф, соответствующий грамматике. Обеспечим покрытие всех ребер графа. Подбираются такие тесты, которые нормально распознаются графом – позитивные тесты.

Дополнительные тесты для циклов в графе.

## **Грязное:**

Тесты подбираются с синтаксическими ошибками, специально чтобы нарушить нормальную работу программы.



# T | Грамматика для e-mail

---

Любой адрес содержит @:

`<address> ::= <prefix>@<suffix>`

Префикс – последовательность слов, разделенная точкой, либо просто слово:

`<prefix> ::= <word> | <prefix>.<word>`

Суффикс – это тоже последовательность слов или слово (то есть префикс), но при этом в конце должен быть указан домен:

`<suffix> ::= <prefix>.<domain>`

Домен – две или три буквы:

`<domain> ::= <letter><letter> | <letter><letter><letter>`

Буква – большая или маленькая буква латинского алфавита:

`<letter> ::= a | b | ... | z | A | B | ... | Z`

Слово может содержать не только буквы – это один или несколько символов:

`<word> ::= <symbol> | <symbol><word>`

Символ – это буква или цифра или знаки «\_», «-»:

`<symbol> ::= <letter> | 0 | 1 | ... | 9 | _ | -`

# Т | Откуда брать грамматику в реальном ПО

---

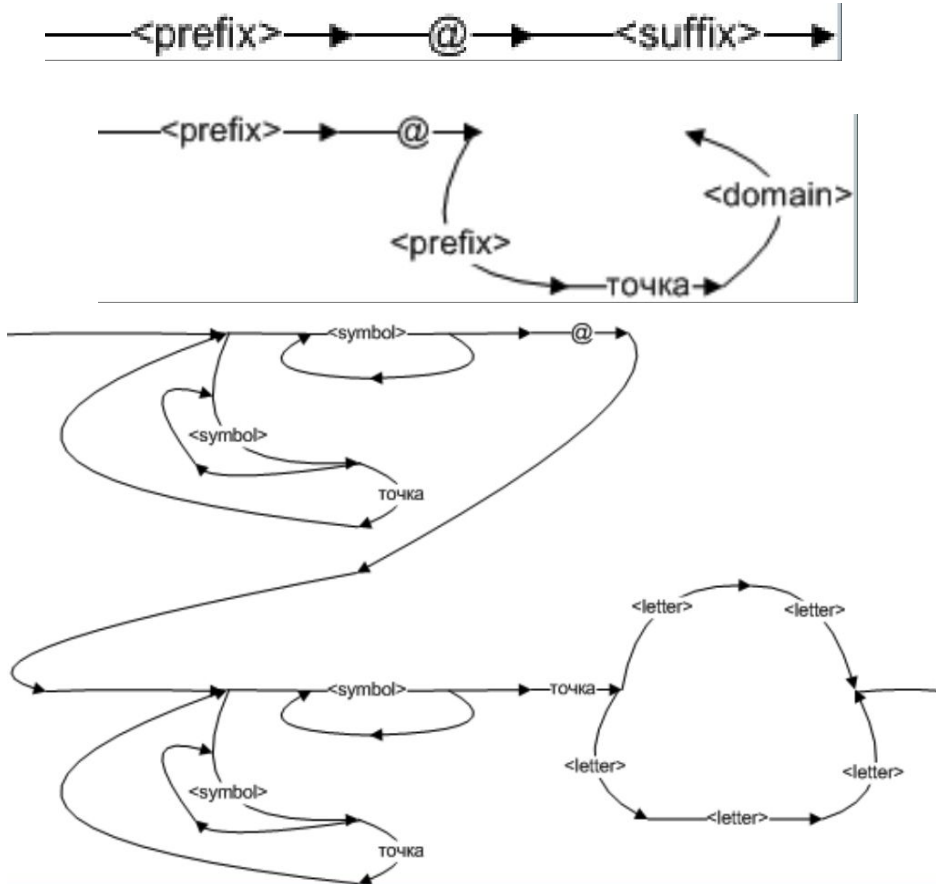
Грамматика может быть задана. Парсер формата DTF (date-time format).

Построить самому на основе:

- a) Неформальной спецификации на естественном языке
- b) Информации о синтаксисе команд из файла справки (help)

! Метод синтаксического тестирования может оказаться трудоемким. Перед применением следует оценить затраты ресурсов и целесообразность применения.

# Т | Граф на основе грамматики



$\langle \text{letter} \rangle ::= a | b | \dots | z | A | B | \dots | Z$   
 $\langle \text{symbol} \rangle ::= \langle \text{letter} \rangle | 0 | 1 | \dots | 9 | \_ | -$   
 $\langle \text{word} \rangle ::= \langle \text{symbol} \rangle | \langle \text{symbol} \rangle \langle \text{word} \rangle$   
 $\langle \text{prefix} \rangle ::= \langle \text{word} \rangle | \langle \text{prefix} \rangle . \langle \text{word} \rangle$   
 $\langle \text{domain} \rangle ::=$   
 $\langle \text{letter} \rangle \langle \text{letter} \rangle | \langle \text{letter} \rangle \langle \text{letter} \rangle \langle \text{letter} \rangle$   
 $\langle \text{suffix} \rangle ::= \langle \text{prefix} \rangle . \langle \text{domain} \rangle$   
 $\langle \text{address} \rangle ::= \langle \text{prefix} \rangle @ \langle \text{suffix} \rangle$

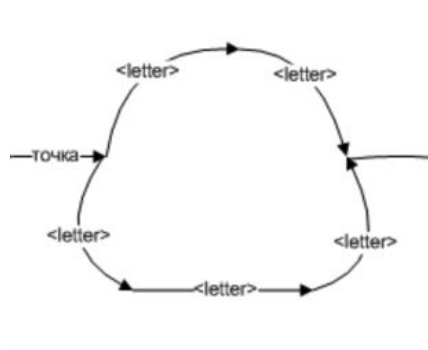
# Т | Чистые (позитивные) тесты

Для покрытия ребер необходимо всего два теста:

abcdefghijklmnopqrstuvwxyz-0123456789\_.abcdefghijklmnopqrstuvwxyz-0123456789@  
abcdefghijklmnopqrstuvwxyz-0123456789\_.abcdefghijklmnopqrstuvwxyz-0123456789.ru

ABCDEFGHIJKLMNOPQRSTUVWXYZ.VWXYZ.ABCDEFGHIJKLMNOPQRSTUVWXYZ@  
ABCDEFGHIJKLMNOPQRSTUVWXYZ.VWXYZ.ABCDEFGHIJKLMNOPQRSTUVWXYZ.xyz

Если раскрывать <letter> в домене и обеспечивать покрытие всех букв, то это приведет еще к  $(26+26-5)/3$  тестам (26 маленьких букв, 26 больших, 5 уже рассмотрено, можно максимум по 3 буквы в домен). Это простые тесты типа: a@a.abc, a@a.def., ... , a@a.ABC, a@a.DEF, ..., a@a.XYZ



# Т | Грязные (негативные) тесты

---

Спецификация записывается по уровням:

Level 1:  $\langle \text{address} \rangle ::= \langle \text{prefix} \rangle @ \langle \text{suffix} \rangle$

Level 2:  $\langle \text{prefix} \rangle ::= \langle \text{word} \rangle | \langle \text{prefix} \rangle . \langle \text{word} \rangle$

Level 2:  $\langle \text{suffix} \rangle ::= \langle \text{prefix} \rangle . \langle \text{domain} \rangle$

Level 3:  $\langle \text{word} \rangle ::= \langle \text{symbol} \rangle | \langle \text{symbol} \rangle \langle \text{word} \rangle$

Level 3:  $\langle \text{domain} \rangle ::= \langle \text{letter} \rangle \langle \text{letter} \rangle | \langle \text{letter} \rangle \langle \text{letter} \rangle \langle \text{letter} \rangle$

Level 4:  $\langle \text{symbol} \rangle ::= \langle \text{letter} \rangle | 0 | 1 | \dots | 9 | \_ | -$

Level 5:  $\langle \text{letter} \rangle ::= a | b | \dots | z | A | B | \dots | Z$

Ошибки вносятся последовательно на каждый уровень.

! Одна ошибка – один тест.

# Т | Грязные (негативные) тесты

---

## Level 1:

Здесь префикс и суффикс разделяет один символ @. Сделаем два @ или ни одного @.  
a@@a.ru, aa.ru

## Level 2:

Префикс – это слово или несколько слов через точку. Пусть слово отсутствует.

@a.ru, a.@a.ru, a..a@a.ru

Суффикс – это префикс и домен через точку. Внесем те же ошибки для префикса, и отдельно для домена: a@ru, a@a..ru, a@.ru, a@a..a.ru, a@a.

## Level 3:

Слово – это один или несколько символов. Недопустимым будет отсутствие хотя бы одного символа, но это означает отсутствие слова, а этот тест уже есть.

Домен – две или три буквы. Возьмем одну или четыре буквы. Отсутствие домена уже проверено уровнем выше. a@a.r, a@a.ruru

## Level 4:

Символ – это буква или цифра или \_ или -. Возьмем вместо них другие символы.

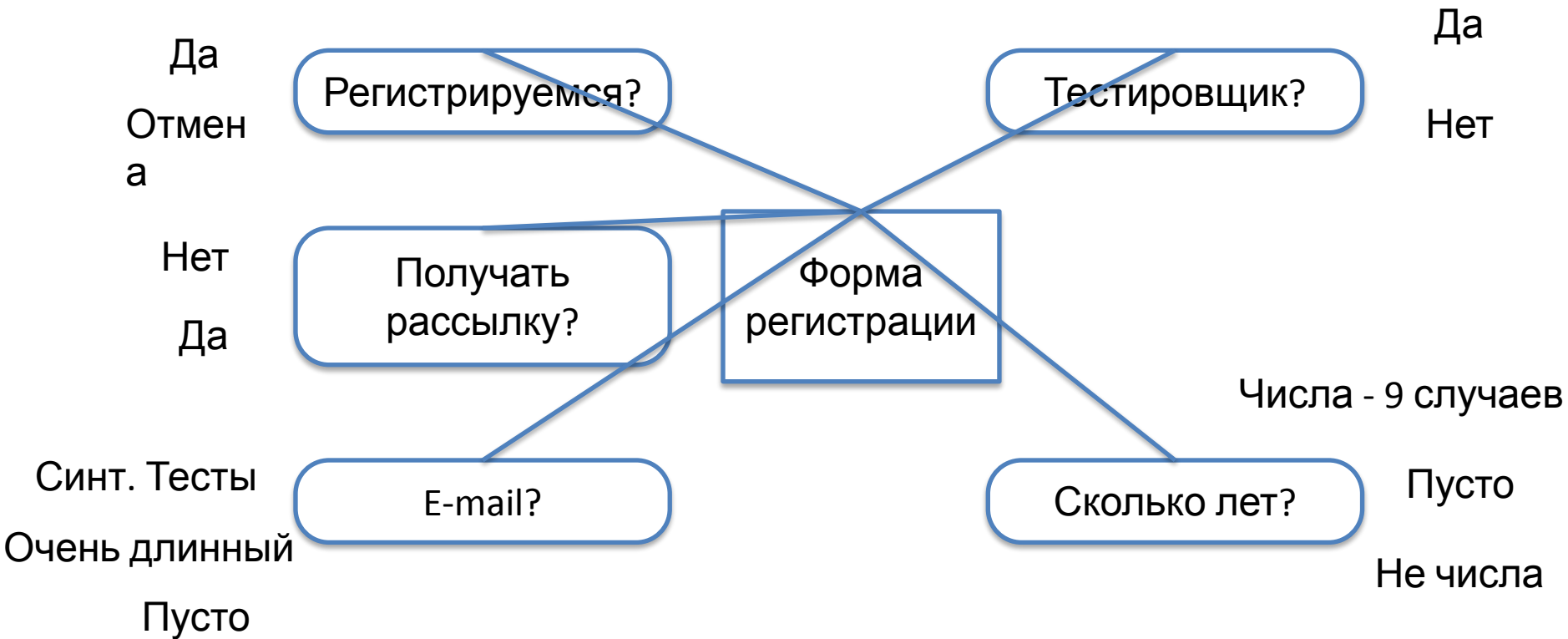
No.!.#.^%.(.).+.\.@~:~!?.ru

## Level 5:

Буква – это латинские буквы

А.Б.В.Г.Д@е.ё.ж.з.й.ru, b@b.бв

# T | Исследуем продукт





# ТЕХНОПОЛИС |

## Лекция 2.2

### Комбинаторика тестов

# T | Таблица значений

Регистрируемся	Получать рассылку	Тестируешь	Возраст	E-mail
Да	Да	Да	-1	<a href="mailto:my@mail.ru">my@mail.ru</a>
Отмена	Нет	Нет	0	m@@ibn.com
			17	Пусто
			18	<u>Очень длинный</u>
			25	
			255	
			256	
			“строка”	
			Пусто	

# Т | Негативные тесты

Негативные тесты не участвуют в комбинаторике тестов!

- В «типичный» позитивный тест по одному поочередно вносятся негативные значения. Почему так?
- Имеем количество тестов - 6

Регистрируемся	Получать рассылку	Тестируемый	Возраст	E-mail
Да	Да	Да	-1	<a href="mailto:my@mail.ru">my@mail.ru</a>
Да	Да	Да	256	<a href="mailto:my@mail.ru">my@mail.ru</a>
Да	Да	Да	“строка”	<a href="mailto:my@mail.ru">my@mail.ru</a>
Да	Да	Да	Пусто	<a href="mailto:my@mail.ru">my@mail.ru</a>
Да	Да	Да	25	<a href="mailto:m@@ibn.com">m@@ibn.com</a>
Да	Да	Да	25	Пусто
Да	Да	Да	25	<a href="mailto:my@mail.ru">my@mail.ru</a>



# Т | Метод минимальных проверок

- В каждом тесте комбинируется максимальное число значений.
- Когда все значения параметра уже поучаствовали в тесте, можно в другие тесты подставлять типичное значение или чередовать.
- Метод дает минимально допустимое количество тестов.
- Кол-во тестов = максимальное кол-во значений у параметра - 5

Регистрируемся	Получать рассылку	Тестировщик	Возраст	E-mail
Да	Да	Да	17	<a href="mailto:my@mail.ru">my@mail.ru</a>
Отмена	Нет	Нет	18	<a href="mailto:my@mail.ru">my@mail.ru</a>
Да	Да	Да	25	<a href="mailto:my@mail.ru">my@mail.ru</a>
Да	Нет	Нет	255	<u>Очень длинный</u>
Да	Да	Да	0	<a href="mailto:my@mail.ru">my@mail.ru</a>

# Т | Перебор значений

- Набор тестов содержит все возможные комбинации параметров.
- Позволяет проверить не только сами значения, но и их влияние друг на друга.
- Метод обеспечивает максимальное тестовое покрытие.
- Кол-во тестов = умножение кол-ва значений всех параметров:  $2*2*2*5*2 = 80$

Регистрируемся	Получать рассылку	Тестируешь	Возраст	E-mail
Да	Да	Да	17	<a href="mailto:my@mail.ru">my@mail.ru</a>
Отмена	Да	Да	17	<a href="mailto:my@mail.ru">my@mail.ru</a>
Да	Нет	Да	17	<a href="mailto:my@mail.ru">my@mail.ru</a>
Отмена	Нет	Да	17	<a href="mailto:my@mail.ru">my@mail.ru</a>
Да	Да	Нет	17	<a href="mailto:my@mail.ru">my@mail.ru</a>
...	...	...	...	...

# Т | Метод атомарных проверок

- Определяется типичный тест
- Каждый следующий тест отличается от предыдущего ровно одним значением.
- Дефекты легко локализуемы по результатам тестов
- Кол-во тестов = сумма значений – кол-во параметров:  $13 - 5 = 8$

Регистрируемся	Получать рассылку	Тестировщик	Возраст	E-mail
Да	Да	Да	17	<a href="mailto:my@mail.ru">my@mail.ru</a>
Отмена	Да	Да	17	<a href="mailto:my@mail.ru">my@mail.ru</a>
Да	Нет	Да	17	<a href="mailto:my@mail.ru">my@mail.ru</a>
Да	Да	Нет	17	<a href="mailto:my@mail.ru">my@mail.ru</a>
Да	Да	Да	17	<a href="mailto:my@mail.ru">my@mail.ru</a>
...	...	...	...	...

- По статистике, 97% ошибок кроется в комбинации не более, чем двух параметров.
- Тестовый набор содержит все возможные пары значений разных параметров.
- Для построения набора используются готовые алгоритмы (лучший AllPairs)
- Дефекты сложно локализуемы
- Кол-во тестов = произведение двух максимальных наборов значений:  $2 * 5 = 10$

Регистрируемся	Получать рассылку	Тестируешь	Возраст	E-mail	Количество пар
Да	Да	Да	17	<a href="mailto:my@mail.ru">my@mail.ru</a>	10
Отмена	Нет	Нет	17	Очень длинный	10
Да	Нет	Да	18	Очень длинный	8
Отмена	Да	Нет	18	<a href="mailto:my@mail.ru">my@mail.ru</a>	8
Да	Да	Нет	25	Очень длинный	6
Отмена	Нет	Да	25	<a href="mailto:my@mail.ru">my@mail.ru</a>	6
Да	Нет	Нет	255	<a href="mailto:my@mail.ru">my@mail.ru</a>	4
Отмена	Да	Да	255	Очень длинный	4
Да	Да	Да	0	<a href="mailto:my@mail.ru">my@mail.ru</a>	4
Отмена	Нет	Нет	0	Очень длинный	4

# Т | Метод взаимосвязанных проверок

- Необходимо анализировать, как связаны параметры
- Эффективность метода зависит от квалификации тестировщика
- Полный перебор или pairwise для связанных параметров
- Минимальные проверки для не связанных параметров
- Кол-во тестов – дифференцированное - ?

Регистрируемся	Получать рассылку	Тестировщик	Возраст	E-mail
Да	Да	Да	17	<a href="mailto:my@mail.ru">my@mail.ru</a>
Да	Да	Нет	17	<a href="mailto:my@mail.ru">my@mail.ru</a>
Да	Да	Да	18	<a href="mailto:my@mail.ru">my@mail.ru</a>
Да	Да	Нет	18	<a href="mailto:my@mail.ru">my@mail.ru</a>
...	...	...	...	...
Отмена	Нет	Да	25	<a href="mailto:my@mail.ru">my@mail.ru</a>



# Т | Метод взаимосвязанных проверок

	Минимальные	Перебор	Атомарные	Pairwise	Взаимосвязанные
Количество тестов	5	80	8	10	?
Глубина покрытия	70 %	100	71 %	97 %	?
Простота создания	Легко	Легко	Легко	Средне	Сложно
Локализация дефектов	Сложно	Легко	Легко	Сложно	Средне
Область применения	Неприоритетный функционал, smoke тесты	Критичный функционал, автоматизация	Функционал среднего приоритета	Высокий приоритет, сжатые сроки, автоматизация	Квалифицированные тест-дизайнеры

# ТЕХНОПОЛИС |

## Лекция 2.3

### Другие методы тестирования «черного ящика»

**Таблицы решений (Decision Tables)** – способ представления сложных бизнес-правил (бизнес-логики), которые программа должна реализовывать.

Метод еще называют тест-анализ на основе бизнес-логики.

**Бизнес-логика** - совокупность правил, принципов, зависимостей, ограничений поведения объектов предметной области.  
реализация предметной области в программе.



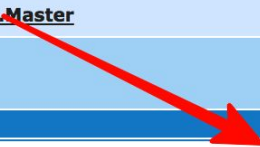
# Т | Таблицы решений

<a href="#">swix41</a>	52	Сегодня, 10:03 Послед.: <b>W.Master</b>
<a href="#">JDante</a>	116	Сегодня, 09:42 Послед.: <b>W.Master</b>

[НОВАЯ ТЕМА](#)


По: последн. сообщению ▾ Я-А ▾ За: все время ▾ Темы: Все ▾

Запомнить эти параметры



От чего зависит поведение при нажатии на кнопку?

# Т | Таблицы решений

<a href="#">swix41</a>	52	Сегодня, 10:03 Послед.: <b>W.Master</b>
<a href="#">JDante</a>	116	Сегодня, 09:42 Послед.: <b>W.Master</b>
		
<a href="#">НОВАЯ ТЕМА</a>		
По: последн. сообщению ▾ Я-А ▾ За: все время ▾ Темы: Все ▾ <input type="button" value="OK"/>		
<input type="checkbox"/> Запомнить эти параметры		

## Бизнес-логика ПО:

Если пользователь зарегистрирован, открыть окно новой темы,  
если нет - не открывать окно

Если пользователь оставлял сообщения последние 60 секунд, выдать ошибку.

Если нет - не выдавать

Если пользователь - модератор, то ограничение на 1 сообщение в 60 секунд не действительно

# Т | Таблицы решений

	1	2	3	4	5	6	7	8
УСЛОВИЯ								
Зарегистрирован	Y	Y	Y	Y	N	N	N	N
Отправлял сообщения в последние 60 секунд	Y	Y	N	N	Y	Y	N	N
Модератор	Y	N	Y	N	Y	N	Y	N
Действия ПО								
Открыть окно создания новой темы	Y	N	Y	Y	N	N	N	N
Выдать ошибку	N	Y	N	N	Y	Y	Y	Y

1. Выписываем все условия.
2. Определяем количество тестов как 2 в степени N. (!Если условия бинарные)
3. Добавляем все возможные значения решений для условий.
4. Анализируем каждый столбец и определяем правильное действие ПО.

# Т | Таблицы решений

	1	2	3	4	5	6	7	8
УСЛОВИЯ								
Зарегистрирован	Y	Y	Y	Y	N	N	N	N
Отправлял сообщения в последние 60 секунд	Y	Y	N	N	Y	Y	N	N
Модератор	Y	N	Y	N	Y	N	Y	N
Действия ПО								
Открыть окно создания новой темы	Y	N	Y	Y	N	N	N	N
Выдать ошибку	N	Y	N	N	Y	Y	Y	Y

Некоторые тесты невозможны – решения противоречат друг другу.  
Итого - 5 тестов

Тестовое покрытие на базе анализа потока управления - оценка покрытия основанная на определении путей выполнения кода программного модуля и создания выполняемых тест кейсов для покрытия этих путей.

**Тестирование циклов** - наиболее распространенная конструкция

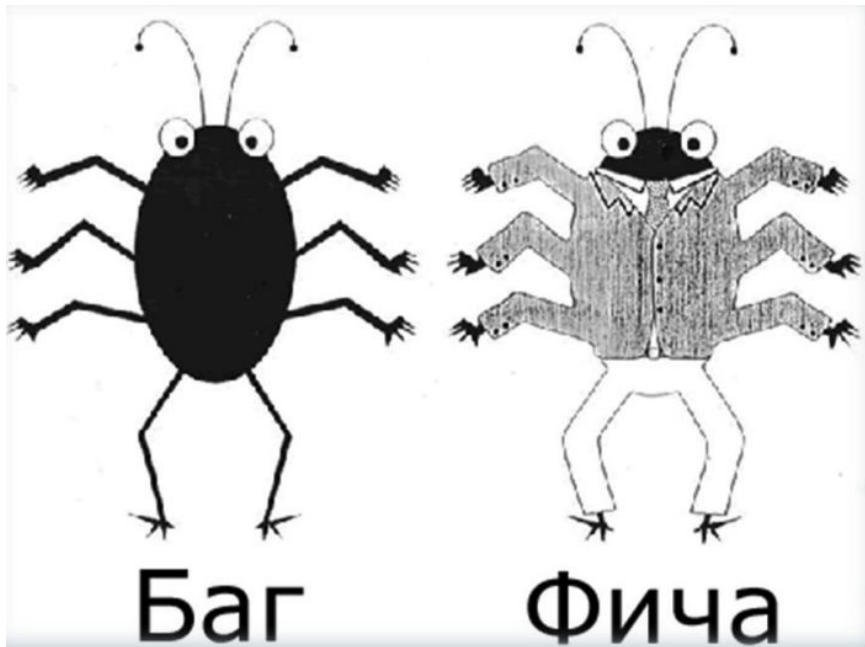
алгоритмов, реализуемых в ПО. Тестирование циклов



# ТЕХНОПОЛИС |

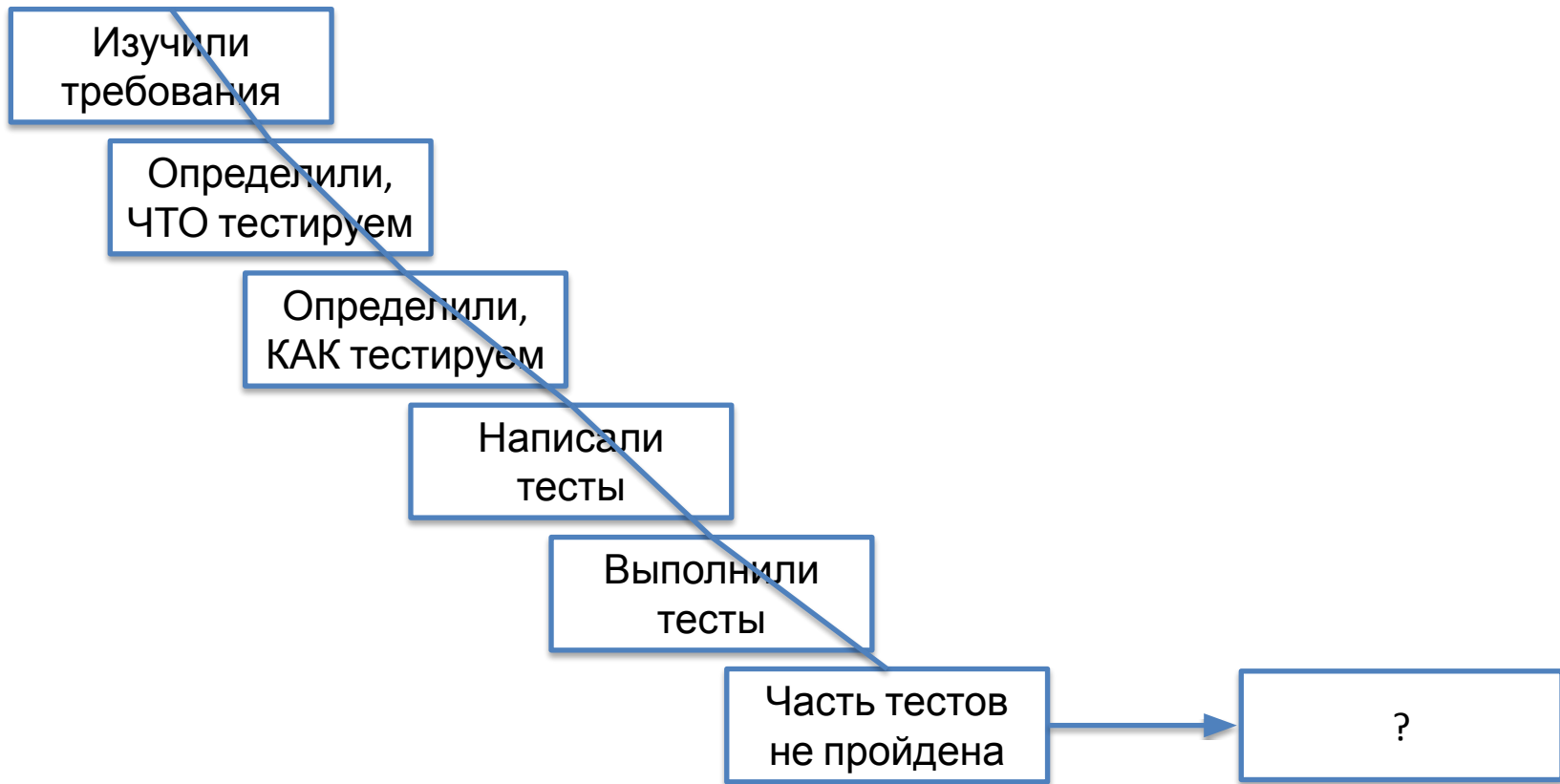
## Лекция 2.4

## Дефекты





# Т Место дефектов в цикле тестирования



# Т | Для чего нужно описывать дефекты

---

- Информирование (программиста, менеджера, заказчика и т.д.) о наличии проблемы
- Локализация проблемы – выявление достаточно четких условий, при которых проблема воспроизводится
- Улучшение механизмов коммуникации внутри команды
- Хранение истории дефектов
- Принятие решений о выпуске продукта



**Все это повышает качество продукта**

**Ошибка (error, mistake)** – ошибка в рассуждениях программиста, его понимании свойств программы.

**Дефект, баг (defect)** - самое общее нарушение каких-либо требований или ожиданий, не обязательно проявляющееся вовне.

**Неисправность (fault, failure)** - наблюдаемое нарушение требований, проявляющееся при каком-то реальном сценарии работы ПО

**Проблема (problem)** – важная для конечного пользователя неисправность.

Человек может допустить **ошибку**. Ошибка приводит к появлению **дефекта** в коде, архитектуре или документации. Если участок кода с дефектом выполняется, программа ведет себя неправильно - появляется **неисправность**. Если такая неисправность важна для пользователя – возникает **проблема**.

# Т | Составляющие дефекта

---

○ **Заголовок (Summary)** – короткое отражение сути проблемы. Одно емкое и информативное предложение (обычно длиной 100-120 символов).

○ **Описание (Description)** – детальное описание проблемы: что, где и как происходит.

○ **Шаги воспроизведения (Steps to reproduce)** – пошаговая инструкция по доведению программы до состояния, когда дефект виден.

○ **Ожидаемый результат (Expected result)** – ожидаемое правильное поведение программы в результате выполнения шагов воспр.

○ **Наблюдаемый результат (Actual result)** – наблюдаемое поведение программы в результате выполнения шагов воспр.

Разница между ожидаемым и наблюдаемым результатом и есть дефект

○ **Приложения (Attachments)** – артефакты, помогающие воспроизведению и пониманию дефекта: файлы логов, скриншоты экрана, тестовые утилиты и т.д.

# Т | Составляющие дефекта. Пример

---

## ○ Заголовок:

При выполнении операции извлечения квадратного корня из отрицательных чисел возникает ошибка «System crash»

## ○ Описание:

Дефект является регрессионным относительно сборки №837. Ошибка проявляется только в том случае, если вид калькулятора – обычный.

## ○ Шаги воспроизведения:

1. Открыть калькулятор;
2. Нажать кнопку '1';
3. Нажать кнопку '+-';
4. Нажать кнопку '√'.

○ Ожидаемый результат: Сообщение «Недопустимый ввод» на дисплее калькулятора.

○ Наблюдаемый результат: Диалоговое окно с текстом «System crash».

# T | Составляющие дефекта. Пример

Mozido Platform / MZP-3887

## getAccountsByCriteria: incorrect security constraints for generic users

Edit Assign Comment More Actions ▾ Analysis In Progress Ready for Dev Workflow ▾

### Details

Type:	Bug	Status:	DevQA In Progress
Priority:	High	Resolution:	Unresolved
Affects Version/s:	build.60	Fix Version/s:	build.60
Component/s:	None		
Labels:	None		

### Description

ADMINISTRATOR can get an account of a generic user and a CSR in his node or a child node of any level.  
ASSOCIATE can get:  
his own account.  
an account of a SUBSCRIBER in his node.  
SUBSCRIBER can get his own account.

Steps to reproduce:

1. Run CreateData project: platform-qa\Test Cases\CreateData\Create-Data-soapui-project.xml
2. Open project platform-qa\Test Cases\Personal Account Management\GetAccountsByCriteria-soapui-project.xml
3. Run 'Prepare'
4. Go to test suite 'Mexico'
5. Run following tests:
  - GAC11 - Caller: Mex Adm
  - GAC12 - Caller: Mex Ass
  - GAC13 - Caller: Mex SC

Expected result: Users must be returned according security constraints.  
Actual result: All generic users returned.



# Т | Классификация дефектов

---

Дефекты недостаточно просто описывать. Их нужно классифицировать.

Существует 2 основных признака классификации:

**Серьезность (Severity)** – степень влияния дефекта на продукт.

- фатальная (fatal, critical)
- серьезная (serious, major)
- ошибка неудобства (inconvenient, minor)
- косметическая (cosmetic)
- предложение по улучшению (improvement, feature request)

**Приоритет (Priority)** – степень важности / срочности исправления дефекта.

- высокий (high)
- нормальный (medium)
- низкий (low)

# T | Классификация дефектов

---

Кто классифицирует дефекты?

**Серьезность (Severity)** – тестировщик, когда описывает дефект.

Серьезность тем выше, чем больше негативные последствия от наличия дефекта.

**Приоритет (Priority)** – менеджер проекта. Это инструмент по планированию работ в рамках текущего этапа работ.

Примеры дефектов с высоким severity и низким priority?

*Невозможно сохранить файл в одном из форматов, НО сохранение через неделю будет переделываться с целью оптимизации.*

Примеры дефектов с низким severity и высоким priority?

*Опечатка в номере телефона горячей линии компании на главной странице сайта.*

# Т | Генерализация и локализация

---

Важно установить границы дефекта. Программист затратит минимум времени на исправление.

Два механизма уточнения:

Генерализация - обобщение. Понять, насколько общим является дефект? Какие свойства конкретного дефекта могут изменяться и при этом дефект по-прежнему будет воспроизводиться.

Локализация – Понять, наличие каких условий приводит к появлению дефекта, а какие условия не важны.

# Т | Генерализация и локализация. Пример

---

**Дефект:** При выполнении операции извлечения корня из -1 возникает ошибка «System crash».

Генерализуем:

- Корень из какого числа приводит к ошибке? Попробуем разные отрицательные, разные положительные, ноль...

=> выяснили, что все отрицательные, а не только -1, приводят к ошибке.

**! Из -1 получили все отрицательные.**

Локализуем:

- Корень любой степени приводит к ошибке? Попробуем разные...  
=> выяснили, что только корень степени 2 приводит к ошибке.

**! Из всех степеней получили только степень 2.**

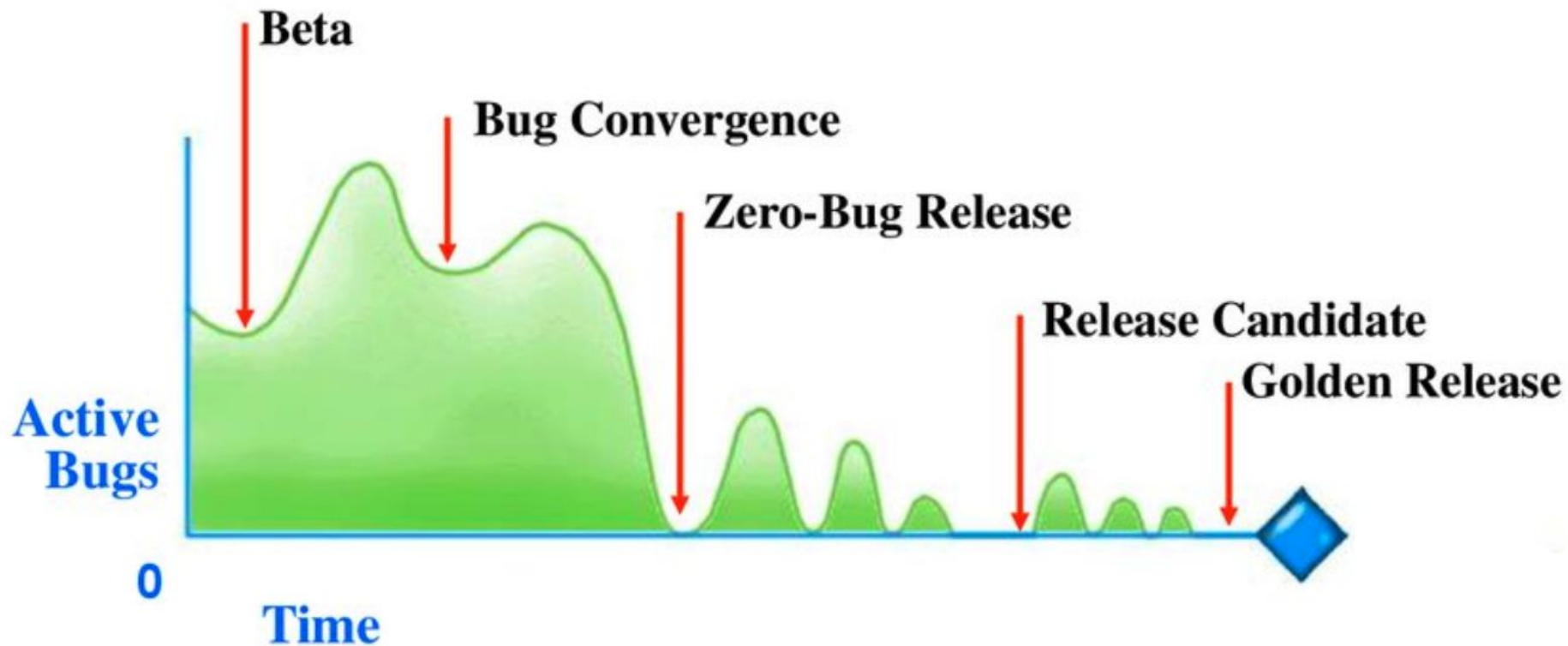
**Дефект (правильный):** При выполнении операции извлечения квадратного корня из отрицательных чисел возникает ошибка «System crash».

# Т | Что важно в описании дефектов

---

Программисту важно:	<ul style="list-style-type: none"><li>• Информация для воспроизведения</li><li>• Локализация и генерализация</li><li>• Дополнительные файлы (логи, скрины)</li></ul>
Аналитику важно:	<ul style="list-style-type: none"><li>• Емкий заголовок</li><li>• Корректная критичность</li><li>• Локализация и генерализация</li></ul>
Руководителю проекта важно:	<ul style="list-style-type: none"><li>• Правильный приоритет</li><li>• Правильная последовательность заведения дефектов</li></ul>

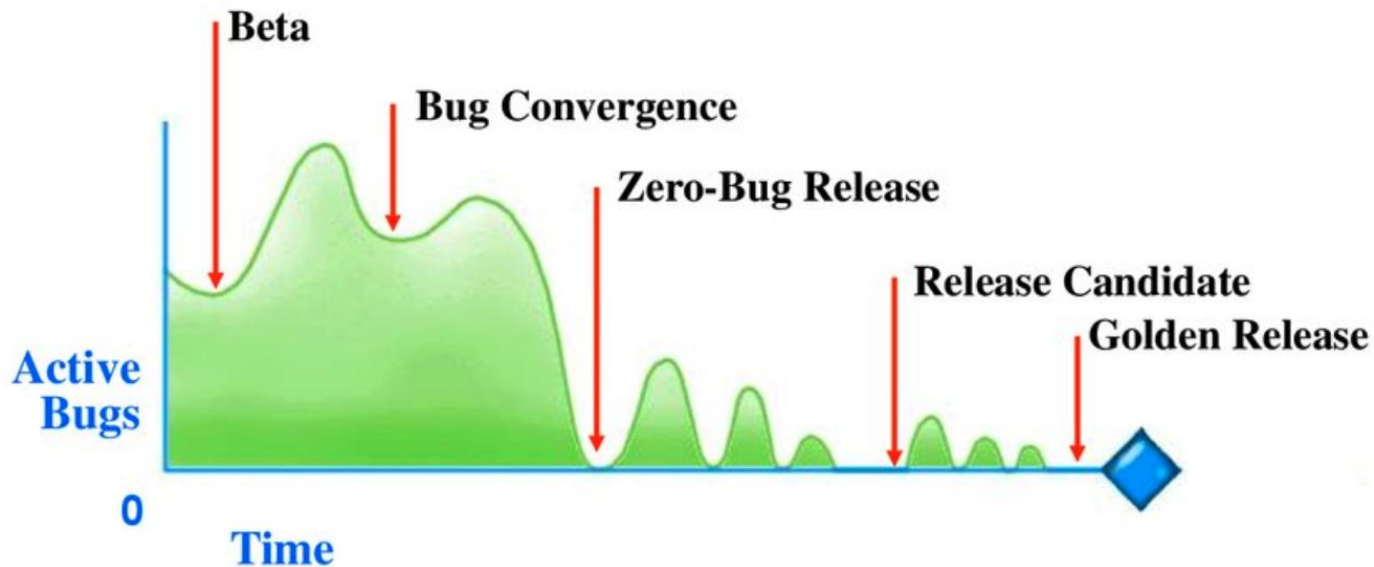
# Т Дефекты и версии продукта



# T | Beta-версия

Beta – версия продукта с основной функциональностью, готовая для тестирования сторонними пользователями. Большая часть дефектов уже закрыта.

Beta-тестирование увеличивает количество дефектов.

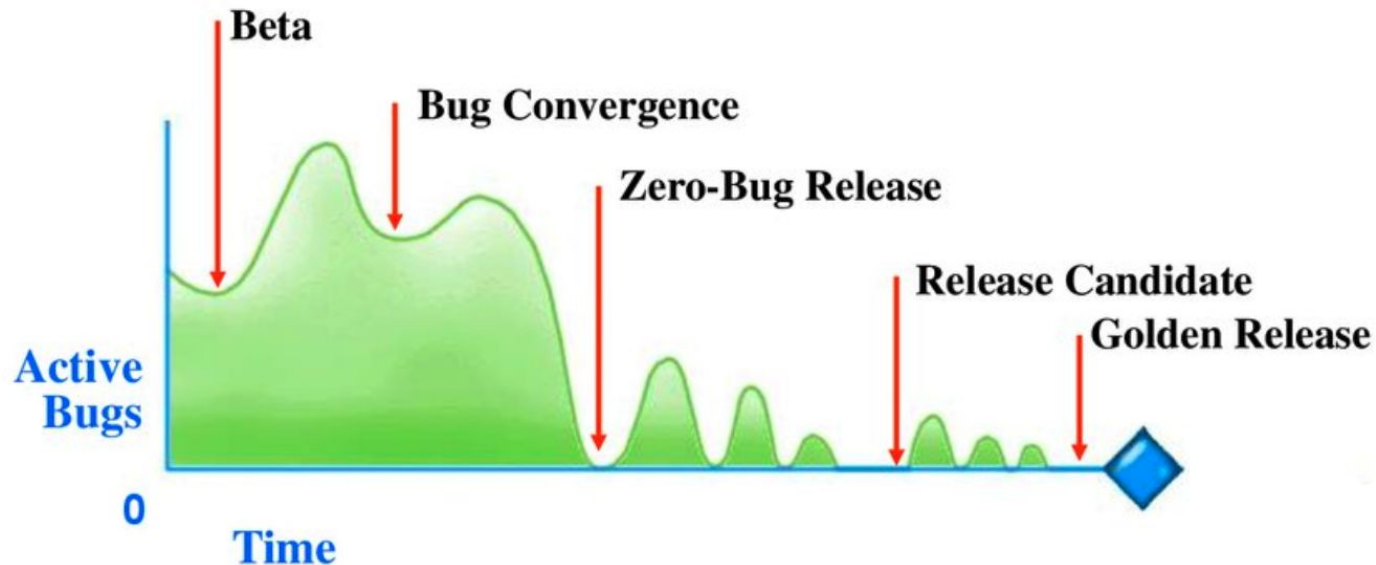


# Т | Точка ковергенции дефектов

Точка проекта, в которой количество исправленных дефектов равно количеству найденных.

Трудно вычислить эту точку, так как количество дефектов – величина постоянно меняющаяся.

Показывает скорее тренд, а не состояние проекта.

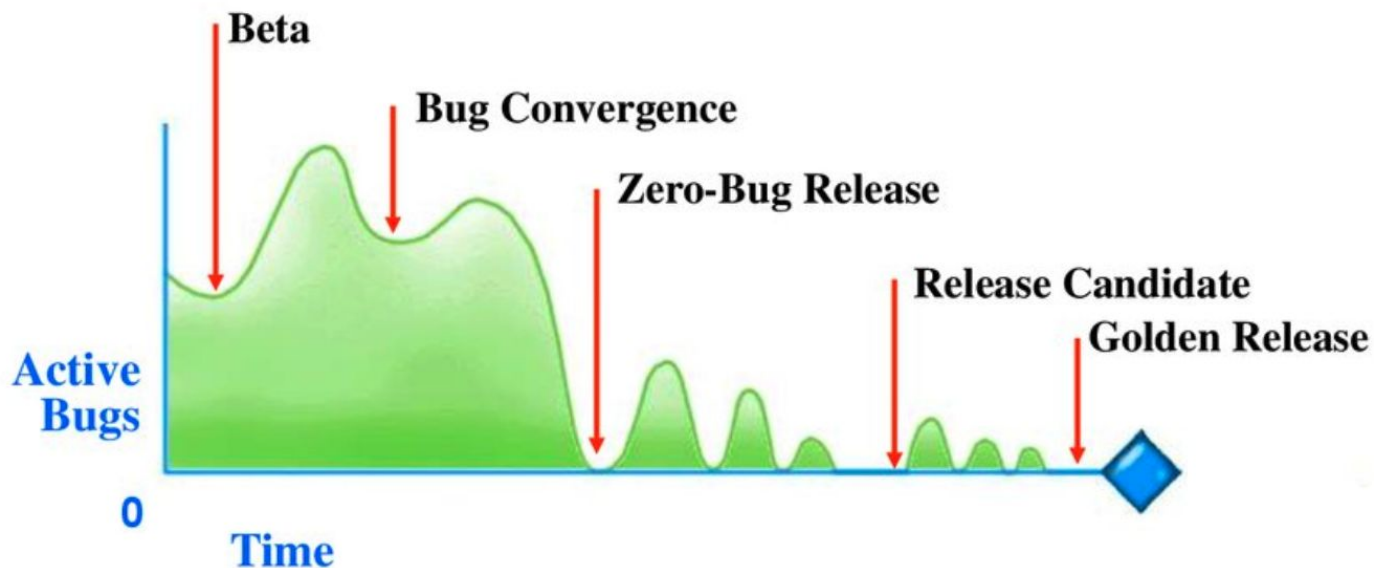




# Т | Точка «Ноль дефектов»

Первая версия продукта, в которой исправлены все дефекты высокого и среднего приоритета.

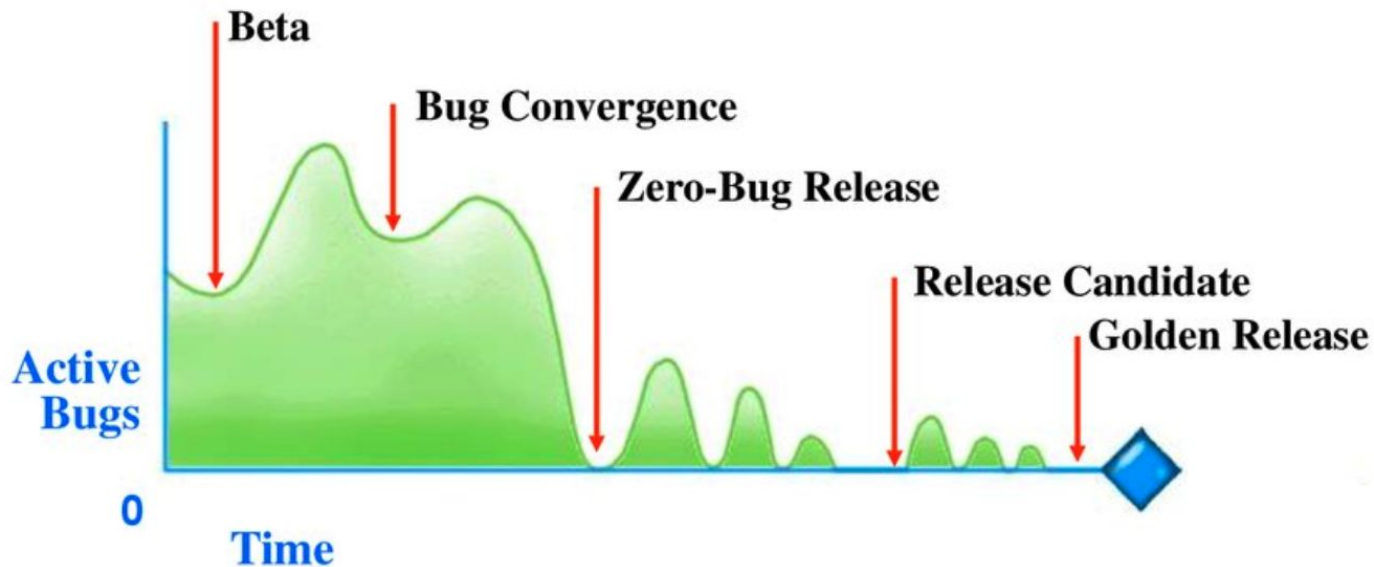
Требует проведения анализа приоритетов дефектов.



# Т | Кандидат

На этой версии проводится финальное тестирование. Продукт потенциально готов к выпуску.

Кандидаты могут появляться один за другим по результатам тестирования.

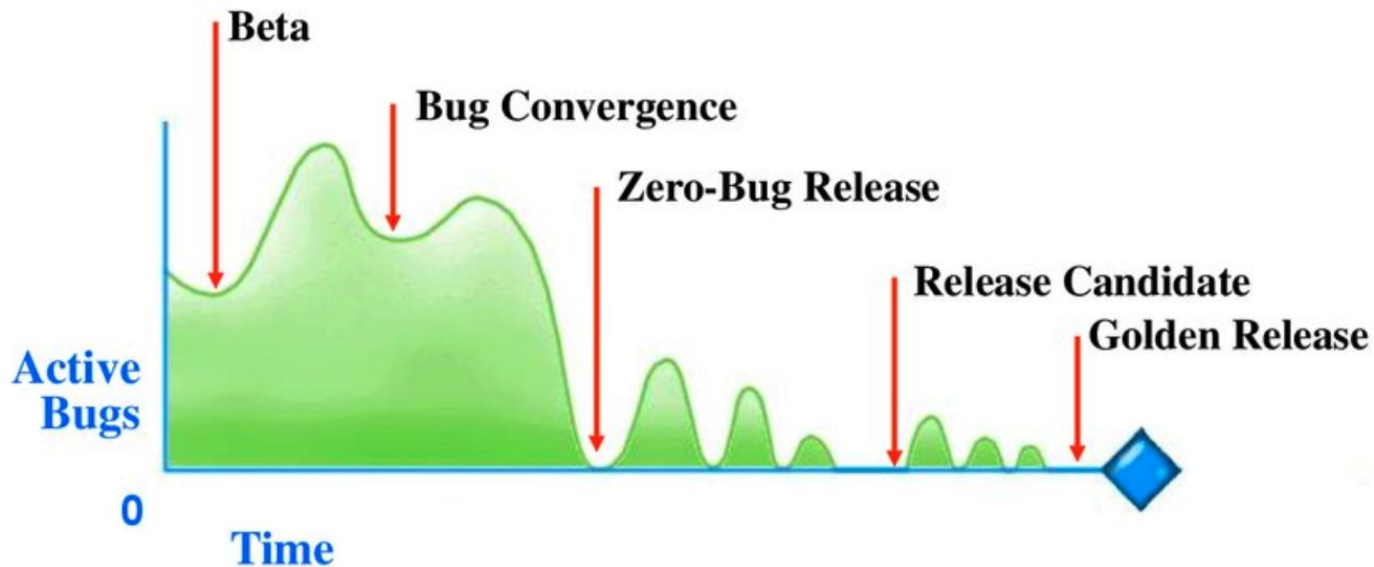


# Т | Утвержденная версия

Утвержденный кандидат. Разработка и тестирование закончены.  
Подписаны все документы.

Дефекты, если они и остались, переносятся

- в следующую версию
- в service pack



# Т Жизненный цикл дефекта



# Т | Верификация дефектов

---

**Верификация дефекта** – процесс проверки исправления, выполненного программистом (confirmation testing).

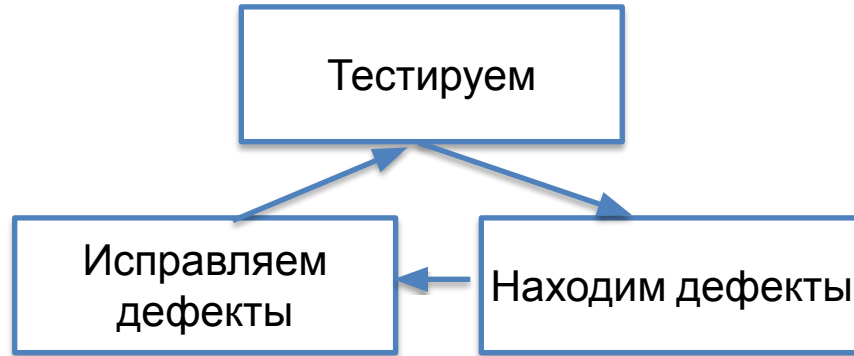
**Цель** – убедиться, что программист верно понял и правильно исправил дефект.

## **Действия:**

1. Точно понять смысл дефекта. Почему он был зафиксирован?
  - Если дефект завели ВЫ, вспомнить обстоятельства.
  - Если дефект завел кто-то другой, воспроизвести дефект.
2. Выполнить шаги воспроизведения и убедиться, что результат теперь соответствует ожидаемому.
3. Выполнить исследовательское мини-тестирование «вокруг» дефекта.
4. Если все ОК, перевести в соответствующий статус (Verified).
5. Если что-то не так:
  - Вернуть на доработку (Verify failed) ИЛИ
  - Завести новый дефект, заблокировав текущий.

# Т | Критерии остановки тестирования

---



Системное тестирование заканчивается, когда мы измерили возможности системы и исправили достаточно проблем, чтобы быть уверенными в том, что система готова к приемочному тестированию.

...исправили достаточно? ...быть уверенными?

# **T** | Критерии остановки тестирования

---

Выделяют 5 основных критериев остановки тестирования:

## **Достижение покрытия**

достигли заранее определенных целей покрытия по критерию покрытия.

## **Плотность обнаружения дефектов**

упала ниже заранее определенного уровня.

## **Маржинальная стоимость**

нахождения следующего дефекта превышает ожидаемые затраты от него.

## **Решение команды разработки**

команда достигла соглашения о готовности продукта к выпуску

## **Босс сказал «выпускаем продукт!»**

# Т | Достижение покрытия

---

Покрытие – мера, определяемая отношением величин:

Сколько было протестировано / Сколько может быть протестировано

Может быть определено:

На уровне модульного тестирования (покрытие операторов, покрытие условий, покрытие ветвлений и т.д.)

На уровне интеграционного тестирования (покрытие API функций, покрытие API параметров, комбинаций параметров и т.д.)

На уровне системного тестирования (покрытие требований, покрытие графа потока управления, покрытие синтаксического графа и т.д.)

Примеры. Останавливаем тестирование, если:

Наши тесты покрывают 95% операторов

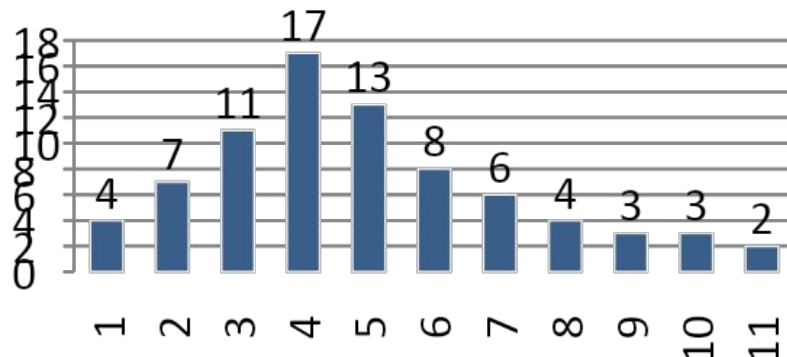
Покрывают все ребра графа потока управления

!!! Проблема: Тестировщики начинают разрабатывать малоэффективные тесты, главное – обеспечить покрытие по критерию. Теряется ориентация на поиск багов.



# Т | Плотность обнаружения дефектов

Через равные промежутки времени (например, каждую неделю) вычисляется количество новых найденных за промежуток времени дефектов. Данные собираются в диаграмму плотности дефектов



Выбирается порог плотности дефектов, ниже которого тестирование останавливается.

На примере тестирование останавливается в конце 9 недели, достигли плотности 3 дефекта в неделю.

!!! Проблема: Могут оставаться критичные дефекты.

!!! Проблема: Тестировщики в отпуске – плотность упала.

# **T** | Маржинальная стоимость

---

## **В экономике**

Затраты на производство одной дополнительной единицы продукции.  
Уменьшается при увеличении количества продукции.

## **В тестировании**

Затраты на обнаружение следующего дефекта.  
Увеличивается для обнаружения каждого следующего дефекта.

Наступает момент, когда маржинальная стоимость дефекта превышает потери компании от выпуска продукта с этим дефектом - пришло время остановить тестирование.

**!!! Критерий пригоден не для всех программных продуктов.**

# **T** | Решение команды разработки

---

На основании различных факторов – технических, финансовых, экономических, «шестого чувства» команда разработки (менеджеры, разработчики, тестировщики, отдел маркетинга и т.д.) решает, что

**выгода от остановки разработки и выпуска продукта**

превышает

**потенциальные обязательства по качеству**

# **Т** | Босс сказал «Выпускаем продукт!»

---

Не бывает идеальных программных продуктов без дефектов.

Экономически бывает выгоднее выпустить продукт раньше, пусть и с некоторым ущербом для качества, чтобы занять нишу рынка.

Задача тестировщика – предоставить максимум информации о возможных рисках и известных проблемах.

Задача отдела продаж – предоставить информацию о финансовой выгоде выпуска сегодня.

Босс взвешивает «за» и «против» и самостоятельно принимает решение о выпуске, неся затем за него ответственность.



# ТЕХНОПОЛИС |

## Лекция 2.5

## Тестовая документация

В соответствии с процессами или методологиями разработки ПО, во время проведения тестирования создаётся и используется определённое количество тестовых артефактов (документы, модели и т.д.)





—  
План тестирования (Test plan) - это главный документ описывающий весь объём работ по тестированию, начиная с описания объекта, стратегии, расписания, критериев начала и окончания тестирования

# **T** | Тест план

---

**Что надо тестировать?** - описание объекта тестирования: системы, приложения, оборудования

**Что будете тестировать?** - список функций и описание тестируемой системы и её компонент в отдельности

**Как будете тестировать?** - стратегия тестирования, а именно: виды тестирования и их применение по отношению к объекту тестирования

**Когда будете тестировать?** - последовательность проведения работ: подготовка (Test Preparation), тестирование (Testing), анализ результатов (Test Result Analysis) и разрезе запланированных фаз разработки

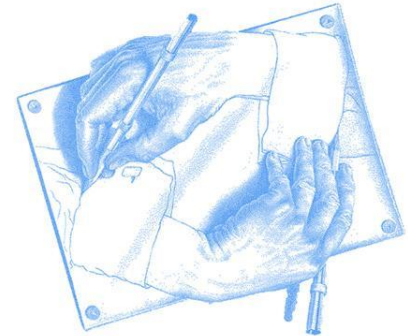
**Критерии начала тестирования** - готовность тестовой платформы (тестового стенда), законченность разработки требуемого функционала, наличие всей необходимой документации.

**Критерии окончания тестирования:**

Результаты тестирования удовлетворяют критериям качества продукта



1. Введение
2. Объекты тестирования
3. Функциональности для тестирования
4. Функциональности которые не будут тестироваться
5. Стратегия тестирования (виды, подходы, методы)
6. Критерии успешности тестирования
7. Критерии остановки и возобновления тестирования
8. Тестовые результаты
9. Тестовое окружение
10. Ответственность



**Чек-лист - это документ, описывающий что должно быть протестировано**

## **Зачем нужен чек лист?**

Не забыть требуемые тесты

Для деления задач по уровню квалификации

Для сохранения отчётности и результатов тестирования

## **Что может(должно) быть в чек-листе?**

Номер

Список проверок(с требуемой степенью детализации)

Статус проверки(сборка, окружение, тестирующий)

Приоритет

Результат

# Т | Чек лист. Пример

Проверка	Результат		
	Win XP	XP SP4	Win Vista
Операции с файлами	ok	ok	ok
Создание файла	ok	ok	ok
Открытие файла	ok	ok	ok
Сохранение документа	ok	ok	ok
Печать	ok	ok	ok
Редактирование файлов	bugs	bugs	bugs
Отмена	ok	ok	ok
Копирование	ok	ok	ok
Вырезание	<a href="#">bug #146</a>	<a href="#">bug #146</a>	<a href="#">bug #146</a>
Вставка	ok	ok	ok
Удаление	ok	ok	ok
Поиск	<a href="#">bug #123</a>	<a href="#">bug #133</a>	ok
Поиск с заменой	<a href="#">bug #126</a>	ok	ok

**Тестовые данные (Test data)** - данные которые используются для тестирования

Пример:

410039303350 - счёт заблокирован (зачисления запрещены)

4100322407607 - корректный счёт (зачисление успешно пройдёт)

[test1@dd.com](#) - логин

123 - пароль

**Тестовый случай (Test Case)** - это документ, описывающий совокупность шагов, конкретный условий и параметров, необходимых для проверки реализации тестируемой функции или её части.



# Т | Пример тест кейса

## Атрибуты тест- кейса

1. Номер (id)
2. Название (Summary/Name)
3. Предусловие (PreCondition)
4. Шаги тест кейса и описание (Steps and Description)
5. Ожидаемы результат (Excepted result)
6. Пост-условие (Post Condition)
7. Автор (Designer)
8. Статус (Status)
9. Дата создания (Creator)

<b>Test Name:</b>		
<b>Status:</b>		
<b>Created Date:</b>		
<b>Designer:</b>		
<b>Pre Conditions:</b>		
<b>Steps</b>	<b>Description</b>	<b>Expected Result</b>
Step 1		
Step 2		
Step 3		
Step 4		
<b>Post Conditions:</b>		

## **Набор тестов(тест комплект) (test suite) -**

Это набор тест кейсов, которые объединены тем что относятся к одному тестируемому модулю, функциональности, приоритету или одному типу тестирования

# T | Матрица прослеживаемости

**Матрица прослеживаемости (Traceability Matrix)** - Это таблица, где

вертикальные колонки - это требования, а горизонтальные - тест-кейсы (или наоборот, как удобно).

Помогает посмотреть

Берём первый тест

пересечении ставим

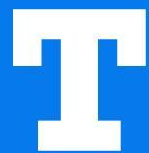
Requirement Traceability Matrix											
Test Case ID	TC_1	TC_2	TC_3	TC_4	TC_5	TC_6	TC_7	TC_8	TC_9	TC_10	# Test Cases for respective Requirement
Req_ID											
Req_1	X		X			X					3
Req_2		X			X						2
Req_3			X								1
Req_4				X		X					2
Req_5					X		X				2
Req_6						X					1
Req_7					X		X				2
Req_8								X			1
Req_9									X		1
Req_10										X	1

ыми тест-кейсами

зкрывает и на

ждым тест-кейсом.



A large, bold, white letter 'T' is positioned on the left side of the slide. To its right is a vertical red bar of the same height as the letter.

Спасибо за внимание!