ТЕХНОЛОГИЯ И ПРОЦЕСС РАЗРАБОТКИ ПО (Л-5)

к.п.н., доцент Касаткин Д. А.
e-mail: kasatkinda@cfuv.ru

Ежедневная сборка (build) и непрерывная интеграция

- Интеграция программного проекта означает: взять все созданные компоненты проекта, скомпилировать их совместно и выполнить тесты (регрессионный набор).
- Наиболее заметным продвижением в этом направлении стала "ежедневная сборка", введенная в Майкрософт в восьмидесятые годы. Идея была проста. В конце каждого дня собираются все изменения, "введенные" (официально подписанные) разработчиками; система компилируется, прогоняются все тесты. Как отмечает менеджер Майкрософт [Cusumano 1995]:
- Создание ежедневных сборок – одно из самых болезненных дел в мире, но это и самая величайшая вещь в мире, по-скольку вы получаете мгновенную обратную связь.
- Правила agile, в частности XP, идут дальше и вместо "ежедневной сборки" рекомендуют "непрерывную интеграцию". Правило Бека [Beck 2005]: Интегрируйте и тестируйте изменения

Парное программирование

- Споры, главным образом, были следствием настаивания XP, что парное программирование является единственным и универсальным способом разработки программ. Бек писал:

«Разрабатывайте все производственные программы двумя людьми, сидящими за одной машиной.»

Повышение дисциплины

- Программисты в паре чаще «делают то, что нужно» и реже устраивают длинные перерывы.
- Лучший код
- Партнёры в паре менее склонны к неудачным решениям и производят более качественный код.
- Высокий боевой дух
- Коллективное владение кодом

(пары меняются)

Недостатки

А че он все время смотрит?

Он меня напрягает!

Ты думаешь, я сам не справлюсь?!

В одиночку я сделаю быстрее

А-а-а-аргх!

искусство

РАБОТАТЬ В ПАРЕ

Создаем эффективную пару

[новичок]

[эксперт]

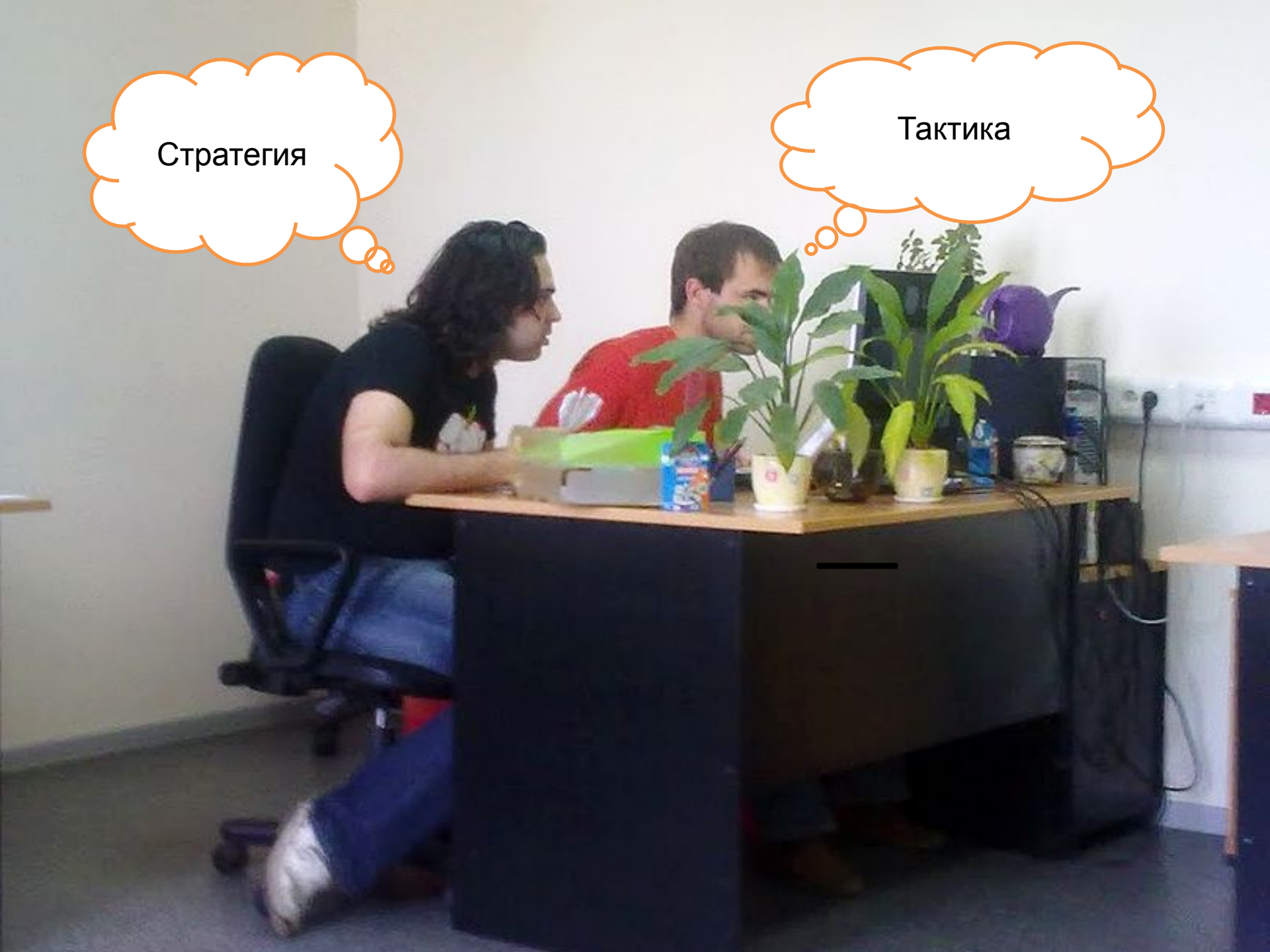
[эксперт]

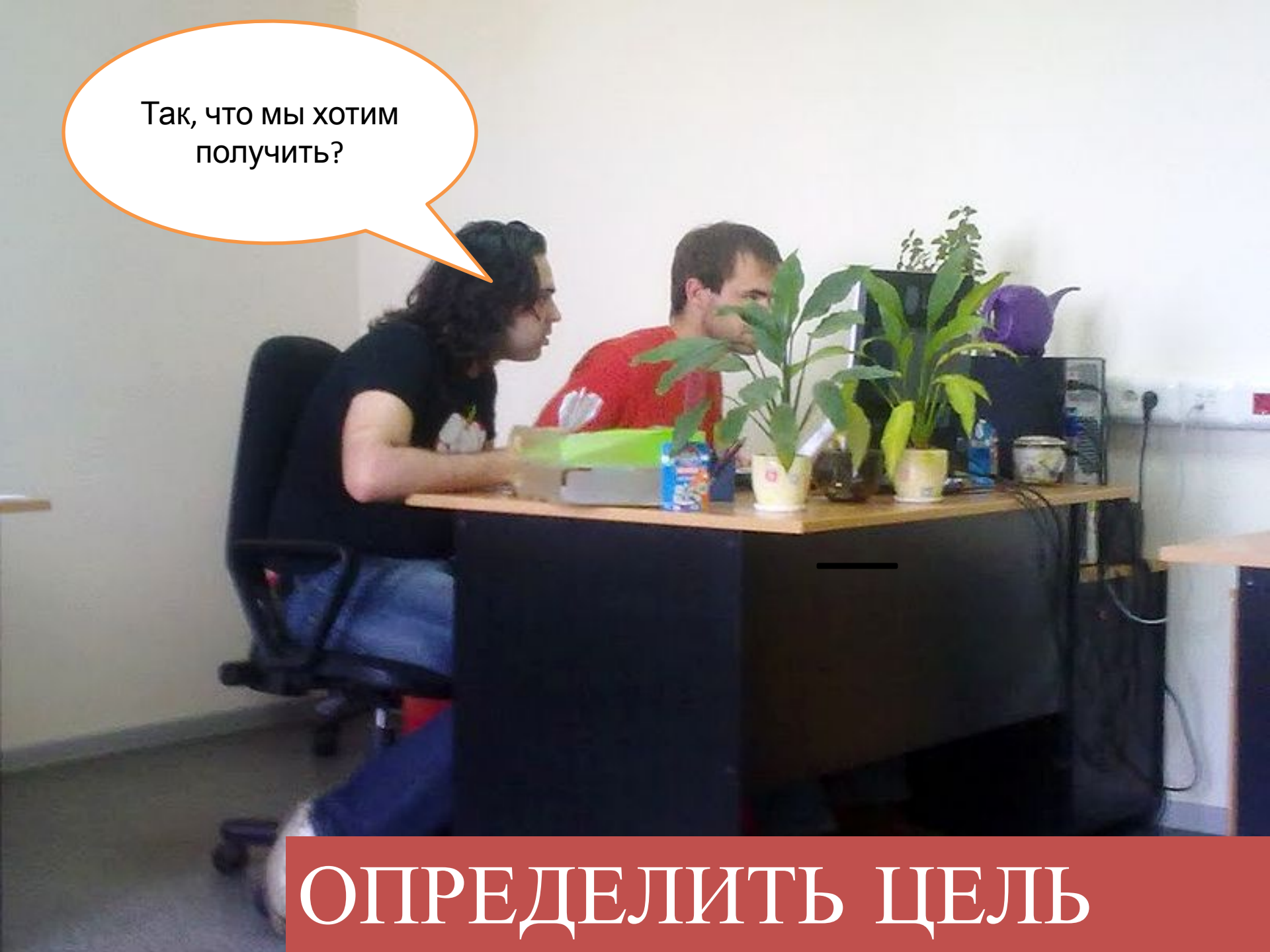
[эксперт]

[новичок]

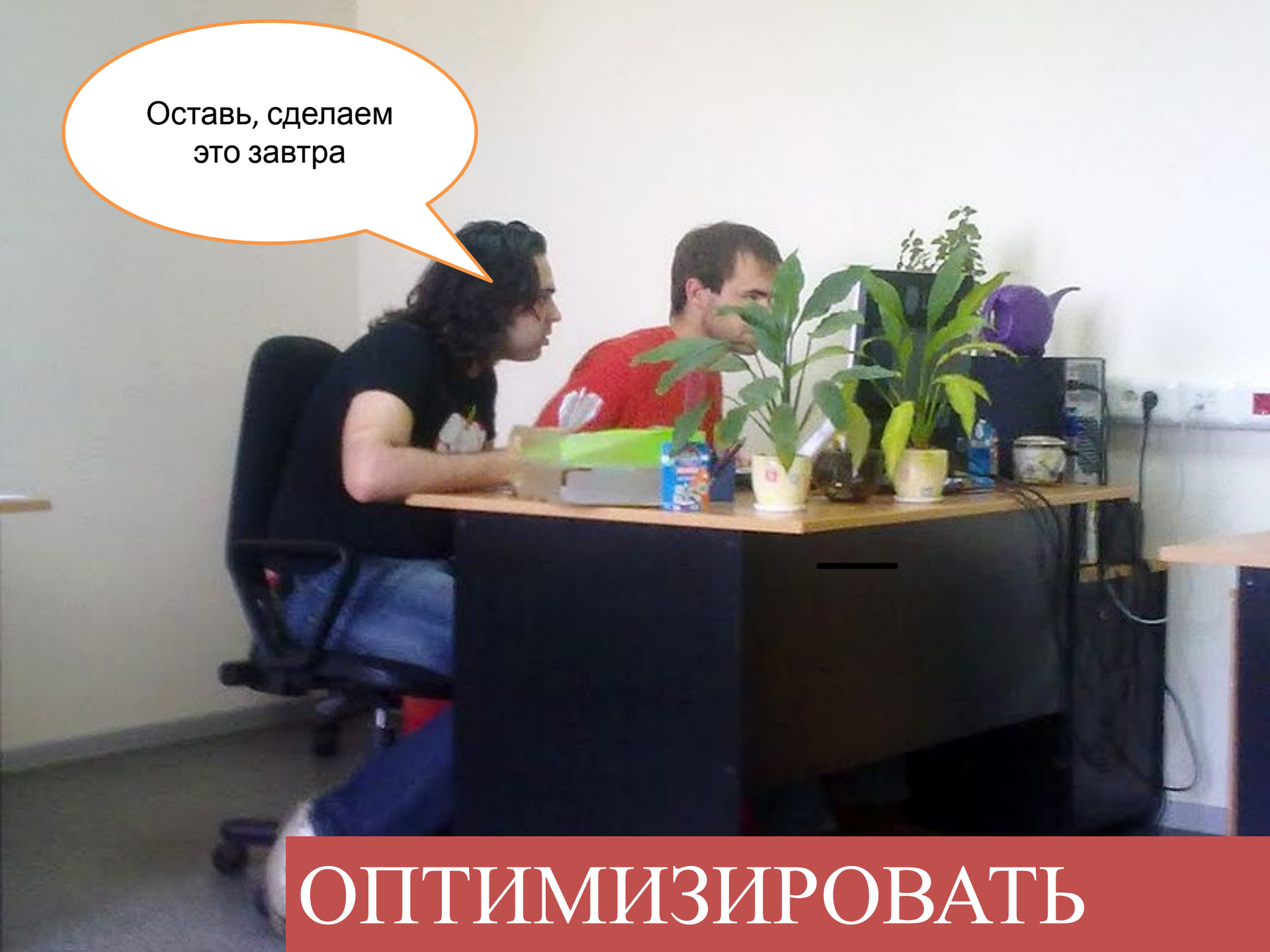
[новичок]

driver
navigator
Один компьютер на двоих


Стратегия
Тактика



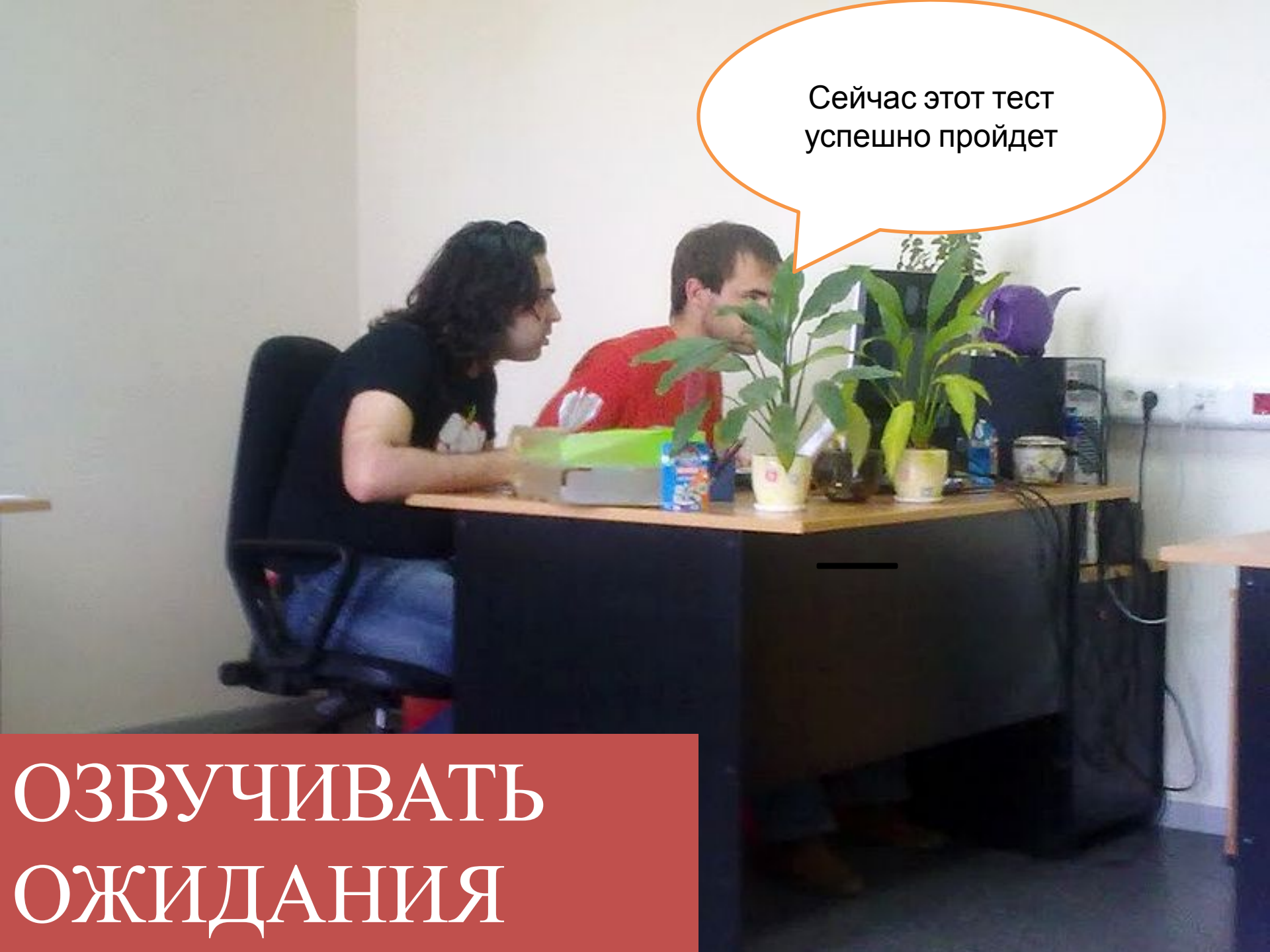
ОПРЕДЕЛИТЬ ЦЕЛЬ



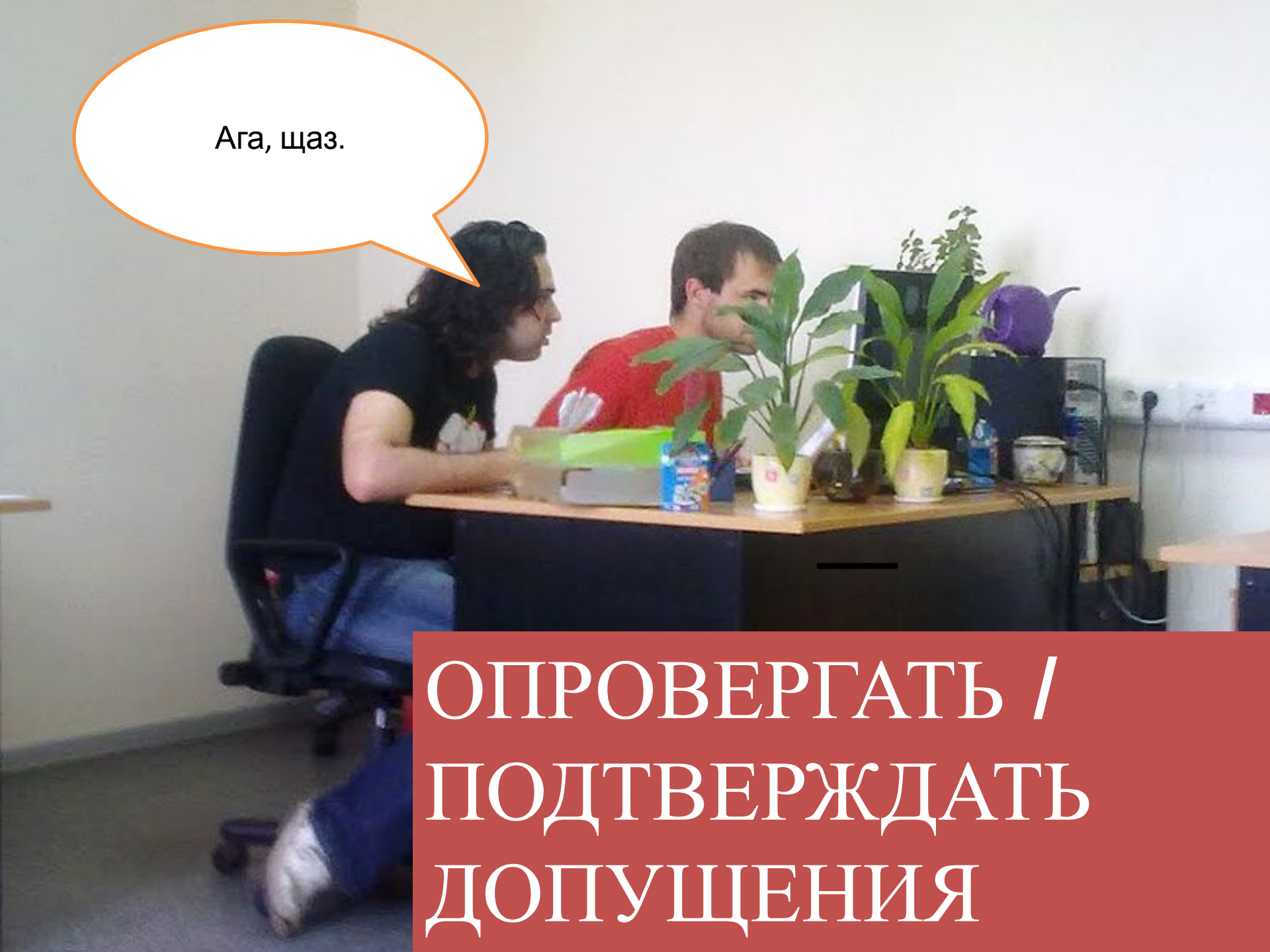
ОПТИМИЗИРОВАТЬ

ДУМАТЬ ВСЛУХ

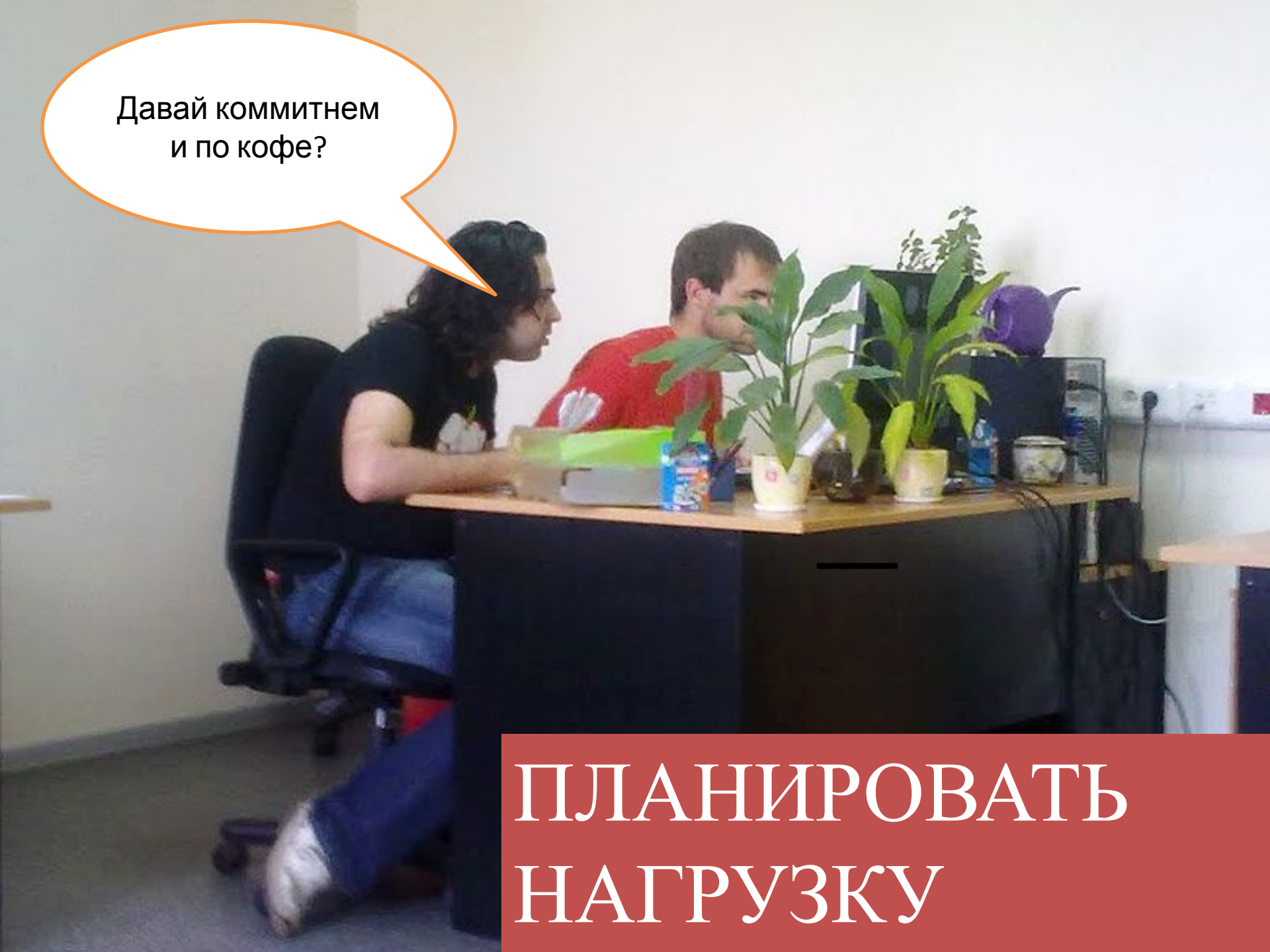
ТРЕБОВАТЬ АРГУМЕНТЫ



ОЗВУЧИВАТЬ ОЖИДАНИЯ



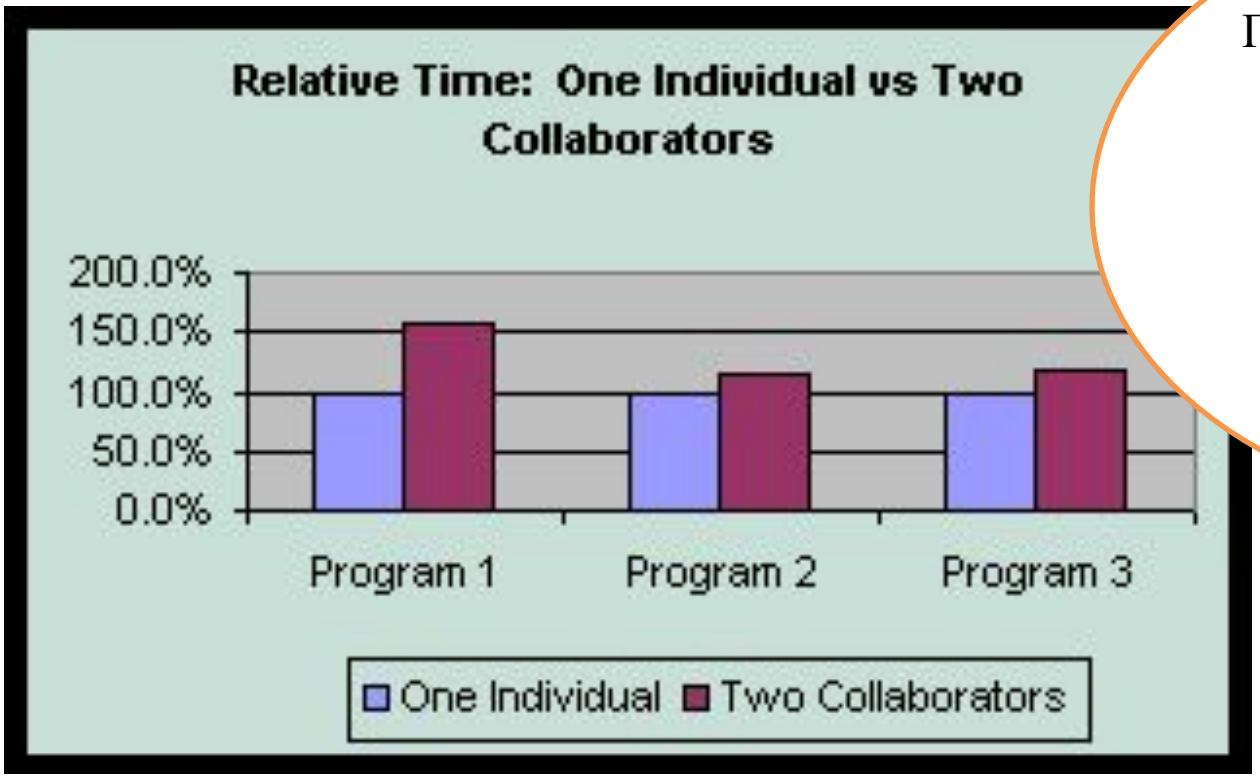
ОПРОВЕРГАТЬ / ПОДТВЕРЖДАТЬ ДОПУЩЕНИЯ



ПЛАНИРОВАТЬ НАГРУЗКУ

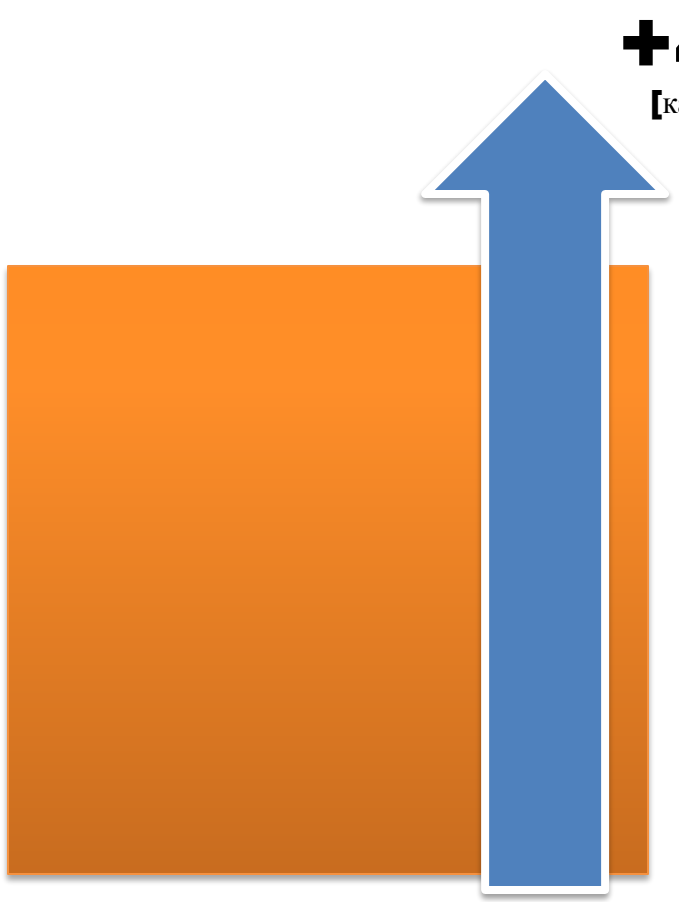
убедительные

ЦИФРЫ

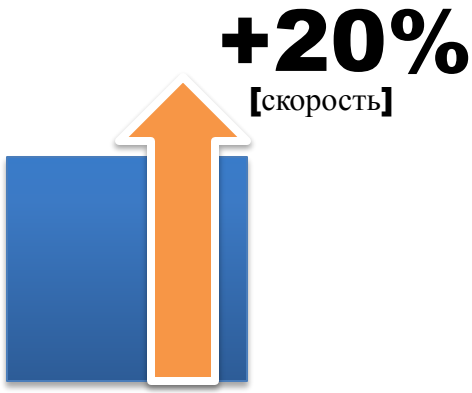


Программисты, работающие в паре, всего на **15%** медленнее двух одиночек, но производят несравнимо более качественный код

*Cockburn, Williams The Costs and Benefits of Pair Programming (2000)



[БОЛЬШОЙ СЛОЖНЫЙ ПРОЕКТ]



[МАЛЕНЬКИЙ ПРОСТОЙ ПРОЕКТ]

*Arisholm. Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise (2007)

РАБОТА

ПРИНОСИТ

БОЛЬШЕ РАДОСТИ!

*Cockburn, Williams The Costs and Benefits of Pair Programming (2000)

Пинг-понг программирование

PING-PONG

STYLE

Стандарты кодирования

- Каждая уважающая себя организация устанавливает точные правила стиля программирования.
- помимо необходимости иметь стандарты кодирования, так это замечание, что никто не может определить авторство кода, предполагающее "безликое программирование"

Концепция рефакторинга

- Не каждому образцу программных изменений соответствует паттерн рефакторинга. Необходимо выполнение двух условий:
- рефакторинг **не должен изменять семантику** программы;
- рефакторинг **должен улучшать качество** кода или архитектуры.

Цели и причины рефакторинга

- Рефакторинг нужно применять постоянно при разработке кода. Основными стимулами его проведения являются следующие задачи:

1. необходимо добавить новую функцию, которая недостаточно укладывается в принятое архитектурное решение;
2. необходимо исправить ошибку, причины возникновения которой сразу не ясны;
3. преодоление трудностей в командной разработке, которые обусловлены сложной логикой программы.

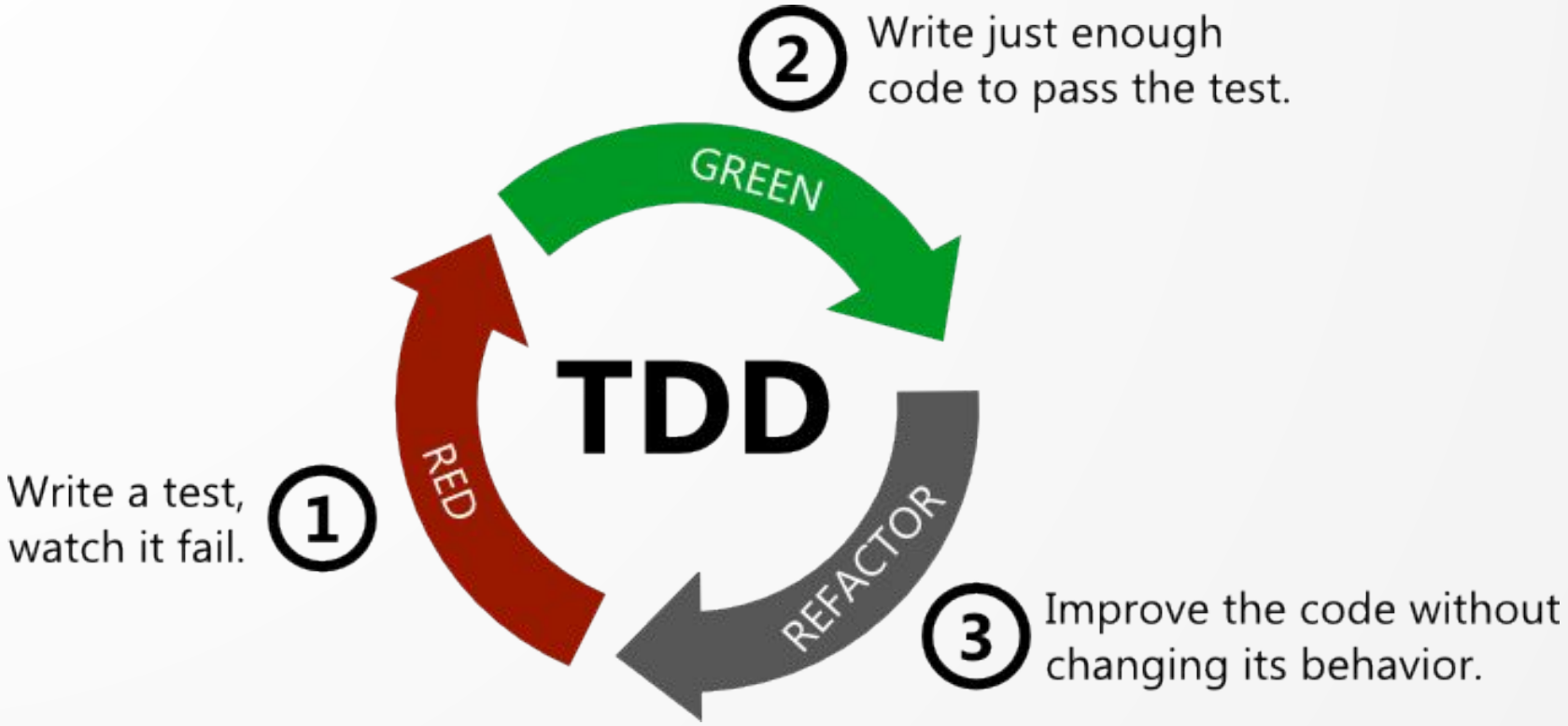
Признаки плохого кода

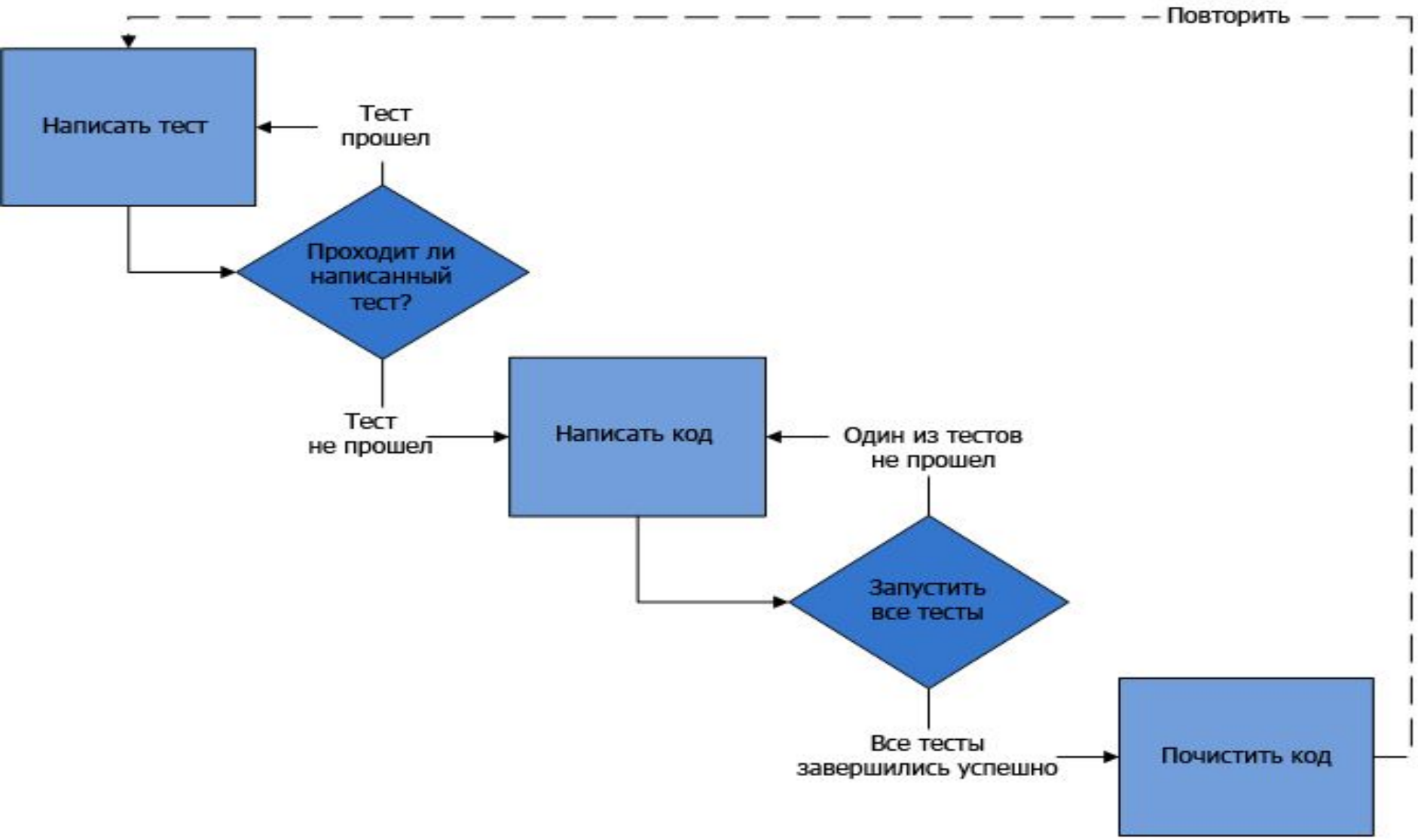
- дублирование кода
- длинный метод;
- большой класс;
- длинный список параметров;
- «жадные» функции — это метод, который чрезмерно обращается к данным другого объекта;
- классы данных;
- комментарии

Разработка "Вначале Тест" и разработка, управляемая тестами

- TDD (Test Driven Development). Разработка TDD является следствием
- TFD (Test First Development) – "Вначале Тест" разработки.

Цикл TDD


2
Write just enough
code to pass the test.
GREEN
TDD
RED
REFACTOR
Write a test,
watch it fail.
1
3
Improve the code without
changing its behavior.


Повторить
Написать тест
Тест прошел
Проходит ли написанный тест?
Тест не прошел
Написать код
Один из тестов не прошел
Запустить все тесты
Все тесты завершились успешно
Почистить код

TDD метод программной разработки

определяет TDD как повторение следующего основного цикла:

TDD цикл:

1. Быстро добавить тест.
2. Выполнить все тесты и увидеть, что новый тест "падает".
3. Выполнить небольшое изменение системы.
4. Убедиться, что все тесты проходят.
5. Выполнить рефакторинг, удаляя

Оценка TFD и TDD

- TDD — это процесс итеративного, непрерывного, параллельного написания тестов и рабочего кода, с обязательными фазами рефакторинга.
- **каждый новый код должен сопровождаться новыми тестами". Не так уж критично, что раньше – код или тест, никогда не создавайте одно без другого.**

TDD за и против
Зависимоть от ТЗ

Плюсы:

- **Упрощение поддержки кода**
- **Четкая модульная структура программы**
- **Борьба со сложностью**

Минусы:

- **Увеличение времени разработки программного продукта**

КНИГИ

- Вот список книг, которые любой TDD-практик просто обязан прочитать (must read) и иметь в любой момент на своем столе:
- Э. Гамма и др. «Design patterns»
- М. Фаулер «Refactoring: Improving the Design of Existing Code»
- М. Фаулер «Patterns of Enterprise Applications Architecture»
- Р. Мартин «Agile software development»

BEHAVIOR-DRIVEN DEVELOPMENT

- BDD (Behavior-driven development, Разработка через поведение) - техника разработки, при котором рассматривается не результат выполнения какого-либо модуля, а та работка, которую он выполняет. Этот принцип можно рассматривать как продолжение TDD.

Создателем техники считается ДенНорт

Принципы BDD

- Тестируемая разработка - это методология разработки программного обеспечения, которая по существу утверждает, что для каждой единицы программного обеспечения разработчик программного обеспечения должен:
- *Сначала* определить тестовый набор для устройства;
- Сделать тесты неудачными;
- Затем реализовать устройство;
- Наконец, убедитесь, что реализация блока делает тесты успешными.

Отличие TDD от BDD

- **This class *should* do something**
- Используйте слово «поведение», а не
- «тест»
- BDD дает «доступный всем язык» для анализа

Общеупотребительный язы

- Для того, заказчик и разработчик могли составлять сценарии вместе, используется концепция общеупотребительных языков

(ubiquitous language)

- Общеупотребительный язык – Набор базовых терминов предметной области. Является общим для заказчика и разработчика.

Системы для программной поддержки TDD и BDD

- JUnit – фреймворк, применяющийся для разработки на Java. В нем тестовые методы начинаются со слова test и наследуются от тест-класса TestCase.
- Nunit – открытая среда юнит-тестирования приложений для .NET. Она была портирована с языка Java (библиотека JUnit). Первые версии NUnit были написаны на J#, но затем весь код был переписан на C# с использованием таких новшеств .NET, как атрибуты.

Системы для программной поддержки TDD и BDD

- Cucumber - среда разработки наязыке программирования Ruby. Разработчик описывает необходимое поведение в обычном тексте.
- Specflow
- BDD-синтаксис Given, When, Then интуитивно понятен. Рассмотрим элементы синтаксиса:
- Given предоставляет контекст выполнения сценария тестирования, например, точки вызова сценария в приложении, а также любые необходимые данные.
- When определяет набор операций, инициирующих тестирование, таких как действия пользователей или подсистем.
- Then описывает ожидаемый результат

Пример разработки системы с использованием BDD

- Начнем с того, что определим нашу функциональность.
- Feature: Show logged in user name
- In order to logged in as a user called “Username"
- I want to see page header displays the caption
- "Здравствуйте, Username!"
- Здесь мы задаем на понятном нам языке, что мы хотим увидеть от нашей функциональности.

Особенности BDD

- BDD интересно тем, что тесты к нему пишутся с помощью сценариев.
- Сценарии – описание функциональности метода, написанное на естественном языке по определенному шаблону.

Написание сценария

- Напишем сценарий, который будет основой для работы cucumber’a
- Scenario: Show logged in user name
- Given I am logged in as a user called “Username"
- When I visit the homepage
- Then the page header displays the caption "Здравствуйте, Username!“
- Scenario – имя сценария.
- Given… - Начальное условие (две категории и их описание)
- When.. – Если я на странице с категориями…
- Then – Я должен увидеть…

Написание сценария

- Для каждого действия также пишем соответствующие функции:
- Given /I am logged in as a user called "(.*)"/ do |name| create_user(name) sign_in_as(name) end
- Then /the page header displays the caption "(.*)"/ do |caption|page_header.should_contain(caption) end
- Таким образом Cucumber или SpecFlow сможет интерпретировать каждый шаг, вычленить с помощью регулярных выражений параметры и запустить соответствующие тесты.