Концепция непрерывной интеграции ПО (Continuous integration)

Выполнила Шульга Е.В., ВМИ-301

Что это?

- **Непрерывная интеграция** — это практика разработки программного обеспечения, которая **заключается в выполнении частых автоматизированных сборок проекта** для скорейшего выявления и решения интеграционных проблем.

Build -> Deploy -> Test

Зачем это нужно?

Интеграция в большом проекте

Зачем это нужно?

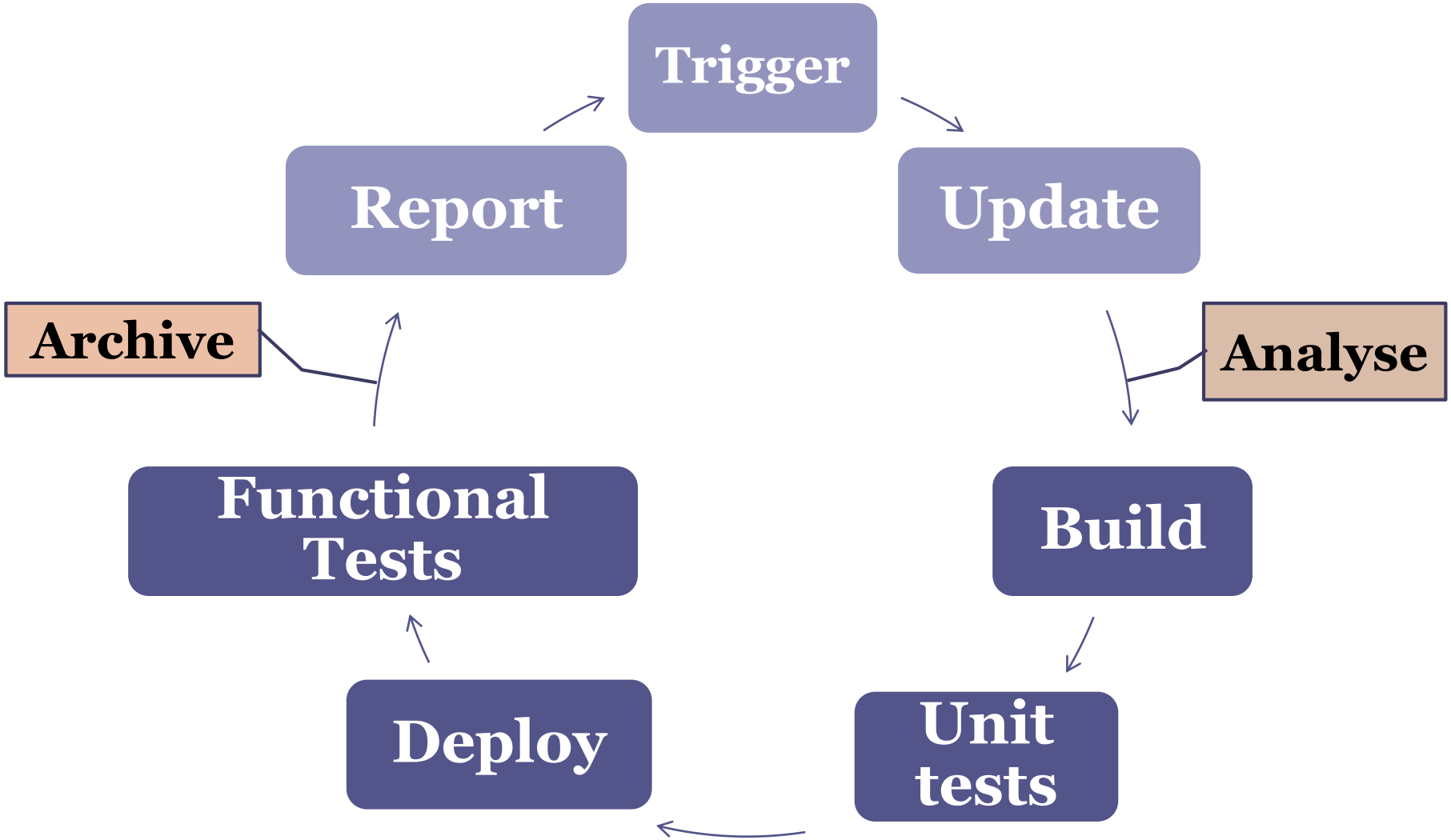
- **Скорейшее обнаружение ошибок** (в течение суток или нескольких часов после коммита) делает процесс их устранения очень быстрым и «дешевым». Обычно болезненный процесс интеграции перестает растягиваться на месяца и становиться кошмаром программистов.

Зачем это нужно?

- На руках есть работоспособная версия проекта со всеми нововведениями. Заказчик день за днем наблюдает развитие проекта — заказчик доволен. Программист быстро получает отзывы и замечания — программист тоже доволен.

- Психологический феномен **«синдром разбитых окон»**: сложнее находить и исправлять ошибки, когда их много

Build-Deploy-Test: подробнее



Trigger

- Событие, с которого начинается цикл интеграции
  - **изменение в системе контроля версий (по коммиту)**
  - изменение в файловой системе
  - определенный момент времени **(ночная сборка)**
  - сборка другого проекта
  - ручной запуск
  - изменение на веб-сервере

Update

- Обновление копии проекта на удаленном сервере, где будет осуществляться сборка.

Analyse *«фатальный» этап*

Ошибка на данном этапе означает сбой сборки

- Анализ кода
  - наличие типичных ошибок
  - соответствие принятым стандартам кодирования
  - глубина наследования
  - связанность классов и пр.

- Этап является необязательным, но позволяет собирать информацию о метриках кода.

Build *«фатальный» этап*

Ошибка на данном этапе означает сбой сборки

- Компиляция (трансляция) исходных кодов
- Осуществляется на специально выделенном интеграционном сервере
- Результат только этой сборки можно считать конечным. Больше никаких «Проект собирается на моей машине!». Есть только одно место, где проект может собираться — это интеграционный сервер.

Unit tests *«фатальный» этап*

Ошибка на данном этапе означает сбой сборки

- Неотъемлемая часть разработки в методологии экстремального программирования!
- «Маленькие» тесты, выполняются очень быстро (менее 1 мс), не требуя операции ввода/вывода, обращение к БД и пр.
- Цель — проверить «условную логику» в коде: if-ы, циклы и пр. <= *места, где рождается большинство багов*
- Позволяют быстро локализовать поломку
- Дополнительную информацию можно извлечь, измеряя покрытие модульных тестов.

Deploy

- «Развертывание» проекта.
- В случае веб-приложения это выкладывание на веб-сервер (сервер приложений) и запуск. Для GUI приложений это (пере)установка в системе.
- Этап развертывания должен проходить как можно более «чисто». Необходимо привести приложение в некое «стандартное» состояние:
  - «залить» дамп базы
  - настроить в стандартном режиме
  - убрать следы предыдущей деятельности приложения

Ошибка на данном этапе означает сбой сборки

Functional tests *«фатальный» этап*

- Прогон через автоматические функциональные тесты (регрессионное тестирование).
- Контроль взаимодействия классов
- После прохождения регрессионных тестов можно считать, что интеграция прошла успешно и в проект не внесено правок, которые могут привести к его неработоспособности.
- В противном случае интеграция неуспешна: код содержит ошибки и требуется его исправление/доработка.

Archive

- После того, как достигнута максимальная уверенность в качестве исходного кода необходимо сохранить его (можно использовать систему контроля версий) .
- Также необходимо сохранить бинарные файлы проекта. Они могут понадобиться, если нужно будет воспроизвести ошибку в конкретной версии и для ручного тестирования.

Report

- Этап генерации и публикации отчетов.
  Отчеты включают в себя следующее:

  - причина сборки (изменения в репозитории, сборка по расписанию и пр.)
  - изменения в исходных кодах (от последней сборки или от последней успешной сборки)
  - отчеты по анализу кода
  - лог сборки
  - лог модульных тестов (какие тесты прошли и, что важнее, какие не прошли)

Report

- лог регрессионных тестов (аналогично модульным тестам)
- статистика сборок проекта:
  - общее число удачных/провальных сборок
  - распределение удачных/провальных сборок во времени
  - статистика результатов статического анализа кода
- прочие метрики для менеджера проекта

Report

- Механизм публикации отчета (рекомендуется сочетать несколько)
  - IRC или jabber бот
  - рассылка по смс и электронной почте
  - публикация на web или ftp сервере
  - специализированные клиенты, позволяющие узнать статус сборки.

Основы успешного применения практики Continuous integration

Во-первых: система контроля версий

- Наиважнейшая часть — надежная система контроля версий. Здесь хранится **всё**.
- На вопрос «Где найти такой-то файл?» должен быть один ответ: «**В репозитории**»
- Кроме исходного кода там должны лежать тестовые скрипты, файлы настроек, схемы баз данных, установочные скрипты и сторонние библиотеки.
- **Философия**: возможность сесть за чистую машину, сделать checkout и полностью собрать и запустить проект.

Во-вторых…

Все сохраняют изменения в репозитории каждый день

- Ключ к быстрому решению проблем – в их быстром обнаружении. Если разработчики заносят свои изменения каждые несколько часов, конфликт обнаруживается быстро и так же быстро разрешается, так как к этому моменту версии кода еще не сильно разошлись.
- Конфликты, которые не обнаруживаются в течение недель, разрешаются очень сложно.

В-третьих…

Тестирование производится в копии среды реальной эксплуатации

- Настроить среду тестирования настолько близко к среде работы приложения, насколько это возможно.
- Те же самые СУБД тех же версий. Та же версия операционной системы. Тот же самый IP-адрес и порты. То же оборудование.
- Положите все необходимые библиотеки из рабочей среды в тестовую, даже если система их не использует.

Тесты — узкое место

- Тесты, особенно функциональные, часто проходят очень долго
- Можно разбить сборку проекта на две части: первая (с unit-тестами) запускается каждый раз при commit'е, вторая с интеграционными тестами — раз в час/сутки
- Продолжать разработку уже после прохождения unit-тестов

А как же branches?

- Мартин Фаулер не рекомендует «увлекаться» разбиением процесса разработки на ветви, утверждая, что это часто приводит к путанице и неудобствам. Коммиты рекомендуется вносить сразу в master (mainstream, trunk и пр. в некоторых VCS)
- Ветви удобны для внесения крупных изменений: когда есть код, разработка которого до финального варианта займет несколько дней, и при этом желательно делать более частые коммиты в репозиторий.

В-четвертых:

Каждое изменение в репозитории должно включаться в сборку на интеграционном сервере

~~«У **меня** всё работает!»~~

Работать должно на специально выделенном сервере — в условиях, максимально приближенным к условиям эксплуатации приложения

Системы поддержки непрерывной интеграции

Популярные:

- Cruise Control
- TeamCity
- Hudson (Jenkins)
- TFS
- Atlassian Bamboo

- Серверное ПО от компании JetBrains для автоматизации процесса непрерывной интеграции
- Написано на Java
- Проприетарная лицензия: бесплатная для небольших команд и открытых проектов; коммерческая для больших команд разработчиков

TeamCity: возможности

- Предварительное тестирование кода перед коммитом.
  Предотвращает возможность коммита программного кода, содержащего ошибки, нарушающего нормальную сборку проекта, путём удалённой сборки изменений перед коммитом.
- Грид-сборка проекта.
  Предоставляет возможность производить несколько сборок проекта одновременно, производя тестирование на разных платформах и в различном программном окружении.
- Интеграция с системами оценки покрытия кода, инспекции кода и поиска дублирования кода.

TeamCity: возможности

- Интеграция с различными средами разработки: Eclipse, IntelliJ IDEA, Visual Studio.
- Поддержка различных платформ: Java, PHP, .NET и Ruby
- Мгновенные уведомления об ошибках сборки. Вам не нужно дожидаться окончания сборки, чтобы узнать о проблемах компиляции или упавших тестах
- Возможность запускать сборку и тестирование измененного кода без коммита в систему контроля версий, прямо из IDE

TeamCity: возможности

- Управление общими ресурсами, позволяющее без проблем ограничивать доступ к совместно используемым базам данных, тестовым устройствам и т.п.
- Конфигурируемые условия падения сборки на основе множества метрик (число упавших тестов, число непокрытых классов и модулей, а также метрики, исключающие деградацию качества кода)
- Функции по поддержке сервера в хорошей форме: встроенная очистка истории сборок, отчеты о занимаемом дисковом пространстве и отчеты о здоровье сервера

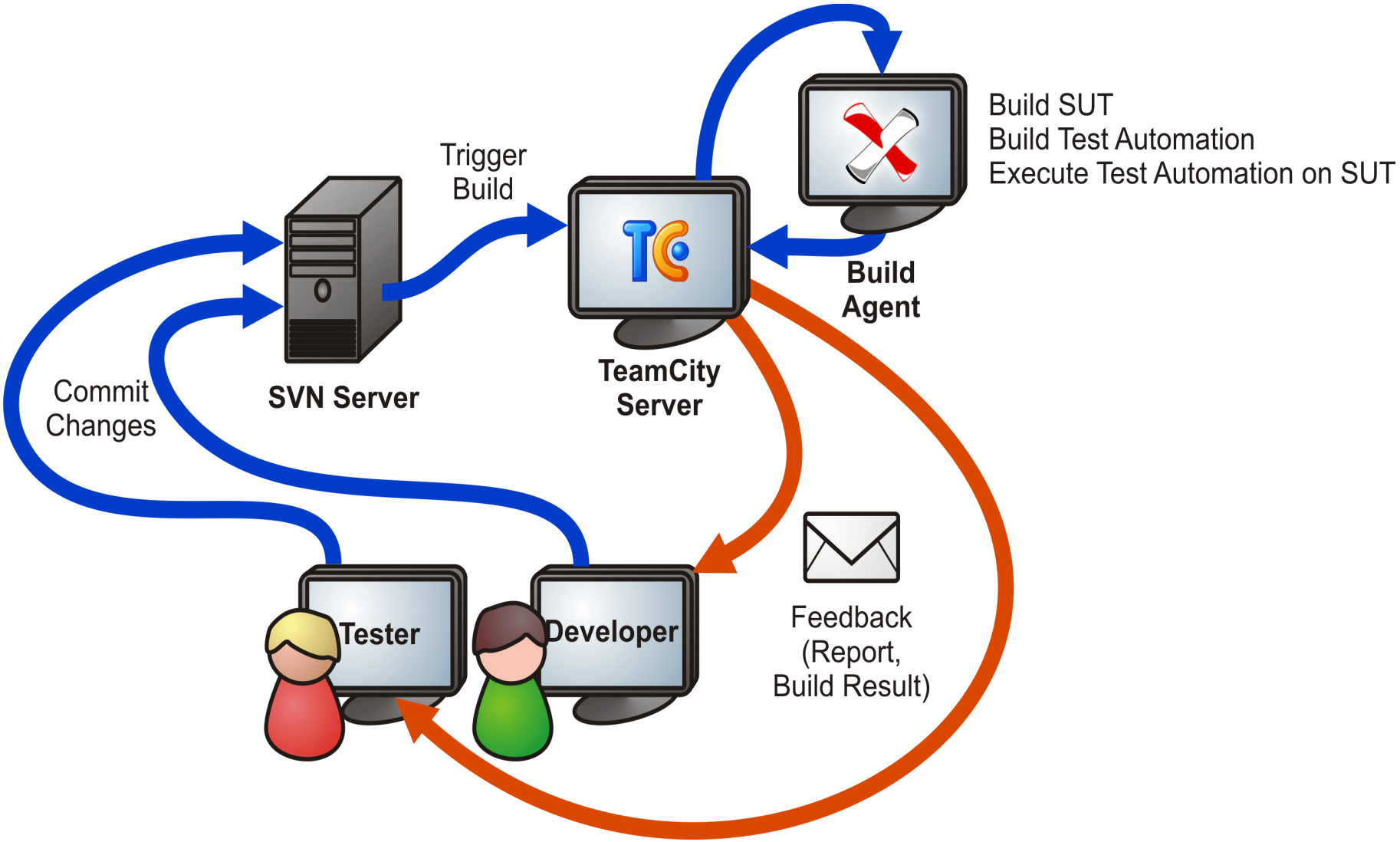
TeamCity: поддержка систем контроля версий

- Git
- Mercurial
- Subversion
- Team Foundation Server (2005, 2008, 2010, 2013)
- Microsoft Visual SourceSafe
- Perforce
- CVS
- Borland StarTeam
- IBM Rational ClearCase (а также UCM)
- SourceGear Vault

TeamCity: архитектура

- На самом деле, «сервер интеграции» и «машина, на которой будет проходить этот процесс», – не обязательно/обычно разные серверы. Более того, машин, на которых запускаются сборки и тесты, может быть много, и все на разных ОС.
- Архитектура TeamCity состоит из сервера интеграции, который управляет процессом, и «фермы» билд-агентов, на которых и производится сборка и тестирование. Один агент может обрабатывать один проект, несколько агентов можно запускать параллельно

TeamCity: архитектура

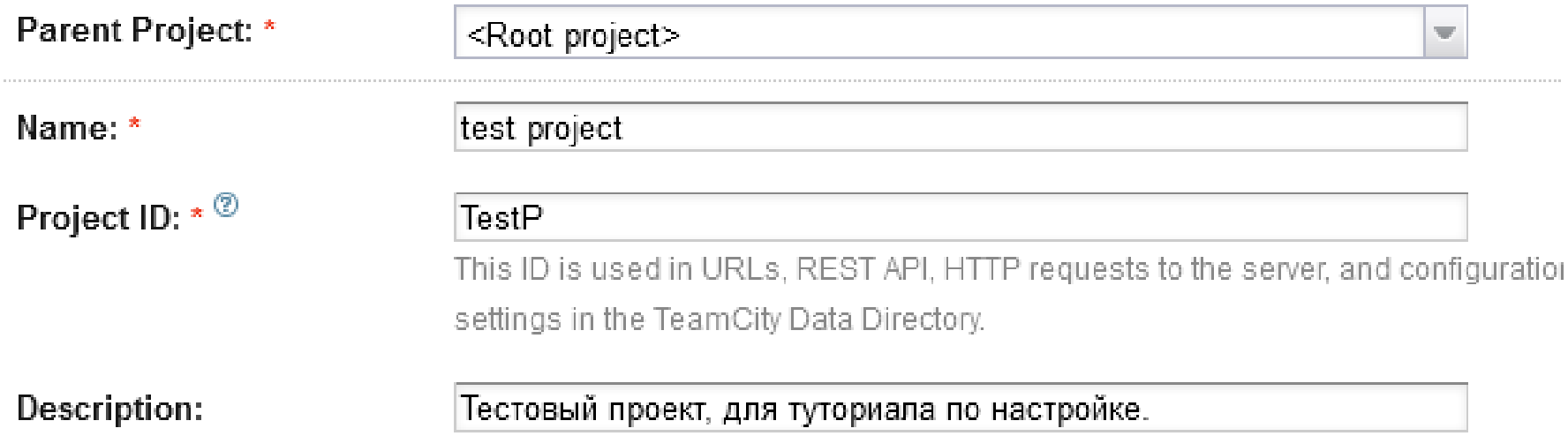


TeamCity: настройка проекта

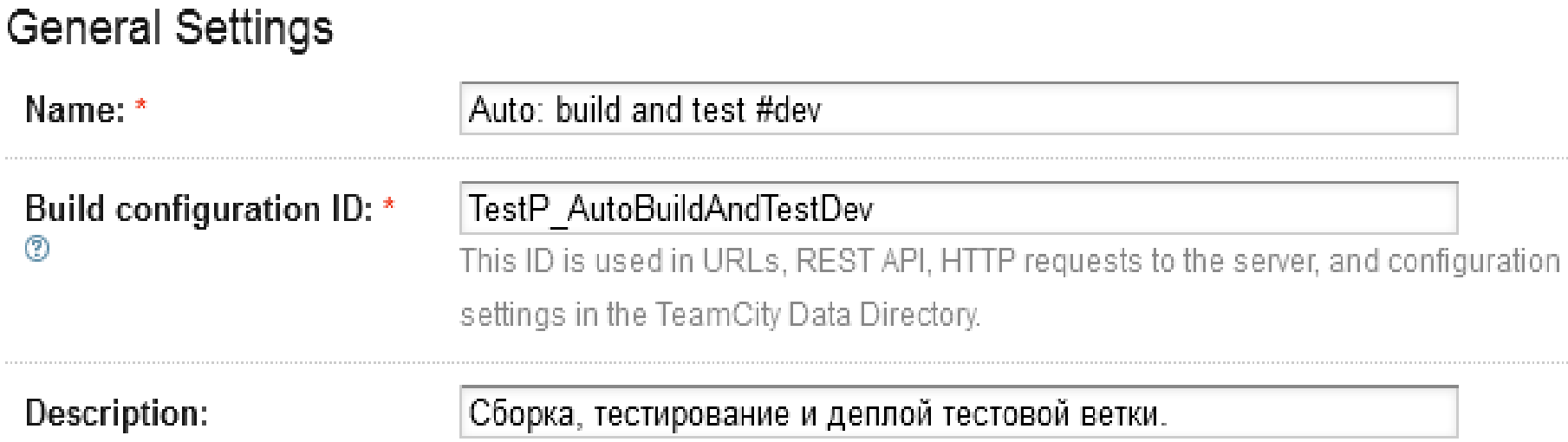
TeamCity: настройка проекта

- Общий сценарий на сервере будет выглядеть так:
  - забрать свежие изменения из репозитория
  - скомпилировать проект;
  - если всё прошло успешно на предыдущем шаге – прогнать юнит-тесты;
  - если всё прошло успешно на предыдущем шаге – прогнать функциональные тесты;
  - если всё прошло успешно на предыдущем шаге – залить изменения на тестовый сервер.

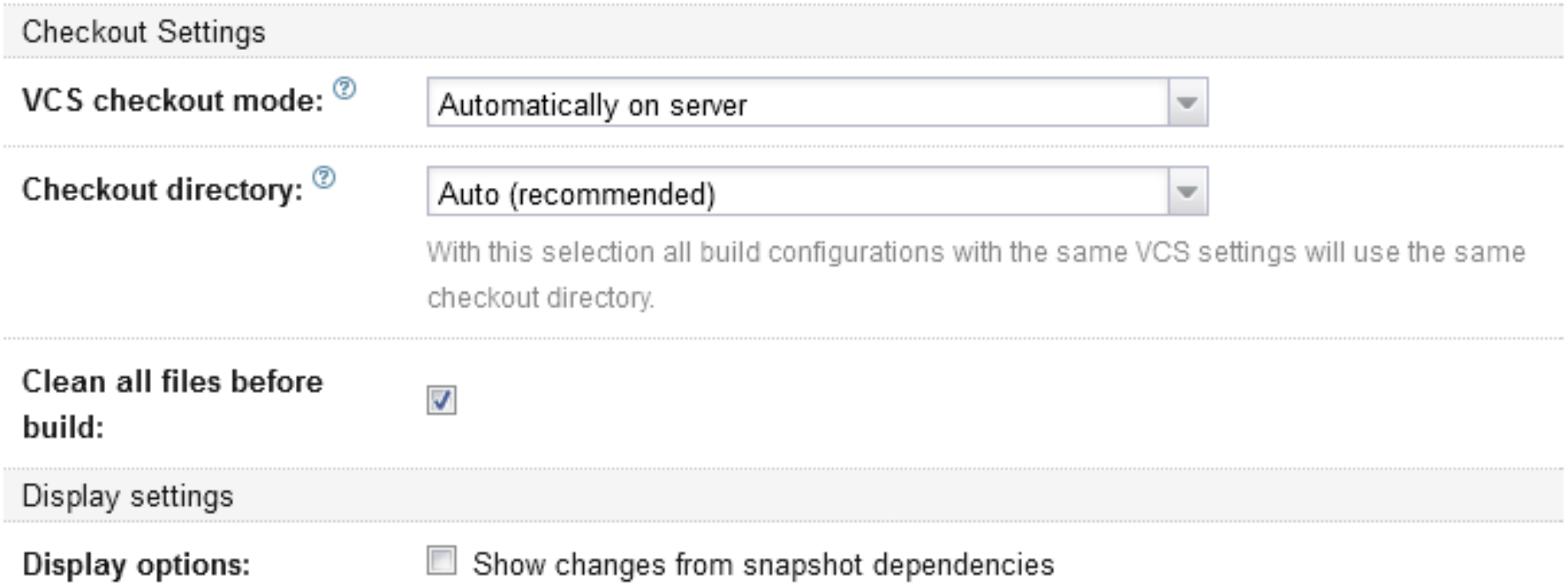
TeamCity: создание проекта



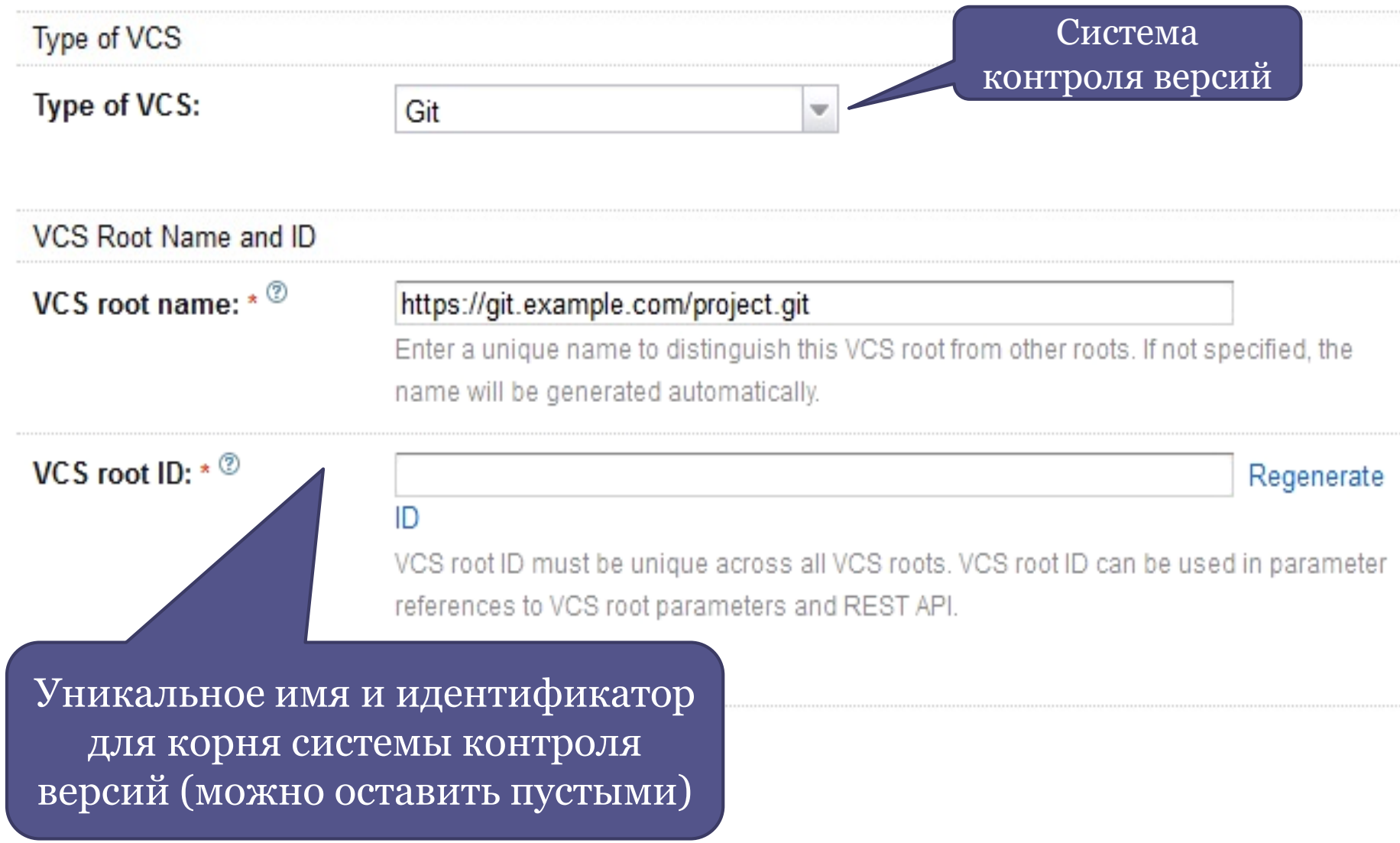
TeamCity: конфигурация сборки



TeamCity: конфигурация сборки



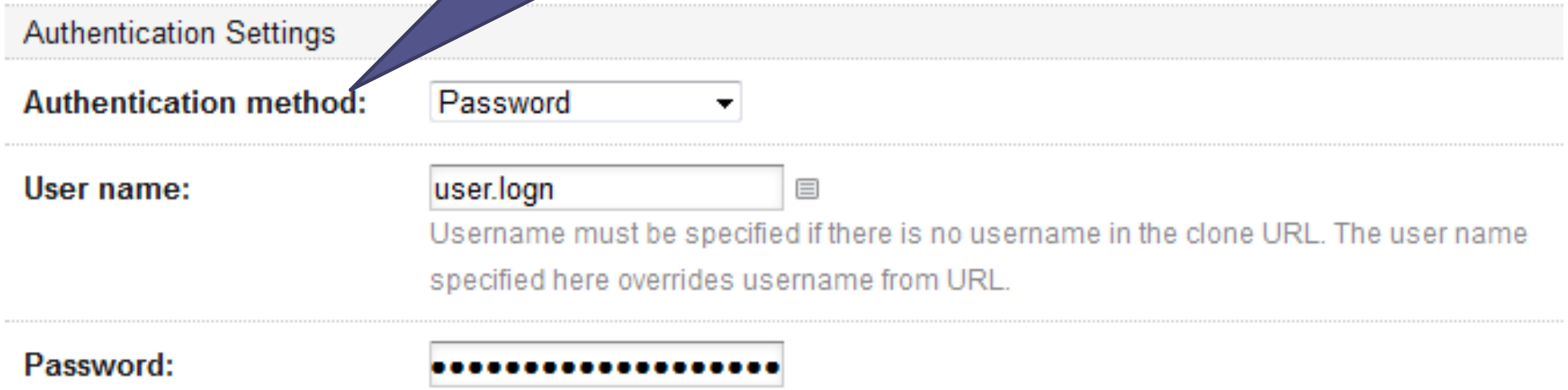
TeamCity: ключевые настройки



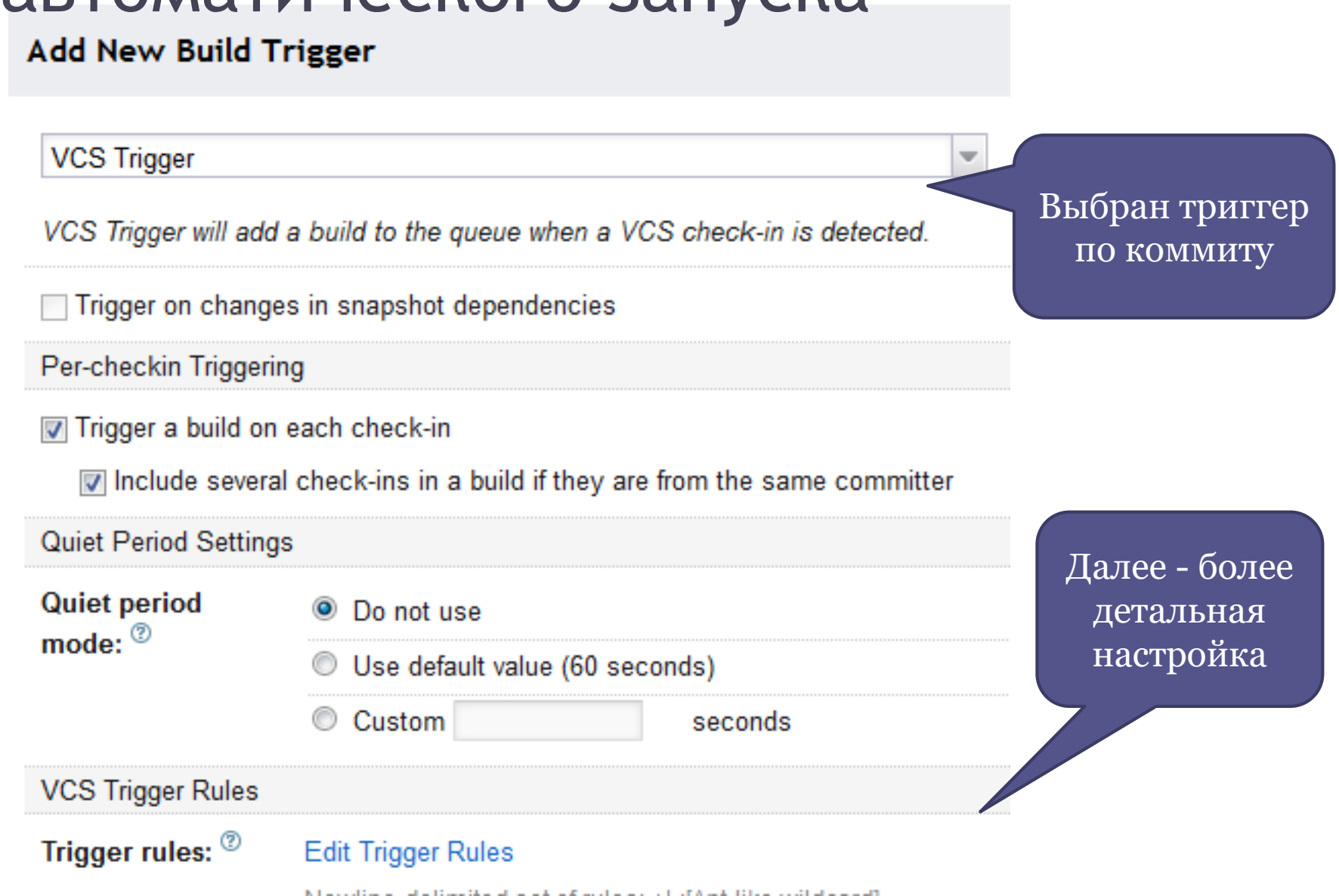
TeamCity: ключевые настройки

TeamCity: настройки авторизации

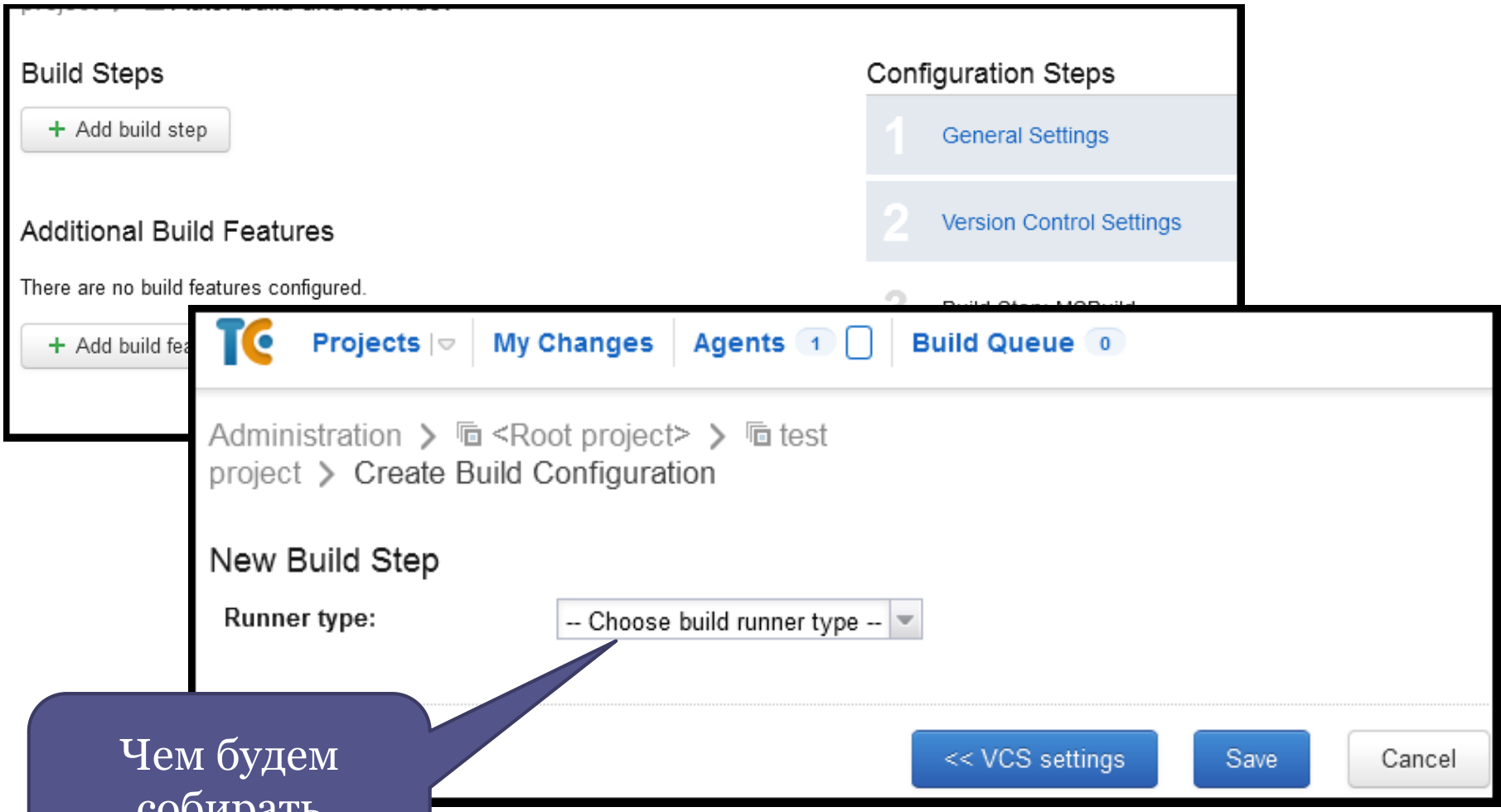
Тут может быть и доступ без авторизации (если данные лежат на внутреннем сервере), и по ключу, и по паролю.



TeamCity: настройка автоматического запуска



TeamCity: настройка шагов сборки



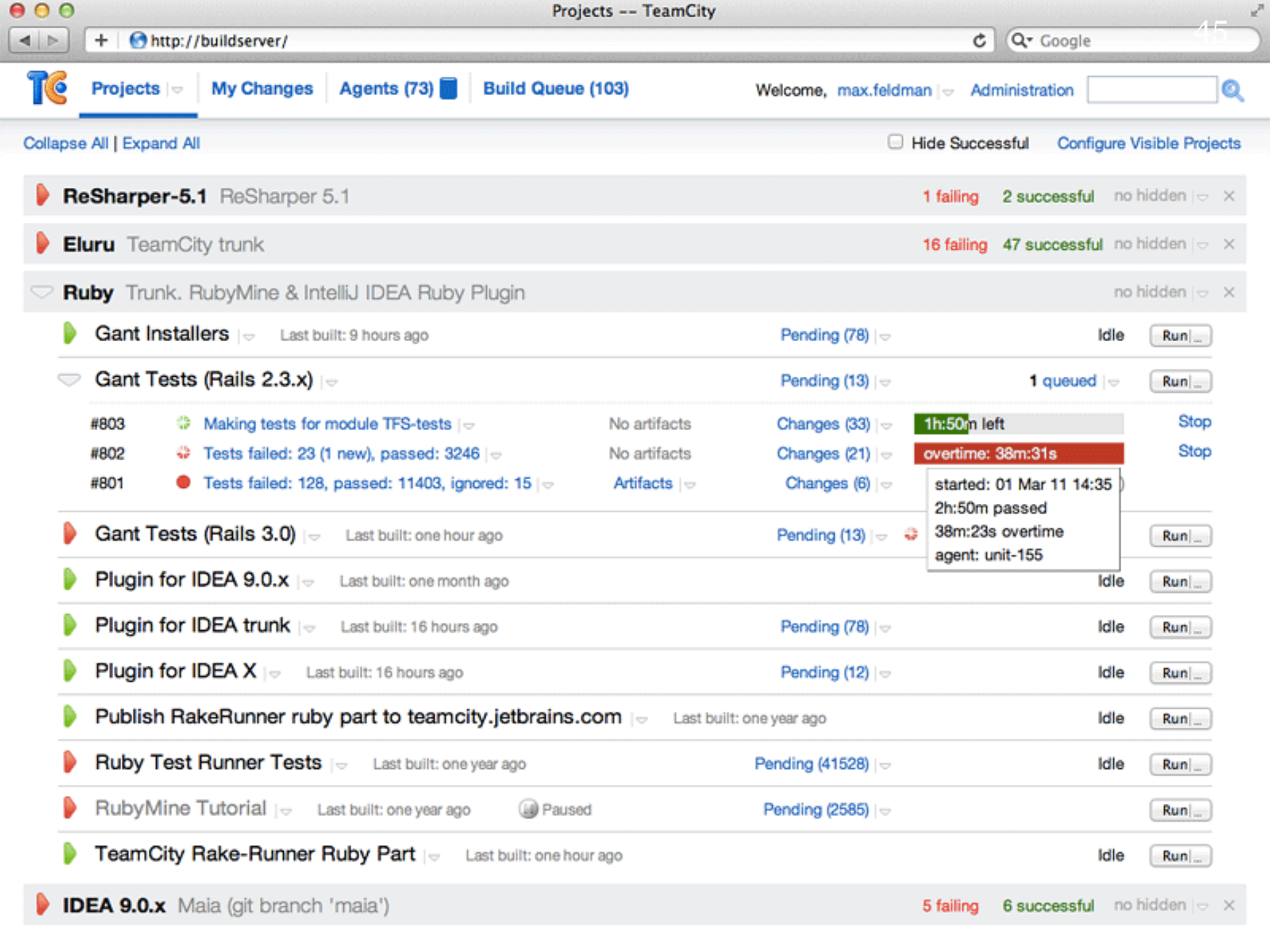
Чем будем собирать исходники

- Более подробный тьюториал можно найти тут:

https://habrahabr.ru/company/skbkontur/blog/205402/]/

TeamCity: использование

- TeamCity GUI будет доступен через HTTP на установленном хосте
- После авторизации, конфигурирования и запуск по триггеру главная страница будет выглядеть примерно как на скриншоте ниже


Projects -- TeamCity
http://buildserver/
Google
45
Projects
My Changes
Agents (73)
Build Queue (103)
Welcome, max.feldman
Administration
Collapse All | Expand All
Hide Successful
Configure Visible Projects
ReSharper-5.1 ReSharper 5.1
1 failing
2 successful
no hidden
Eluru TeamCity trunk
16 failing
47 successful
no hidden
Ruby Trunk. RubyMine & IntelliJ IDEA Ruby Plugin
no hidden
Gant Installers
Last built: 9 hours ago
Pending (78)
Idle
Run
Gant Tests (Rails 2.3.x)
Pending (13)
1 queued
Run
#803
Making tests for module TFS-tests
No artifacts
Changes (33)
1h:50m left
Stop
#802
Tests failed: 23 (1 new), passed: 3246
No artifacts
Changes (21)
overtime: 38m:31s
Stop
#801
Tests failed: 128, passed: 11403, ignored: 15
Artifacts
Changes (6)
started: 01 Mar 11 14:35
2h:50m passed
38m:23s overtime
agent: unit-155
Gant Tests (Rails 3.0)
Last built: one hour ago
Pending (13)
Run
Plugin for IDEA 9.0.x
Last built: one month ago
Idle
Run
Plugin for IDEA trunk
Last built: 16 hours ago
Pending (78)
Idle
Run
Plugin for IDEA X
Last built: 16 hours ago
Pending (12)
Idle
Run
Publish RakeRunner ruby part to teamcity.jetbrains.com
Last built: one year ago
Idle
Run
Ruby Test Runner Tests
Last built: one year ago
Pending (41528)
Idle
Run
RubyMine Tutorial
Last built: one year ago
Paused
Pending (2585)
Run
TeamCity Rake-Runner Ruby Part
Last built: one hour ago
Idle
Run
IDEA 9.0.x Maia (git branch 'maia')
5 failing
6 successful
no hidden

Jenkins (Hudson)

Jenkins (Hudson)

- Серверное ПО, написанное на Java.
- Запускается в контейнере сервлетов, таких как Apache Tomcat или GlassFish
- Основной разработчик Hudson — Косукэ Кавагути — ранее работал в Sun Microsystems и в 2010 году, после поглощения Sun компанией Oracle, основал компанию InfraDNA.
- В ответ на отказ корпорации Oracle передать права на торговую марку Hudson он ответвил проект, дав ему наименование Jenkins.
- В мае 2011 года Oracle отказалась от контроля над проектом и наименованием, предложив целиком передать разработку инструмента под управление Eclipse Foundation.

Jenkins (Hudson): возможности

- Поддерживает системы контроля версий:
  - CVS
  - Subversion
  - Mercurial
  - Git
  - Clearcase и пр.
- Может собирать проекты Apache Ant и Apache Maven (Java), а также исполнять shell-скрипты и команды Windows.
- Hudson имеет «Remote Access API», позволяющее кроме прочего, инициировать сборку проекта, сделав GET-запрос.

Jenkins (Hudson): особенности

- Архитектура Hudson'а построена на основе plugin'ов.
- Сборка проекта заключается в запуске в определённом порядке установленных плагинов, включённых в настройках проекта.

Jenkins (Hudson): особенности

- Плагины можно разделить на несколько условных групп, образующих цикл сборки
  - управление исходным кодом (получение/обновление кода проекта из репозитория),
  - триггеры сборки (настройка времени автозапуска для сборки проекта),
  - среда сборки (настройка среды сборки проекта: версия JVM),
  - сборка (основной этап: запуск плагинов, осуществляющих логику сборки, интеграции и тестирования),
  - послесборочные операции (формирование/публикация отчётов, нотификация).

Jenkins (Hudson): особенности

- По-умолчанию Hudson поставляется с уже установленными плагинами для работы с SVN (централизованная система управления версиями) и Maven (инструмент для сборки java-проектов). Для работы не с Java-проектами требуется установить дополнительные плагины.
- Hudson позволяет изменять только порядок выполнения плагинов, входящих в группу «сборка» (порядок выполнения остальных плагинов в рамках своей группы определяется на основе значений аннотации *@Execution* коде плагинов).

Jenkins (Hudson): особенности

- В случае если необходимо реализовать свой сценарий сборки, для которого не достаточно набора стандартных плагинов из группы «Сборка», можно:
  - вызвать любой внешний исполняемый скрипт, реализующий этот сценарий (пункт ”Execute Shell” из меню ”Add build step”),
  - подключить плагин системы сборки проекта (Phing, Ant, Maven) и указать необходимую цель,
  - написать свой плагин

Jenkins (Hudson)

- Так же, как TeamCity, поддерживает режим ведущий-ведомый. Работа по сборке проектов делегируется нескольким ведомым узлам.

Hudson

search | log in

Hudson » Dashboard

ENABLE AUTO REFRESH

People

Build History

Build Queue

No builds in the queue.

Build Executor Status

#	Status	
1	Idle	
2	Idle	

All | **Dashboard**

S	W	Job ↓	Last Success	Last Failure	Last Duration
		Semantic Assistants	6 hr 58 min (#30)	N/A	5 min 47 sec

Icon: S M L

Legend | for all | for failures | for just latest builds

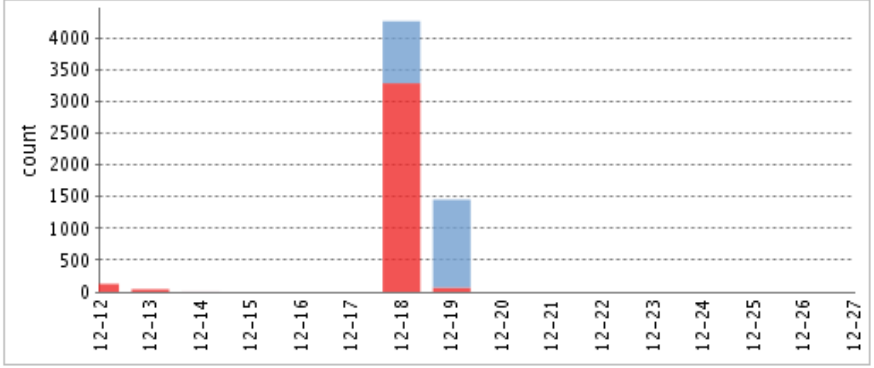
Warnings per project

Job ↓	Checkstyle	Duplicate Code	FindBugs	PMD	Open Tasks	Compiler Warnings	Total
Semantic Assistants	2537	20	125	271	9	83	3045
Total	**2537**	**20**	**125**	**271**	**9**	**83**	**3045**

Build statistics

Status of the build	Description	Number of builds
	Failed	0
	Unstable	0
	Success	27
	Pending	0
	Disabled	0
	Aborted	0
Total builds	All builds	27

Warnings trend graph (new vs. fixed)

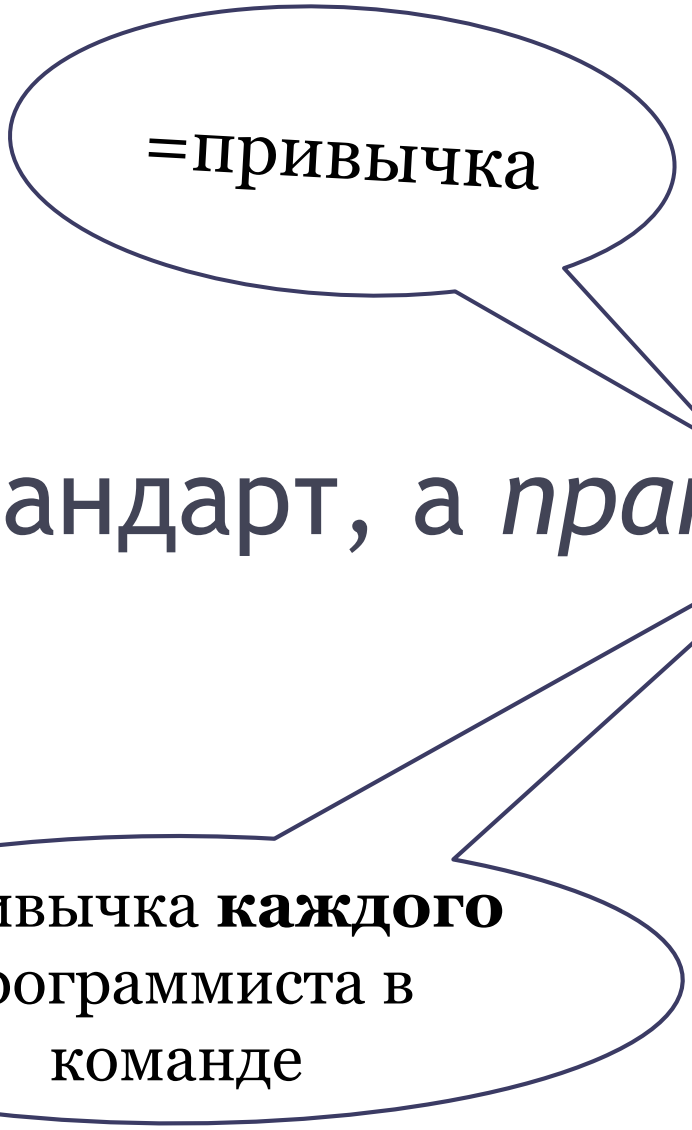


Jobs Grid

Semantic Assistants

- Более подробный тьюториал по настройке:
- https://habrahabr.ru/post/82724/

Внедрение continuous integration: опыт компаний


=привычка
Не стандарт, а *практика*
...привычка **каждого** программиста в команде

Внедрение continuous integration: опыт компаний

- -- Ты последний билд не бери, в нем вот не работает это.
- -- Хорошо
- При этом все билды в CI-системе зеленые.

Программисты придумывают собственный индикатор. Ценность CI сводится к нулю

Внедрение continuous integration: опыт компаний

- Вредная привычка проверять качество кода через исходники.
- Часто — следствие плохо составленных тестов или малого покрытия тестами

«Зеленый» индикатор прохождения тестов — единственный верный критерий качества кода!

Внедрение continuous integration: опыт компаний

- «Как начать писать тесты, как это сделать в проектах к кучей легаси-кода — это отдельная тема, но тут девелоперам надо быть чуть храбрее. Не бойтесь провалиться, начните писать тесты. Плохие тесты лучше, чем никакие...»

Внедрение continuous integration: опыт компаний (Instagram)

- Добавление теста с использованием «канареек» — групп конечных пользователей, которые могут не знать про участие в тесте.
- Вместо запуска раздельного развертывания на одной машине, скрипт развертывается на машинах «канареек», записывает пользовательские логи, а затем запрашивает разрешение на дальнейшее развертывание. Затем осуществляется обработка собранных данных.

Внедрение continuous integration: опыт компаний (Instagram)

- Чтобы исправить падение на тестах, нужно было сначала заметить неполадки, откатить проблемный коммит, подождать прохождения тестов для последнего успешного коммита, а затем вручную развернуть коммиты, которые успели добавить после упавшего, прежде чем процесс сможет продолжиться в автоматическом режиме.
- Все это сводило на нет главное преимущество CI

Решение — увеличение скорости прохождения тестов до 5 минут за полный цикл

Внедрение continuous integration: опыт компаний (Instagram)

- Несмотря на внедренные улучшения, все еще регулярно образовывалась очередь изменений, ждущих развертывания после ошибки.
- После исправления проблемы, система автоматизации выбирала по одному коммиту для развертывания — чтобы развернуть их все, нужно было время, за которое в очередь могли попасть и новые изменения.

Реализован алгоритм развертывания множества изменений при наличии очереди

Внедрение continuous integration: опыт компаний (Instagram)

- **Людям должно быть удобно**.
- Большое препятствие: люди перестают понимать, что происходит в текущий момент времени, и лишаются возможности контроля.
- Система должна обеспечивать хорошее понимание того, что сделано, что делается, и (в идеале) будет сделано на следующем шаге.
- Необходим хорошо работающий механизм экстренной остановки автоматического развертывания.

По ночам компьютеры собираются вместе и смеются над людьми, если те делают работу, которую могли бы делать компьютеры

Ссылки на материалы

- Статья Мартина Фаулера о CI http://martinfowler.com/articles/continuousIntegration.html
- Общее описание процесса: русскоязычный источник http://lib.custis.ru/Непрерывная_интеграция
- Категории программных тестов https://habrahabr.ru/post/64874/

Ссылки на материалы

- Настройка TeamCity https://habrahabr.ru/company/skbkontur/blog/205402/
- Использование TeamCity https://habrahabr.ru/post/105895/
- Настройка Hudson https://habrahabr.ru/post/108928/
- Опыт внедрения https://megamozg.ru/post/5306/

  https://habrahabr.ru/company/latera/blog/283102/