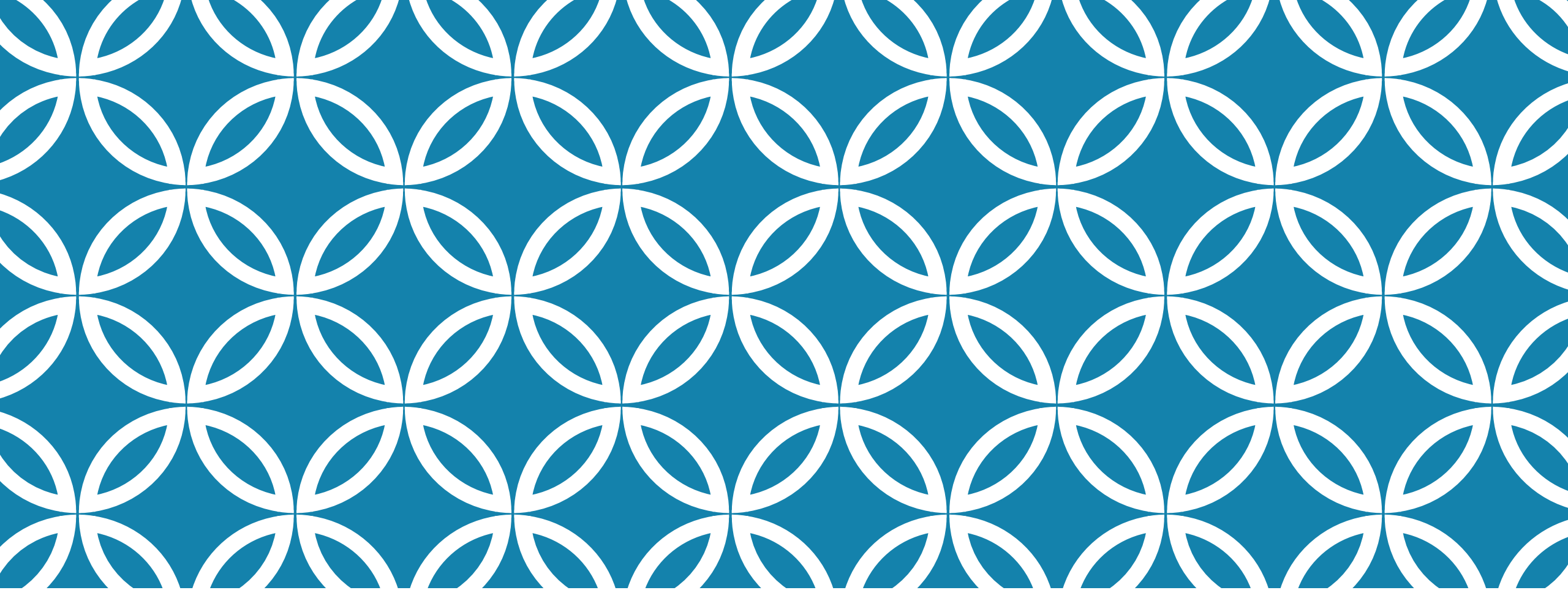


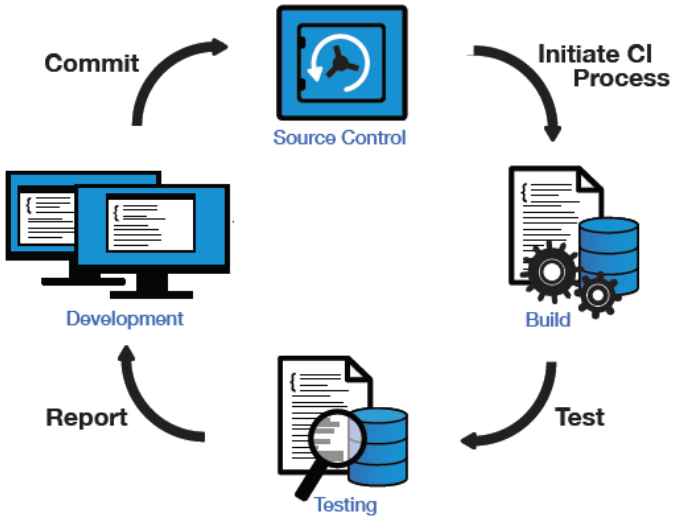
CI И РОЛЬ ТЕСТИРОВАНИЯ

Антонина Когут,
ПЗС-2

CI? НЕТ, НЕ СЛЫШАЛ...

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

Зачем? – Чтобы не дожидаться конца проекта для проведения интеграции и внезапной всемирной боли.



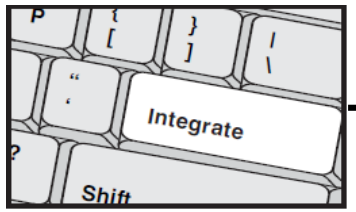
БЕЛЫЙ ЯЩИК

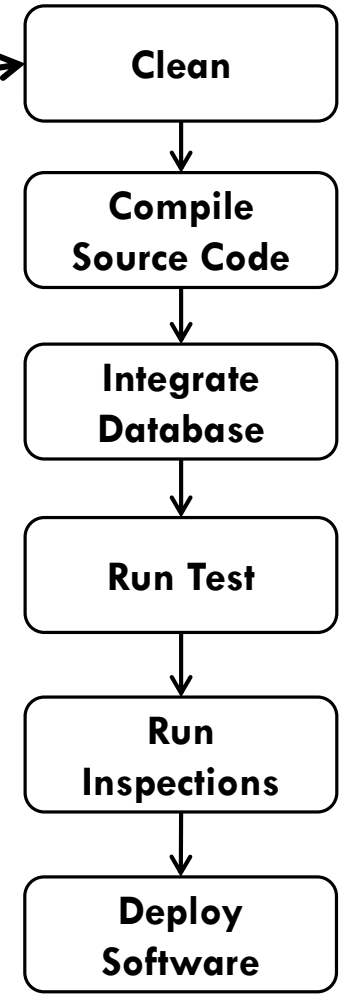
1. Система контроля версий из которой забираются исходники.
2. Билд-скрипт.
3. Сервис по запуску сборки.
4. Сервис по запуску тестов.
5. Статический анализ кода.
6. Анализ покрытия кода тестами.

ОСНОВНЫЕ ПРИНЦИПЫ

Perform Single Command Builds	Automate Builds	Separate Build Scripts from Your IDE	Centralize Software Assets
Create a Consistent Directory Structure	Fail Builds Fast	Build for Any Environment	Use a Dedicated Integration Build Machine
Use a CI Server	(Optional) Run Manual Integration Builds	Run Fast Builds	Stage Builds

СДЕЛАТЬ BUILD ОДНОЙ КОМАНДОЙ





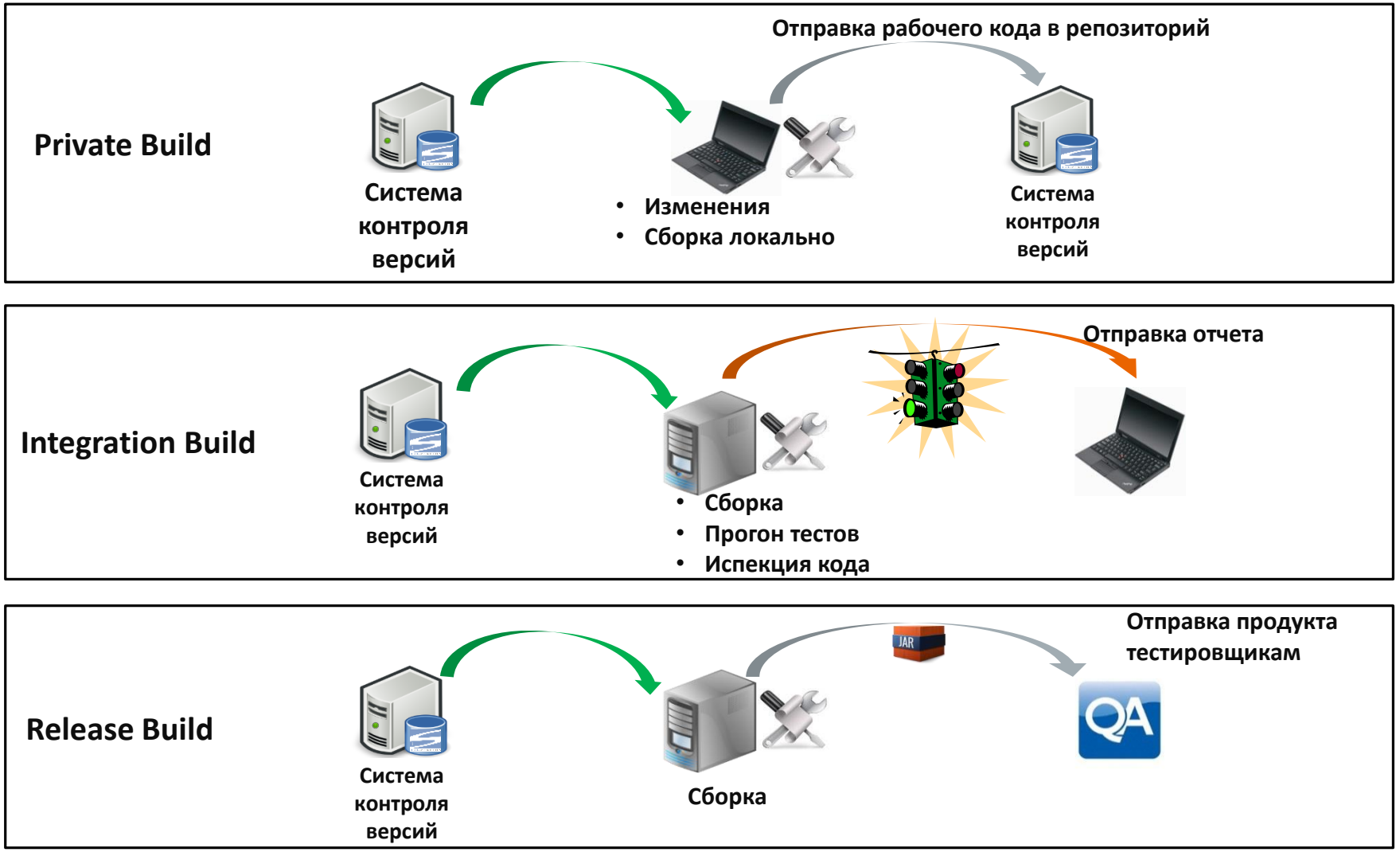
«Нажать кнопку» – и получить build!

maven

<APACHE ANT>

CMake

АВТОМАТИЗИРОВАТЬ БИЛДЫ

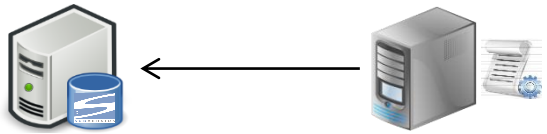


МЕХАНИЗМЫ СБОРКИ

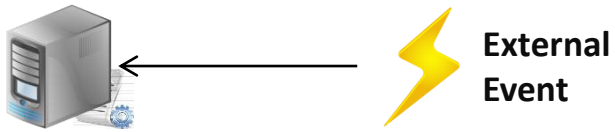
По требованию

По расписанию

Poll for Changes

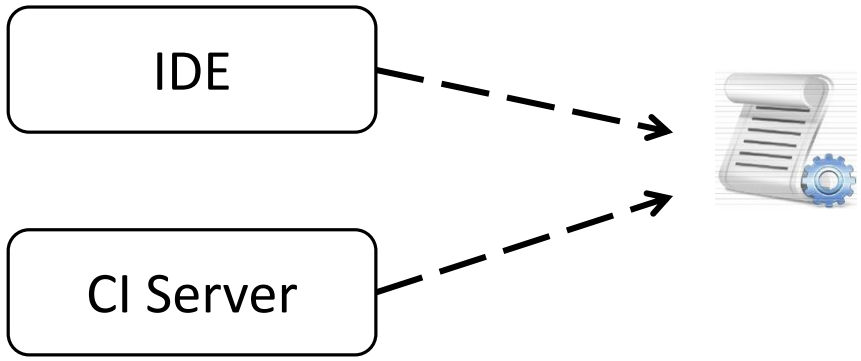


Event Driven



Private Build	По требованию
Integration Build	По требованию, Poll for Changes, По расписанию, Event Driven
Release Build	По требованию, По расписанию

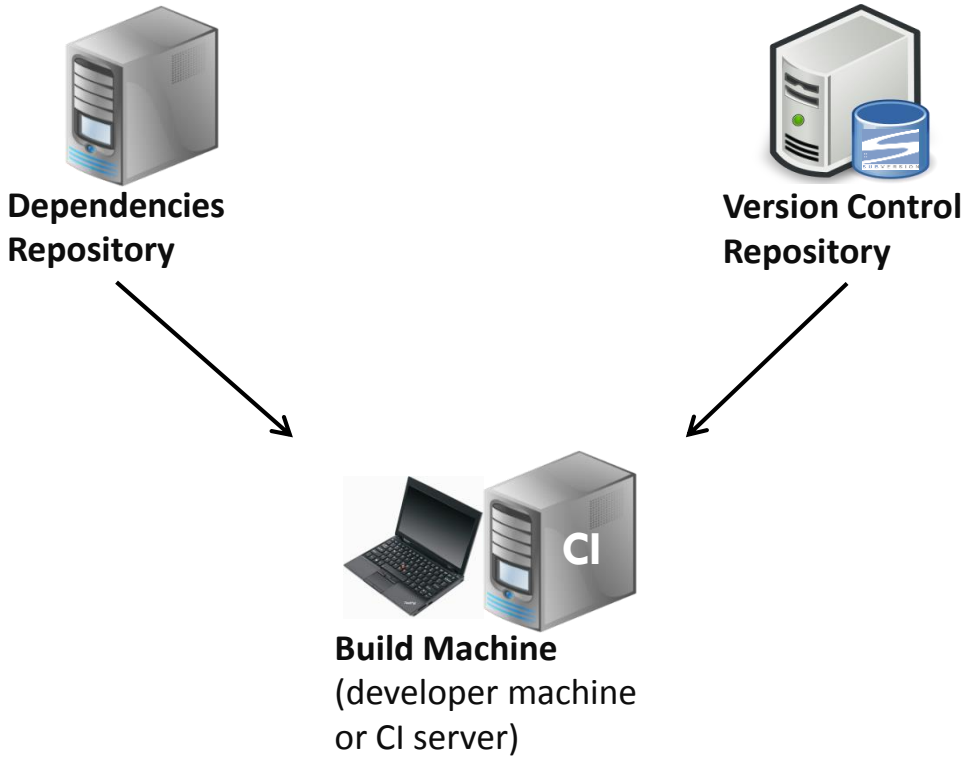
ОТДЕЛИТЬ СКРИПТ ОТ IDE



1. Разработчики используют разные IDE.
2. CI сервер должен выполнять сборку без вмешательства человека.

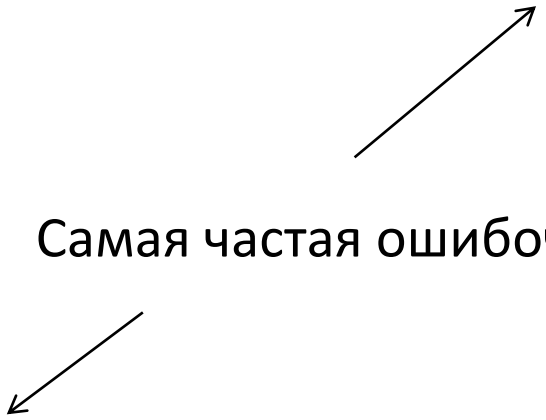
ЦЕНТРАЛИЗИРОВАНЫЙ ДОСТУП

Возможность получить данные в любом месте в любое время.



ЕСЛИ ПАДАТЬ, ТО БЫСТРО

Test Case	Status
Test Case #1	OK
Test Case #2	OK
Test Case #3	OK
Test Case #4	OK
Test Case #5	OK
…	…
Test Case #(N-1)	**FAIL**
Test Case #N	OK



Test Case	Status
Test Case #1	**FAIL**
Test Case #2	OK
Test Case #3	OK
Test Case #4	OK
Test Case #5	OK
…	…
Test Case #(N-1)	OK
Test Case #N	OK

Узкие места должны тестироваться в первую очередь.

ДЕЛАЙТЕ ТЕСТЫ

Тесты просто необходимо включать в continuous integration процесс.

Основными двумя ограничителями на количество тестов будет:

1. время интеграции — сборка по-прежнему должна оставаться быстрой, основное тестирование можно перенести «на ночь»,
2. наличие автоматизированных тестов — не все тесты требуют автоматизации, нет смысла делать автоматизированные тесты только для самих тестов, они должны быть целесообразны.

Чем лучше ваши тесты, тем раньше находятся ошибки и раньше исправляются

Это одно из основных преимуществ практики непрерывной интеграции — снижение стоимости исправления ошибок (не всех конечно). Попутно наличие хорошего набора тестов в процессе интеграции дает больше уверенности в том, что проект работает правильно.

СБОРКА ДЛЯ ЛЮБОЙ СРЕДЫ

Developer Machine #1

…

Developer Machine #N

или

CI Server #1

…

CI Server #N

или

или

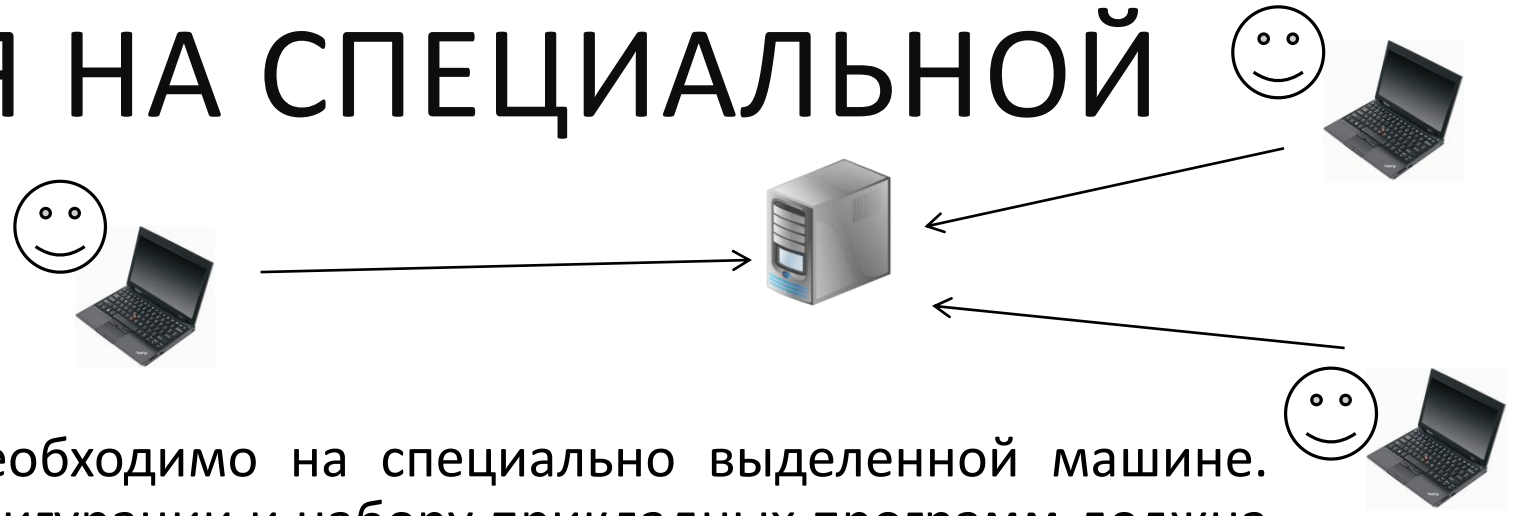
Separate Build Script

Source Code
Dependencies
Scripts
Test Sources
Config Files
etc.
from centralized storages (repositories)

Output build files **must be identical**!

*Почему не работает на сервере??? **У меня локально все работает!***

ИНТЕГРАЦИЯ НА СПЕЦИАЛЬНОЙ МАШИНЕ



Организовывать процесс необходимо на специально выделенной машине. Такая машина по своей конфигурации и набору прикладных программ должна максимально соответствовать окружению в котором проект будет развернут (production enviroment).

Очевидно, что полного совпадения достичь практически невозможно — маловероятно, что эксплуатироватся программа будет на машине с установленными средствами сборки, тестирования и проч. Но точное совпадение версий операционных систем (и сервис паков) необходимо.

ИСПОЛЬЗУЙТЕ CI СЕРВЕР

Bamboo

cruisecontrol

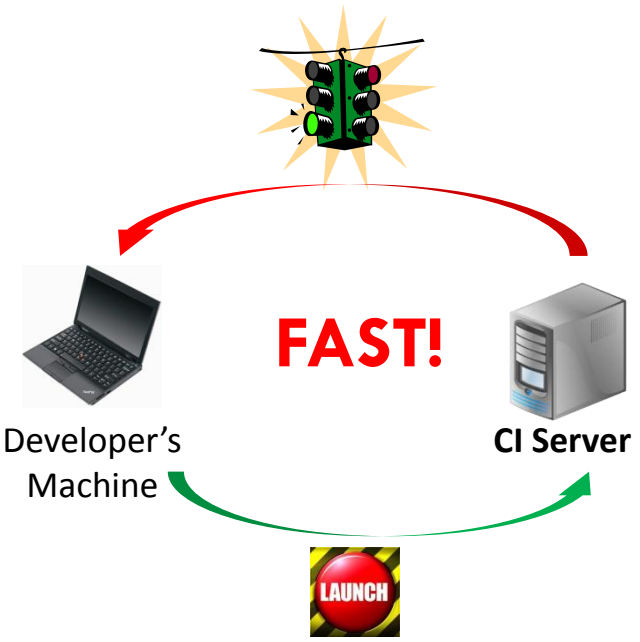
Jenkins

continuum ™

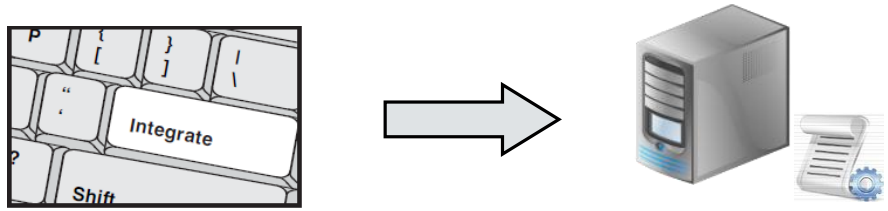
TeamCity

БЫСТРЫЕ БИЛДЫ

1. Использовать по максимуму машину, которая отвечает за сборки.
2. Делать комиты чаще.
3. Оптимизировать инфраструктуру.
4. Улучшить производительность тестов.
5. Улучшить производительность кода.



СТАДИИ БИЛДОВ



Commit Build (Lightweight)

- Компиляция исходников
- Интеграция БД
- Юнит-тесты
- Инспекция
- Деплой

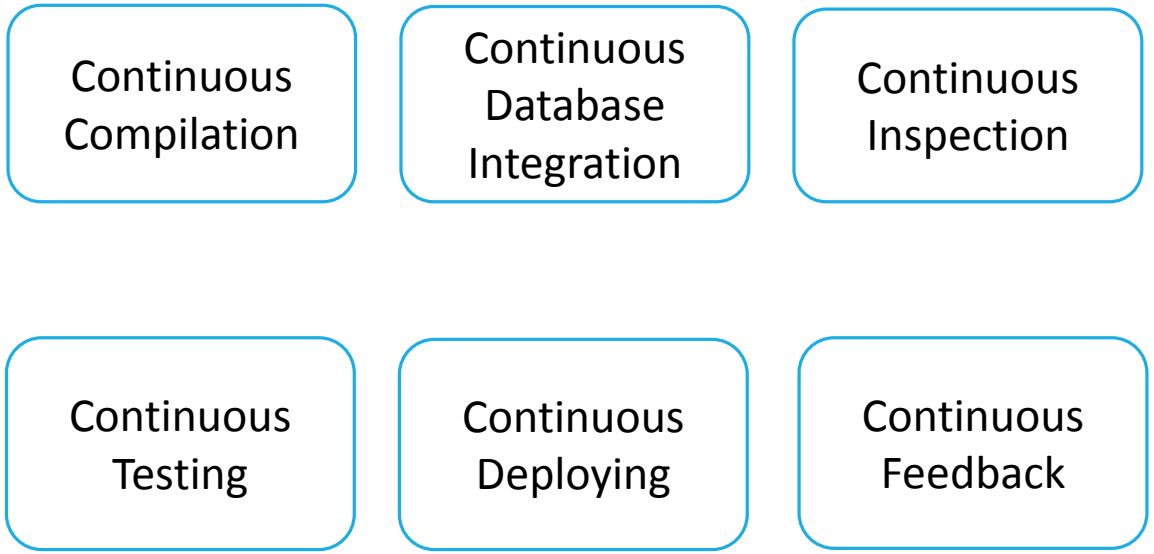
Secondary Build (Heavyweight)

- component тесты
- Системные тесты
- Функциональные тесты
- Тесты производительности
- code coverage

Custom Build (project-specific)

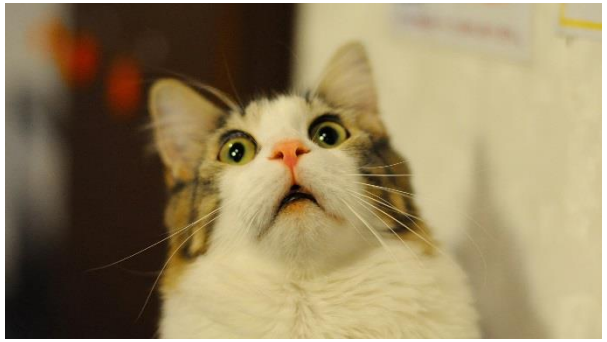
- Ручные билды
- Специфические интеграционные билды

ОСНОВНЫЕ СОСТАВЛЯЮЩИЕ CI СИСТЕМЫ



1. Разработчики должны видеть результаты изменений в исходниках.
2. Первое, что нужно сделать на сервере.
3. Убедиться в отсутствии проблемы «а у меня локально работает!».
4. Не использовать Continuous Compilation вместо локальных билдов.

Скрипты тоже являются частью кода.



Советы:

1. Наиболее частые операции с БД сохранить в скрипте.
2. Такие скрипты нужно хранить с СКВ.
3. Использовать локальную копию базы.
4. Использовать инструменты инспекции.

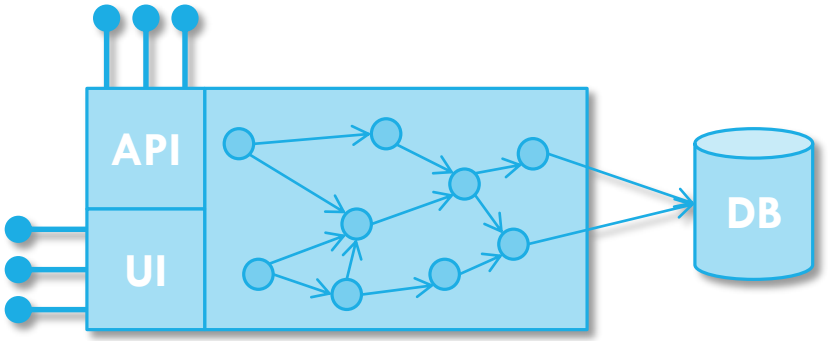
1. Сложные особенности языка программирования.
2. Непонятные API методы.
3. Неиспользованные переменные.
4. Пустые catch-блоки.
5. Ctrl+C, Ctrl+V

Инструменты инспекции улучшают качество кода, находят скрытые баги перед тем, как они уйдут в общий репозиторий. Такие инструменты учат программиста писать код чисто.

Continuous Testing значит, что все тесты должны быть автоматизированными, выполняться в любой среде и часто. Собирать их нужно вместе, чтобы получить комплексный отчет.

Создаются тесты на фазе разработки.

Согласно принципам CI, тесты, вероятность падения которых выше, чем у остальных, должны выполняться первыми.



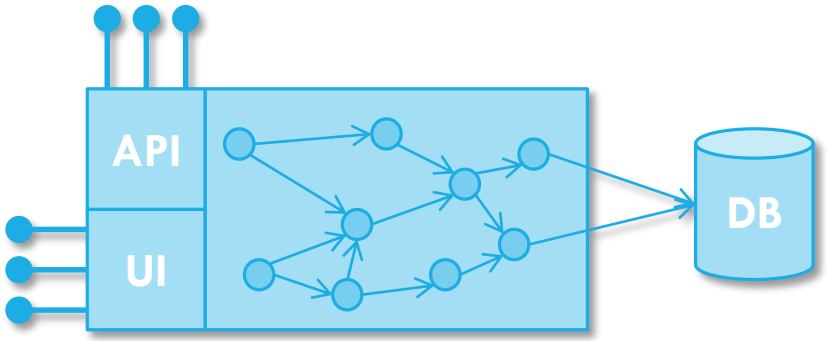
Типы тестов в CI:

1. Юнит-тесты
2. Интеграционные тесты
3. Системные тесты
4. Функциональные тесты.

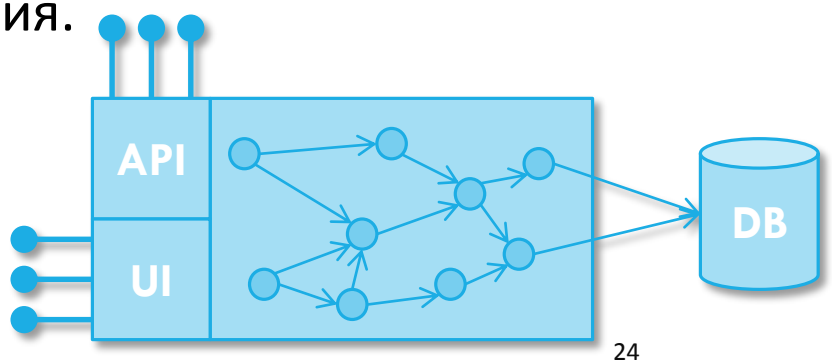
Юнит-тесты проверяют поведение маленьких компонентов системы, как правило методов или классов. Каждый такой тест должен удовлетворять следующим характеристикам (F.I.R.S.T):

Название	Описание
Fast	требует мало времени для исполнения (< 0.01 sec)
Isolated	Не взаимодействует с другими компонентами системы
Repeatable	Можно повторить в любое время
Self-checking	Нету ручной проверки
Timely	Пишут перед появлением кода

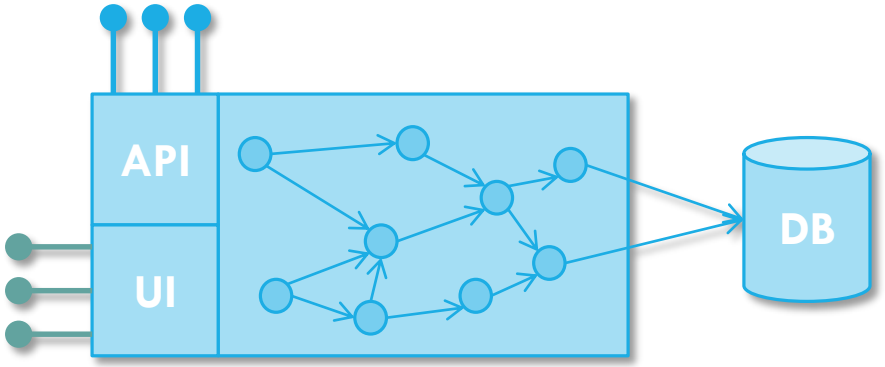
1. **Компонентные тесты** обычно выполняются дольше, чем юнит-тесты.
2. Могут требовать полностью установленной системы и каких-либо зависимостей (БД, ФС, составляющих сети)
3. Не настолько изолированные, как юнит-тесты.



1. **Системные тесты** полностью проверяют поведение системы.
2. Требуют, чтобы система была полностью сконфигурирована и настроена.
3. Могут использовать веб-сервисы или ejb-вызовы.
4. Более медленные, чем компонентные тесты, поскольку требуют полного программного деплоя, инициализации БД и взаимодействия между веб-протоколами.
5. Используют системные API для выполнения тестрования.

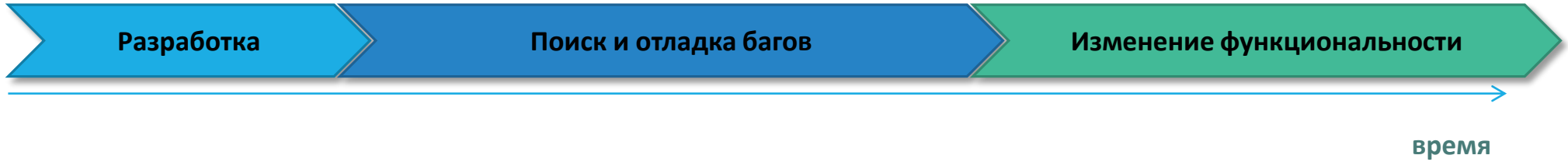


1. **Функциональные тесты** тестируют приложение с точки зрения клиента.
2. Обычно выполняются дольше, чем системные тесты.
3. Требуют более четких и изящных проверок, чем предыдущие и должны покрывать большое количество возможных ситуаций.

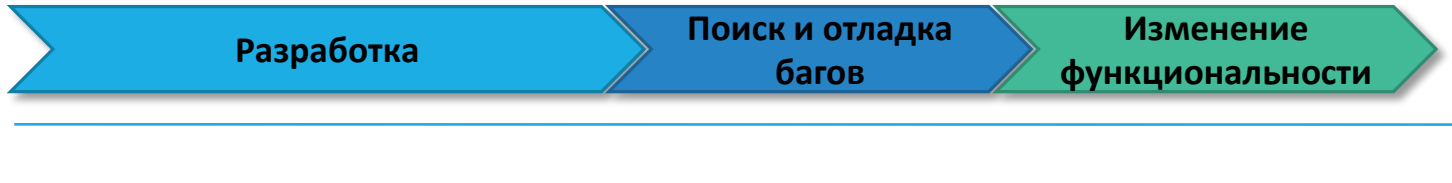


	Юнит-тесты	Компонентные тесты	Системные тесты	Функциональные тесты
Так же известны как		Интеграционные		Acceptance/Story
Зависит от				
Время выполнения	мс	сек	мин	ч
Описание	Проверка метода/класса	Запросы в БД	Системное АПИ(ВС), внешнее клиентское взаимодействие	Ориентированы на заказчика

Без Continuous Testing



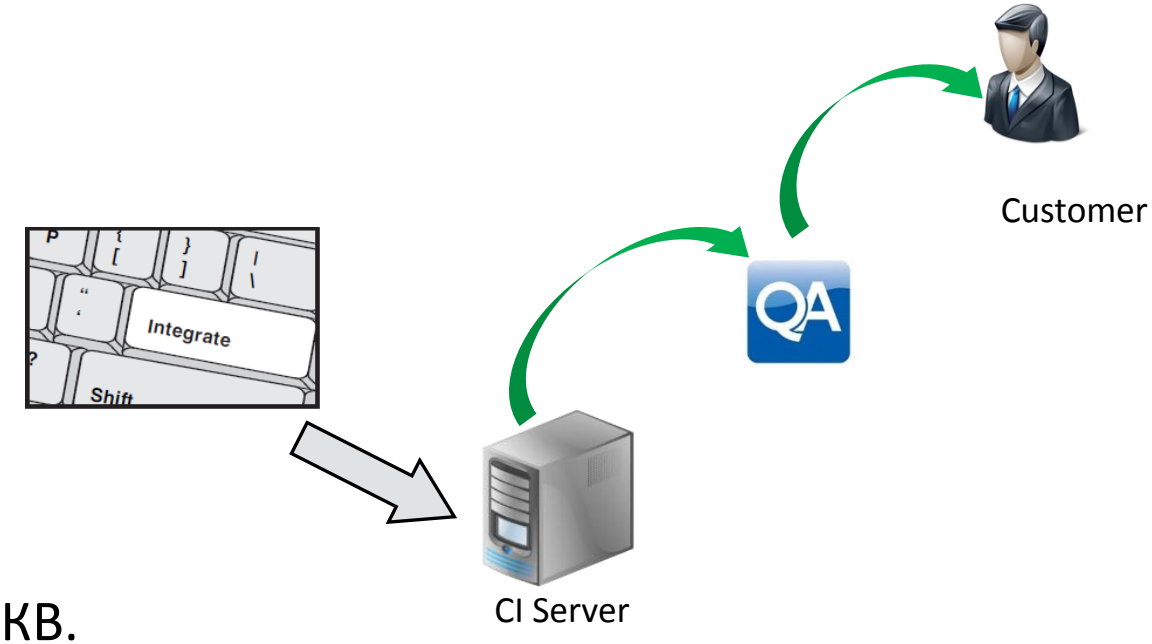
С Continuous Testing



Тесты уменьшают время для поиска багов, их исправление, а также для изменения функциональности.

Зачем?

- Для системных/функциональных тестов.
- Деплой артефактов в репозиторий.

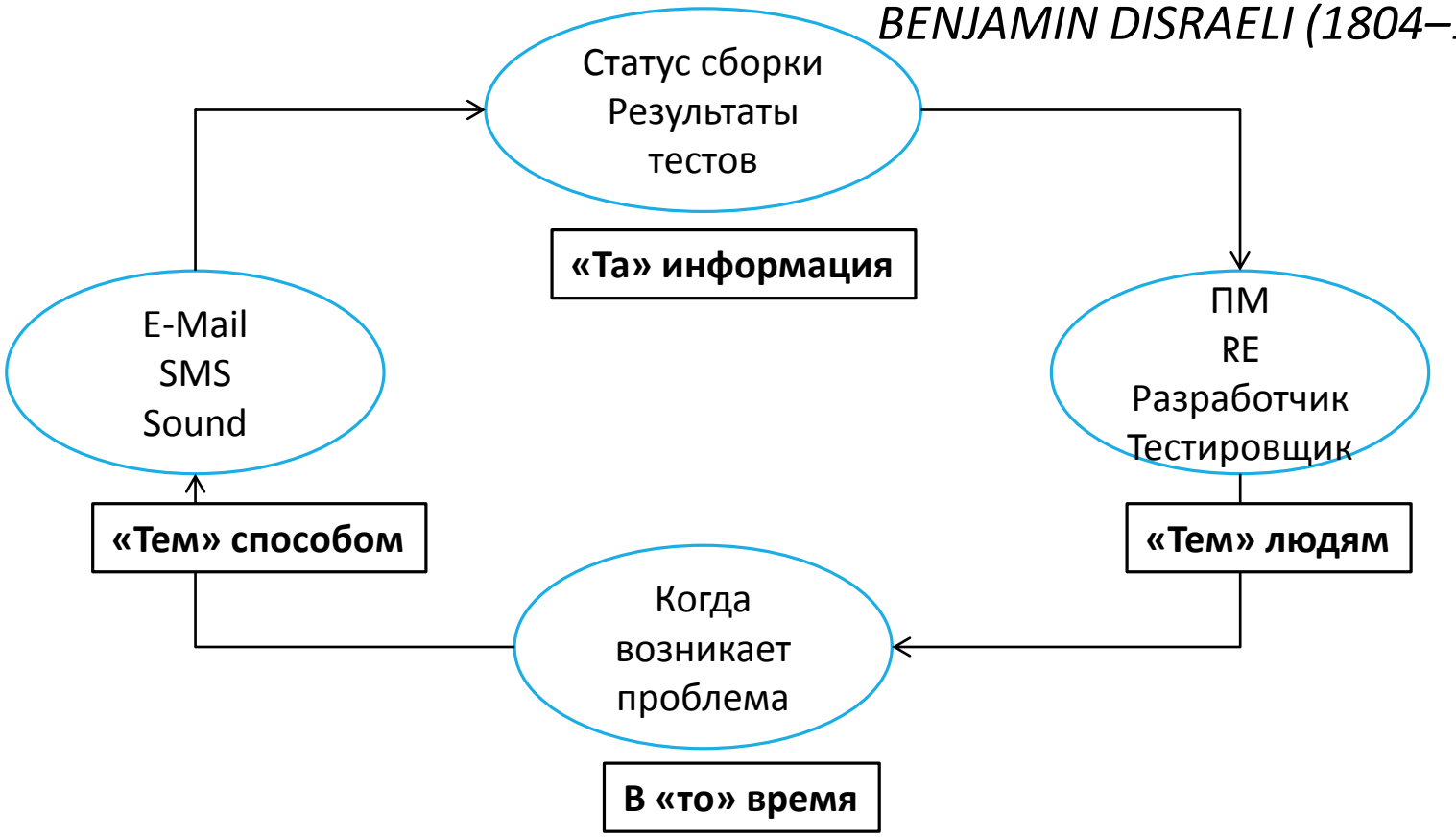


Принципы:

- Деплой в любом месте, в любое время
- Деплой делам только из исходников, что в СКВ.
- «Деплой с меткой» (версия, например)
- Выполнение всех тестов перед каждым деплоем.

As a general rule, the most successful man in life is the man who has the best information.

BENJAMIN DISRAELI (1804–1881)



ПРЕИМУЩЕСТВА И НЕДОСТАТКИ

1. Регулярная интеграция всего приложения.
2. Экономия времени
3. Уменьшаются риски получить «гранату». Дефекты находятся раньше. Это достигается путем запуска тестов (тесты можно запускать различных уровней: unit, GUI, API) и ранней отдачи нового/измененного функционала на тестирование.
4. Почти всегда есть развернутое окружение для тестирования и демонстрации работы заказчику и прочим заинтересованным.
5. Можно безболезненно эмитировать процесс деплоя на тестовых серверах.

«-»

1. Затраты на поддержку окружения

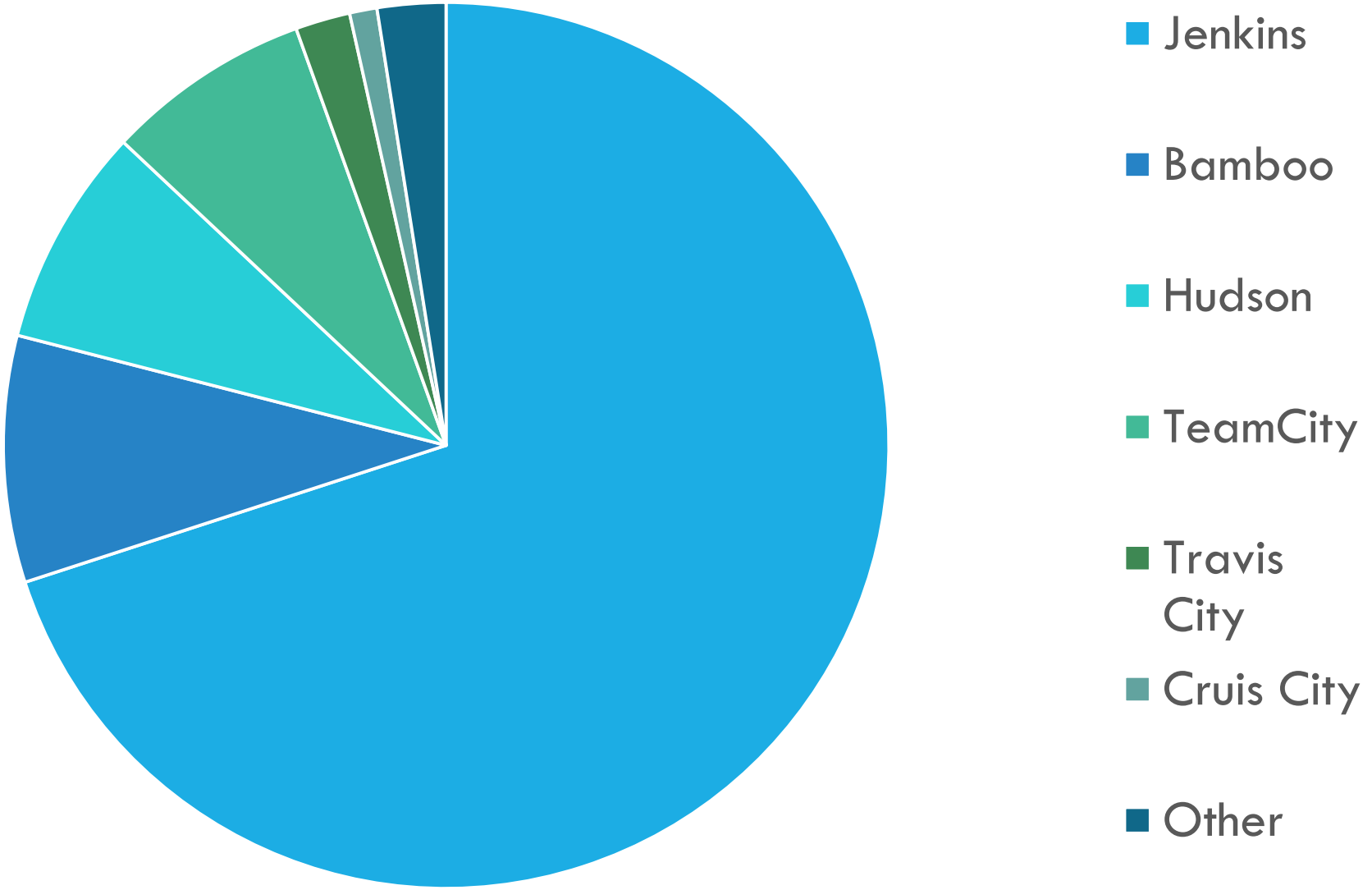
ОСОБЕННОСТИ ТЕСТИРОВАНИЯ. 1/2

1. Отсутствие авто-тестов.
2. Постоянное обновление тестовых серверов.
3. Иногда, при очередной сборке тестовые данные могут теряться.
4. На проектах с большим количеством веток и окружение иногда теряешься, где и что настроено. Это проблема решается путем публичности — все участники команды тестирования должны знать, где посмотреть настройки окружения.

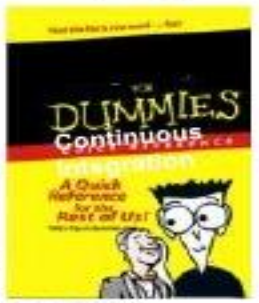
ОСОБЕННОСТИ ТЕСТИРОВАНИЯ. 2/2

1. Постоянно доступны последние изменения.
2. Стабильная сборка.
3. Автоматически готовое окружение для тестирования.
4. Есть ответы на многие вопросы — что? где? когда?
5. CI стимулирует наших разработчиков к улучшению кода и внедрению инженерных практик.
6. Можно быстро откатиться на предыдущие версии приложения.
7. Окружение, которое максимально похоже на продакшан делает тестирование полезней. Также, созданное «эталонное» окружение должно избавить от вечных «у меня локально не воспроизводится» от программистов.

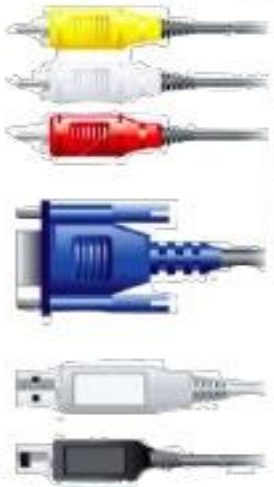
ИНСТРУМЕНТЫ



ПОЧЕМУ JENKINS?



Простая установка и использование



Плагины

Отчеты

Distributed build

Свободный доступ

Крос-платформенный

УЖЕ ИСПОЛЬЗУЮТ

JENKINS. TEST REPORTS

Сборки генерируют тестовые отчеты в различных форматах, которые поддерживаются множеством плагинов.

https://wiki.jenkins-ci.org/display/JENKINS/Cucumber+Test+Result+Plugin

https://wiki.jenkins-ci.org/display/JENKINS/Test+Results+Analyzer+Plugin

TEAMCITY

1. Pre-tested commit (коммит с предварительной проверкой) позволяющий избежать попадания ошибок в финальный код и оптимизировать цикл интеграции.
2. Показ результатов тестирования «на лету».
3. Для передачи результатов тестов используют плагины.
4. Порядок и логику вызова тестов все же настраивают в билд-скрипте.
5. Нету возможности рестартовать отдельный тест.
6. Грид-сборка проекта. Предоставляет возможность производить несколько сборок проекта одновременно, производя тестирование на разных платформах и в различном программном окружении

https://teamcity.jetbrains.com/

TEAM FOUNDATION SERVER

Сервер корпоративного уровня для групп, который позволяет им совместно использовать код, отслеживать работу и поставлять программное обеспечение

Включает в себя репозитории кода, непрерывную интеграцию, отслеживание ошибок и задач

Работайте на любом языке, включая Java, Python, HTML5, JavaScript, C#

Используйте Visual Studio, Eclipse, Xcode или собственную интегрированную среду разработки

Предоставляется бесплатно TFS Express для пяти участников группы с возможностью расширения по мере ее роста.

TEAM TEST LOAD AGENT

В дополнение к Team Foundation Server Microsoft также предлагает серверный компонент Team Test Load Agent (модуль командного нагрузочного тестирования).

Этот инструмент, который лицензируется отдельно от Team Foundation Server и Visual Studio, предназначен для использования тестировщиками для выполнения автоматизированного нагрузочного тестирования веб- или Windows-приложений.

ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ

Team Web Access	Microsoft Test Manager
Планирование ручных тестов	Произвольное тестирование
Создание ручных тестов	Планирование ручных тестов
Выполнение ручных тестов	Выполнение ручных тестов
Отслеживание результатов тестов	Конфигурация тестов: указание платформ тестирования
Совместное использование шагов в тестовых случаях	Сбор дополнительных данных диагностики в ручных тестах
Повторение теста с другими данными	Тестирование в лабораторной среде
	Автоматизация системных тестов

https://msdn.microsoft.com/library/ms182409(v=vs.140).aspx

ВОПРОСЫ?

СПАСИБО ЗА ВНИМАНИЕ!