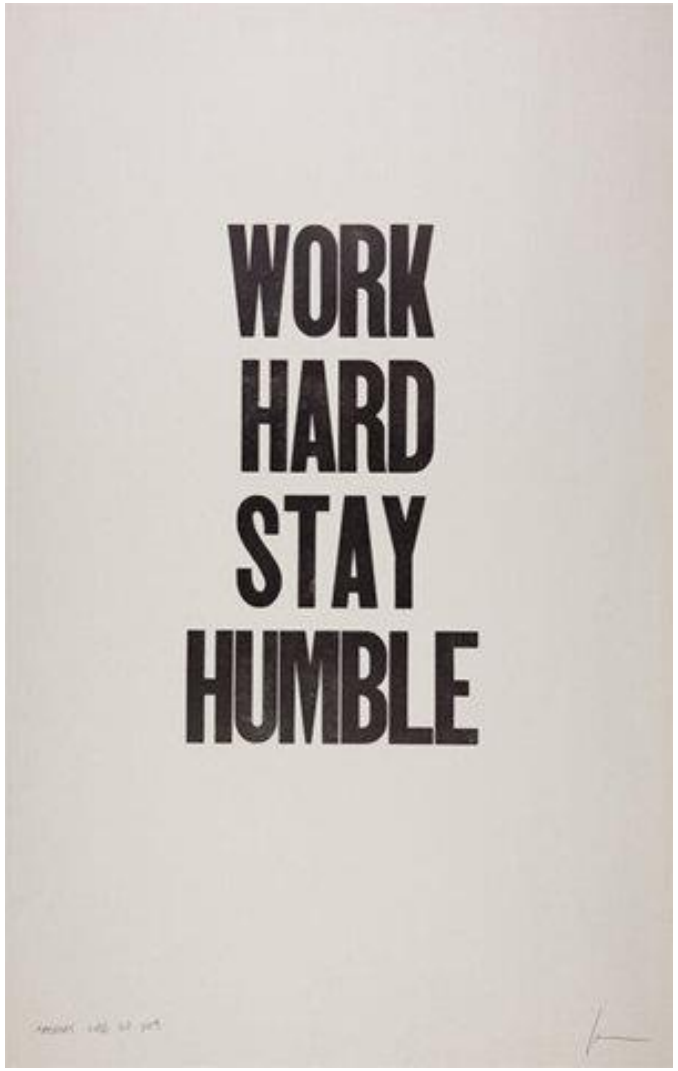
Один день из жизни разработчика-исследователя

Устройство компаний

- Горизонтальное
- Вертикальное

Горизонтальное Устройство компаний



Горизонтальное Устройство компаний

- Все сотрудники компании равнозначны
- Компания ведет открытую политику касательно всего
- Сотрудник не может вырасти вертикально (программист → старший программист → технический директор). Должностей как таковых по сути нет

Горизонтальное Устройство компаний

- Более 300 сотрудников
- Стоймость компании более 2.000.000.000$

valve employee handbook


G-Man
Our Mutual Friend
Communication (Ha
VALVE
PC gamer
N. KOREA
S. KOREA
Linux Update
BreenGrub
Half-Life (1)
Half-Life 2
Episode One ✓
Episode Two ✓
Episode Three ✗
Half-Life 3
2013
I WANT TO BELIEVE
예측하
과에 대비하라...
BOREALIS
Aperture Science
Cave Johnson
Gordon Freeman
Grdn Frman
Gn Fman
G-man
Gabe's Gun

Горизонтальное Устройство компаний

- Более 30 сотрудников
- Евангелисты современного web2.0 (Ruby On Rails)
- Авторы нескольких культовых книг о разработке ПО и ведении бизнеса

https://gettingreal.37signals.com/

Вертикальное Устройство компаний

- Становится необходимым при увеличении количества сотрудников и усложнении структуры бизнеса
- Иерархическая модель исторически более популярна и широкоприменима

Вертикальное Устройство компаний

- Более 3000 сотрудников
- Самая популярная онлайн игра в мире

National Geographic’s Ultimate Factories: Wargaming

Лаборатория Касперского

Лаборатория Фильтрации Контента

- Антифишинг
- Антиспам
- Родительский контроль
- Отдел аналитики
- Команда инфраструктуры

ЗАДАЧИ

- Поддержка и создание инфраструктурных сервисов для внутреннего использования
- Исследование новых методов анализа контента (преимущественно в области антиспама)

Команда

- 1 менеджер
- 1 системный администратор
- 3 тестировщика
- 4 разработчика

Общий рабочий процесс

- Год – 4 квартала, глобальные цели
- Квартал – квартальные цели
- Цели – декомпозиция на задачи, необходимые для их достижения
- Задачи – планирование, 2х недельные итерации
- Итерации синхронизованы между всеми командами в отделе

МОДЕЛИ РАЗРАБОТКИ ПО

- Каскадная
- Итеративная
- Спиральная
- Vee/Double-Vee

Методологии Итеративной разработки ПО

- Agile:
  - Kanban
  - Scrum
  - Feature-Driven Development
  - eXtreme Programming
  - RUP

SCRUM

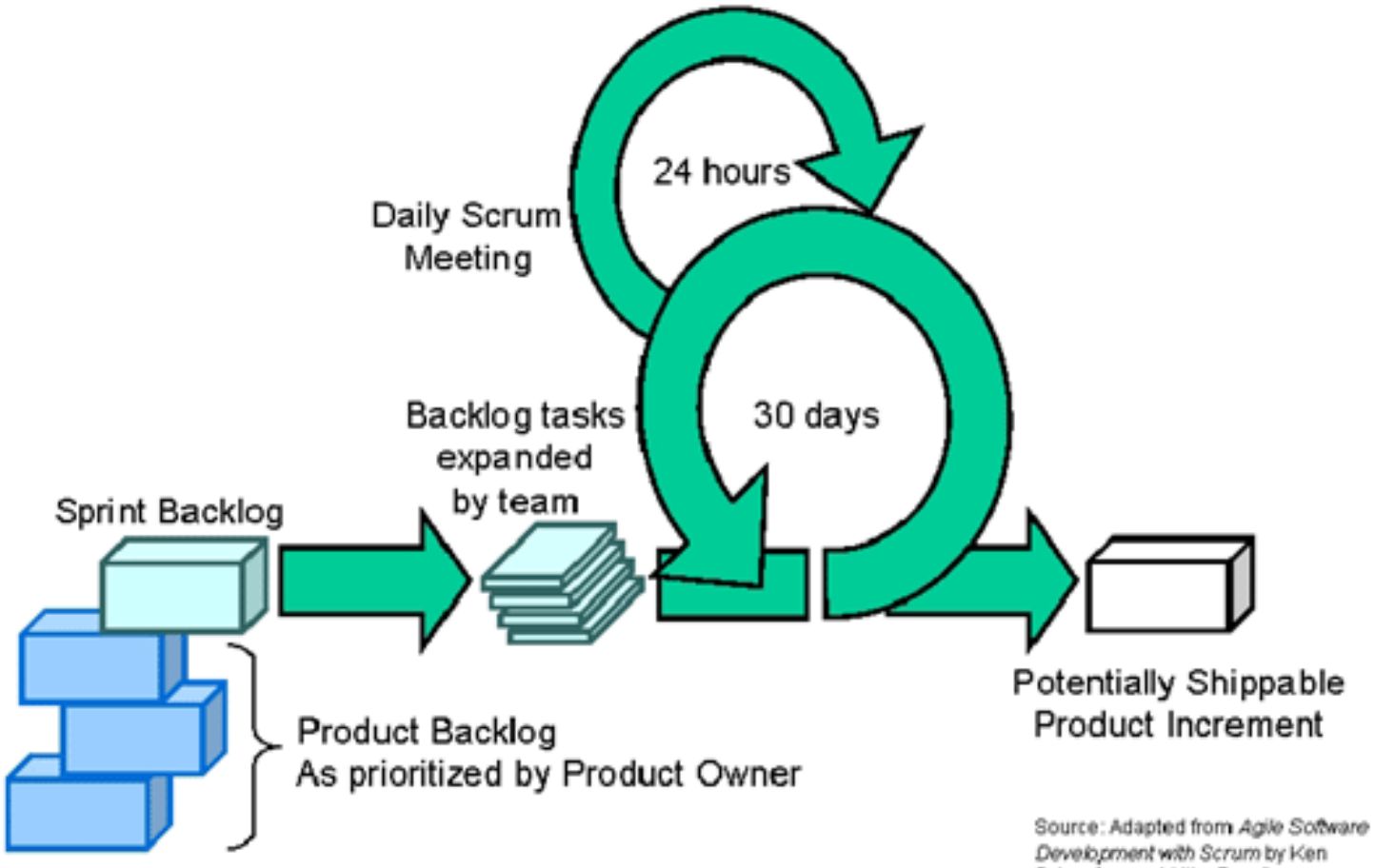
- Команда маленького размера, все разработчики в идеале должны обладать одинаковой областью компетенции
- Декомпозиция задач на подзадачи, выполнимые за относительно короткий промежуток времени (итерацию)
- Длительность итерации – 2 недели
- В конце каждой итерации – ретроспектива по недостаткам в итерации, демонстрации выполненных задач, планирование следующей итерации
- В каждую итерацию закладывается фиксированное количество часов на разнообразную “текучку”
- Задачи не привязаны к конкретному разработчику
- Короткие встречи каждый день с обсуждением сделанного за прошлый и новостей/изменений

SCRUM

В итоге мы получаем гибкую систему, позволяющую подменять одним разработчикам других в ходе работы над проектом.

Проекты разрабатываются итеративно, маленькими шагами, но с частыми интеграциями. Это позволяет заметить ошибки на ранних этапах и оперативно внести корректировки в процесс разработки/проект.

SCRUM

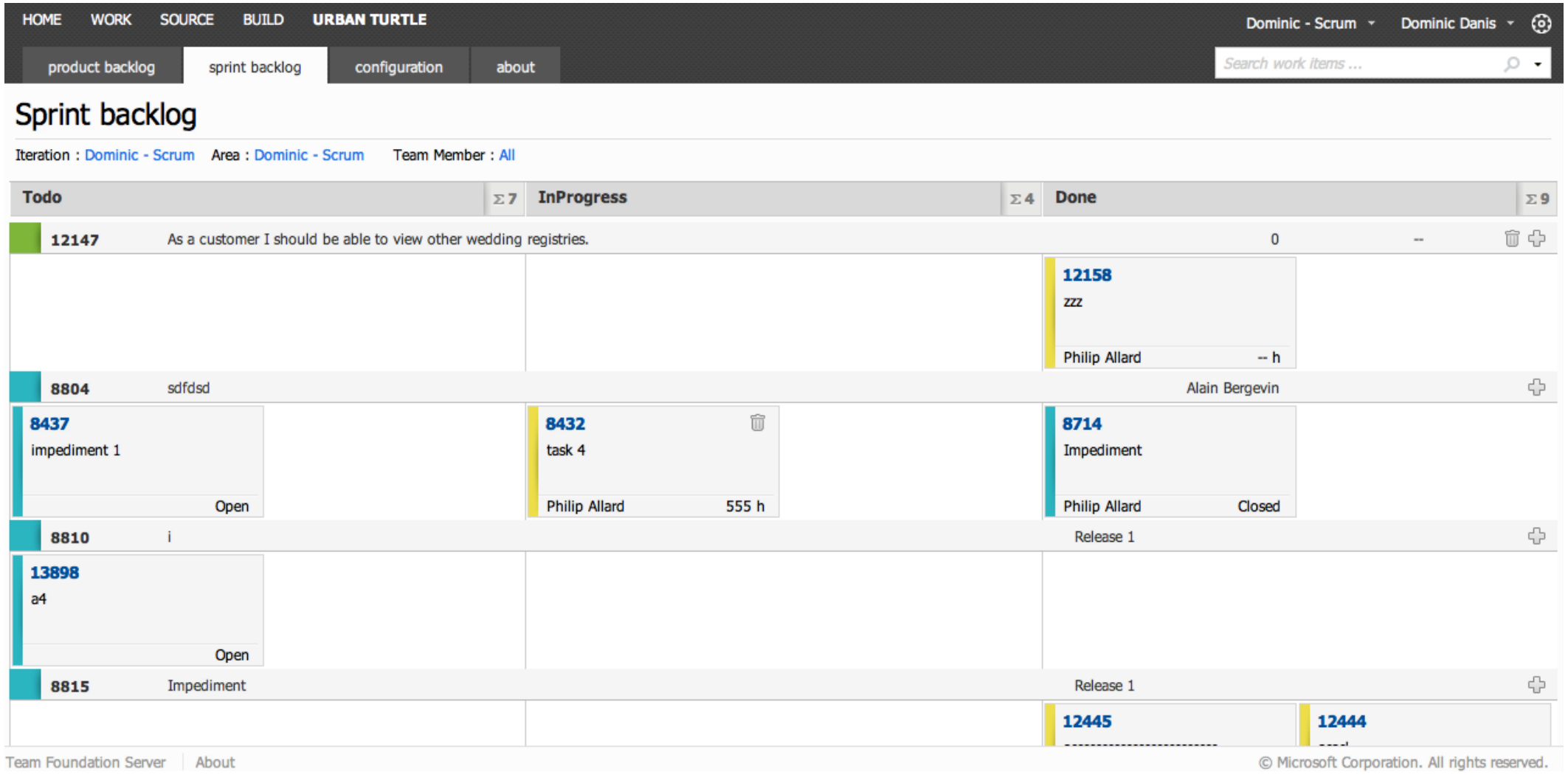


Source: Adapted from *Agile Software Development with Scrum* by Ken Schwaber and Mike Beedle.

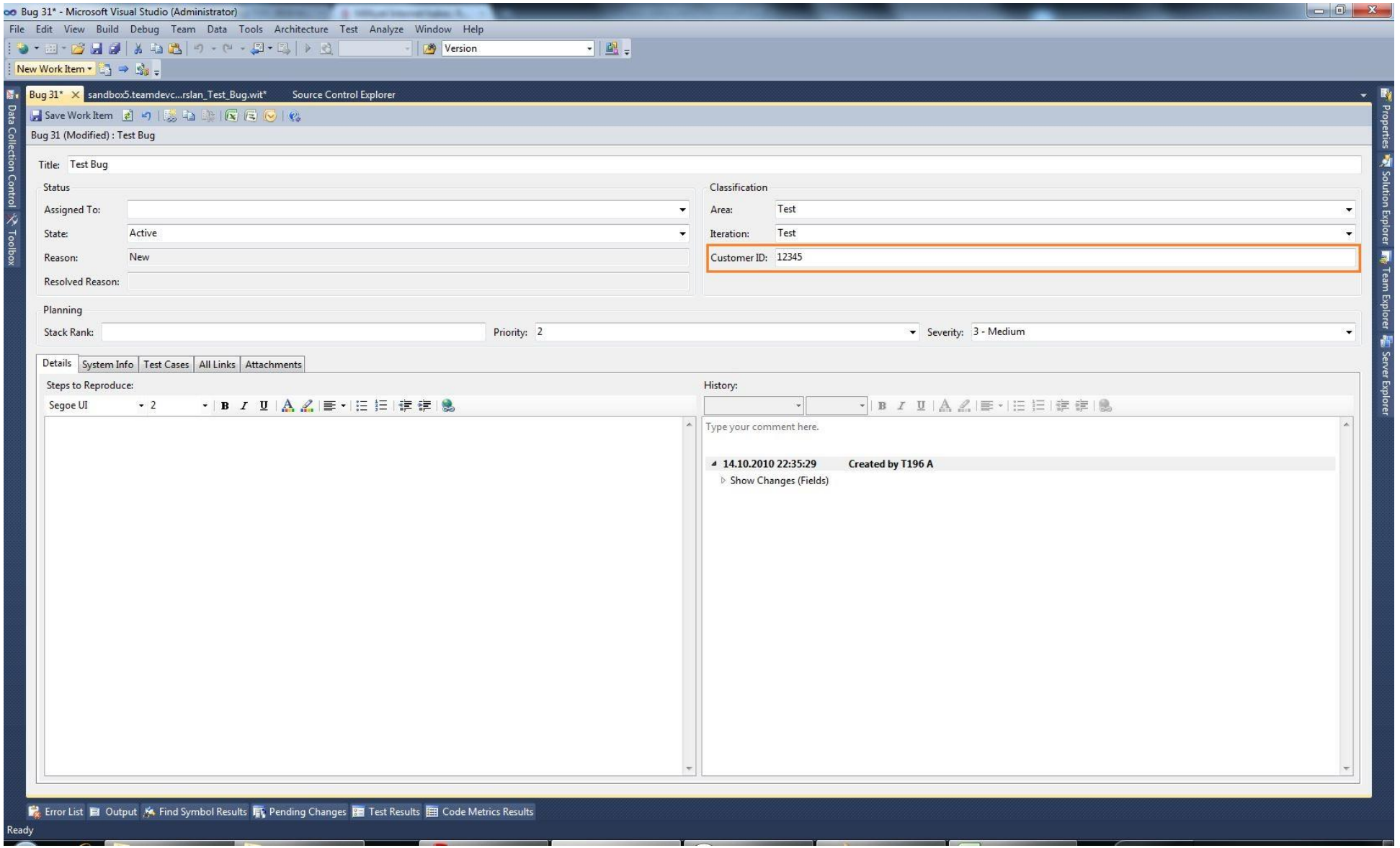
СИСТЕМЫ УПРАВЛЕНИЯ ПРОЕКТАМИ И отслеживания ошибок

- JIRA
- Bugzilla
- Trac
- YouTrack
- Redmine
- Basecamp
- TFS
- …

СИСТЕМЫ УПРАВЛЕНИЯ ПРОЕКТАМИ И отслеживания ошибок



СИСТЕМЫ УПРАВЛЕНИЯ ПРОЕКТАМИ И отслеживания ошибок



ПРОЦЕСС РАЗРАБОТКИ

- Кодирование
- Тестирование/Подготовка к развертыванию
- Багфикс (если нужен) ➔ повторное тестирование
- Развертывание

Toolchain

- OS: Linux (RHEL/CentOS) / FreeBSD
- Console tools: *nix toolchain (sh/awk/sed/…)
- Programming languages: C / C++ [gcc/clang] / Python [cpython]
- Version control: Git
- Continuous Integration: Jenkins
- Deploy: ansible, system packets (rpm/tgz)

О ПЛОХОМ И ХОРОШЕМ КОДЕ

```
if (verbosity > 1 && U->last_local_id > 1800 && E->user_id == 6492 &&
((M->flags & 199) == 0 || (U->last_local_id > 1864 && U->last_local_id < 1880)))
```

https://github.com/vk-com/kphp-kdb/blob/f9a2f927aa97612d0c74a0e296eda8f414f13cce/text/text-index.c#L1372

О ПЛОХОМ И ХОРОШЕМ КОДЕ

```
case LEV_SEQ_STORE_INF + 0x100 ... LEV_SEQ_STORE_INF + 0x3ff:
    s = E->type & 0xff;
    if (size < 8 || size < sizeof (struct lev_seq_store_inf) + 4 * s + 4 * E->a) { return -2; }
    store_inf ((void *)E);
    return sizeof (struct lev_seq_store_inf) + 4 * s + 4 * E->a;
case LEV_SEQ_STORE_TIME + 0x100 ... LEV_SEQ_STORE_TIME + 0x3ff:
    s = E->type & 0xff;
    if (size < 12 || size < sizeof (struct lev_seq_store_time) + 4 * s + 4 * E->b) { return -2; }
    store_inf ((void *)E);
    return sizeof (struct lev_seq_store_time) + 4 * s + 4 * E->b;
…
return -1;
```

https://github.com/vk-com/kphp-kdb/blob/master/seqmap/seqmap-data.c#L1029

О ПЛОХОМ И ХОРОШЕМ КОДЕ

В промышленной разработке код должен писаться с расчетом на возможную текучку в команде и его долгую поддержку при необходимости.

Соответственно код должен быть:

- Протестирован и максимально полно покрыт тестами
- Удобочитаем
- Соответствовать командным стандартам кодирования (именование переменных/функций, применение/неприменение исключений, …)
- Легок в поддержке

О ПЛОХОМ И ХОРОШЕМ КОДЕ

- Книги:
  - Идеальный программист. Как стать профессионалом разработки ПО
  - Совершенный код
  - Идеальный код
  - Чистый код
  - Джоэл о программировании
- Просмотр и разбор качественно документированных open-source проектов
- Изучение различных общепризнанных style-guide кодирования
- Опыт и постоянное расширение кругозора

О ПЛОХОМ И ХОРОШЕМ КОДЕ

Быстрый обратный квадратный корень

```
float Q_rsqrt( float number )
{
        long i;
        float x2, y;
        const float threehalfs = 1.5F;

        x2 = number * 0.5F;
        y  = number;
        i  = * ( long * ) &y;                   // evil floating point bit level hacking
        i  = 0x5f3759df - ( i >> 1 );               // what the fuck?
        y  = * ( float * ) &i;

        y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration
//      y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this can be removed
        return y;
}
```

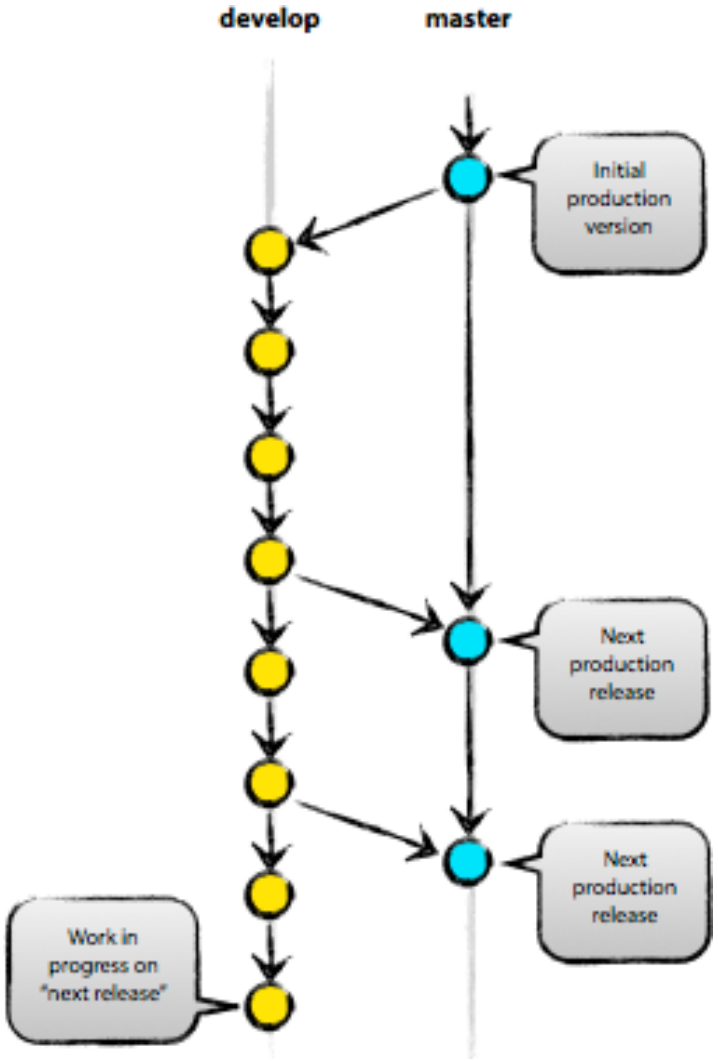
СИСТЕМЫ УПРАВЛЕНИЯ ВЕРСИЯМИ

Система управления версиями (от англ. Version Control System, VCS или Revision Control System) — программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

СИСТЕМЫ УПРАВЛЕНИЯ ВЕРСИЯМИ

- CVS
- Bazaar
- SVN
- Git
- Mercurial
- Perforce
- TFS
- ….

СИСТЕМЫ УПРАВЛЕНИЯ ВЕРСИЯМИ



Юнит Тестирование

Модульное тестирование, или юнит-тестирование (англ. unit testing) — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Юнит Тестирование

- Каждый добавленный кусок кода обязательно покрывается тестами.
- Тесты должны успешно отрабатывать.
- Замеряется функциональное (для каждого ли из методов в коде есть вызывающий его тест) и условное покрытие кода тестами (проход всех возможных условных веток, покрытие всех крайних условий).
- Суммарное покрытие проекта тестами не должно падать ниже фиксированного значения.

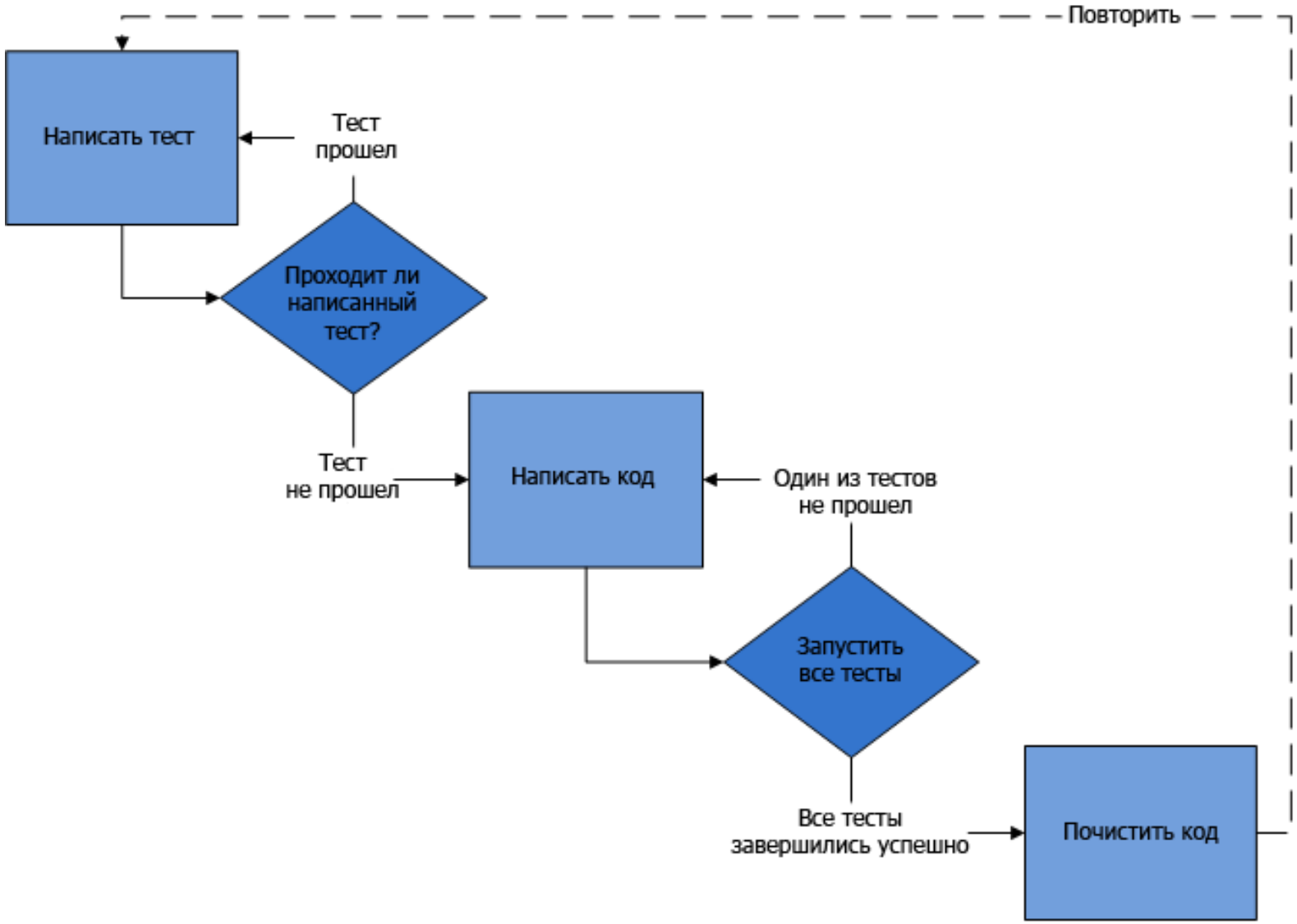
Юнит Тестирование

- C++: google-mock, google-test, …
- Python: pytest, pyunit, …

Разработка Через Тестирование

Разработка через тестирование (англ. test-driven development, TDD) — техника разработки программного обеспечения, которая основывается на повторении очень коротких циклов разработки: сначала пишется тест, покрывающий желаемое изменение, затем пишется код, который позволит пройти тест, и под конец проводится рефакторинг нового кода к соответствующим стандартам.

Разработка Через Тестирование



КОД РЕВЬЮ

Просмотр кода (англ. code review) или *инспекция кода* (англ. code inspection) — систематическая проверка исходного кода программы с целью обнаружения и исправления ошибок, которые остались незамеченными в начальной фазе разработки.

Каждый коммит до попадания в репозиторий рассматривается одним (или несколькими) разработчиками с целью поиска в нем очевидных ошибок, нарушения стилистики кода либо неудачных решений.

Повышает качество кода, улучшает обмен опытом в команде, тонизирует.

СТАТИЧЕСКИЙ АНАЛИЗ КОДА

Статический анализ кода (англ. static code analysis) — анализ программного обеспечения, производимый (в отличие от динамического анализа) без реального выполнения исследуемых программ.

СТАТИЧЕСКИЙ АНАЛИЗ КОДА

Позволяет находить:

- Ошибки, связанные с неопределенным поведением (неинициализированные переменные)
- Ошибки, связанные с известными недокументированными побочными эффектами
- Переполнение буфера
- Повторяющийся код
- Ненужный код
- Функционал, связанный с возможными оптимизациями компилятора
- Некросплатформенный код
- …

СТАТИЧЕСКИЙ АНАЛИЗ КОДА

```
const uint32 kRedCoefficient = 2125;
const uint32 kGreenCoefficient = 7154;
const uint32 kBlueCoefficient = 0721;
const uint32 kColorCoefficientDenominator = 10000;
```

СТАТИЧЕСКИЙ АНАЛИЗ КОДА

```
const uint32 kRedCoefficient = 2125;
const uint32 kGreenCoefficient = 7154;
const uint32 kBlueCoefficient = 0721;
const uint32 kColorCoefficientDenominator = 10000;
```

СТАТИЧЕСКИЙ АНАЛИЗ КОДА

```
void doSomething(const char* x)
{
    char s[40];
    sprintf(s, "[%s]", x);
}
```

СТАТИЧЕСКИЙ АНАЛИЗ КОДА

```
void doSomething(const char* x)
{
    char s[40];
    sprintf(s, "[%s]", x);
}
```

СТАТИЧЕСКИЙ АНАЛИЗ КОДА

- C++: cppcheck, PVS Studio / CppCat, …
- Python: pychecker, pylint, pyflakes

q.viva64.com

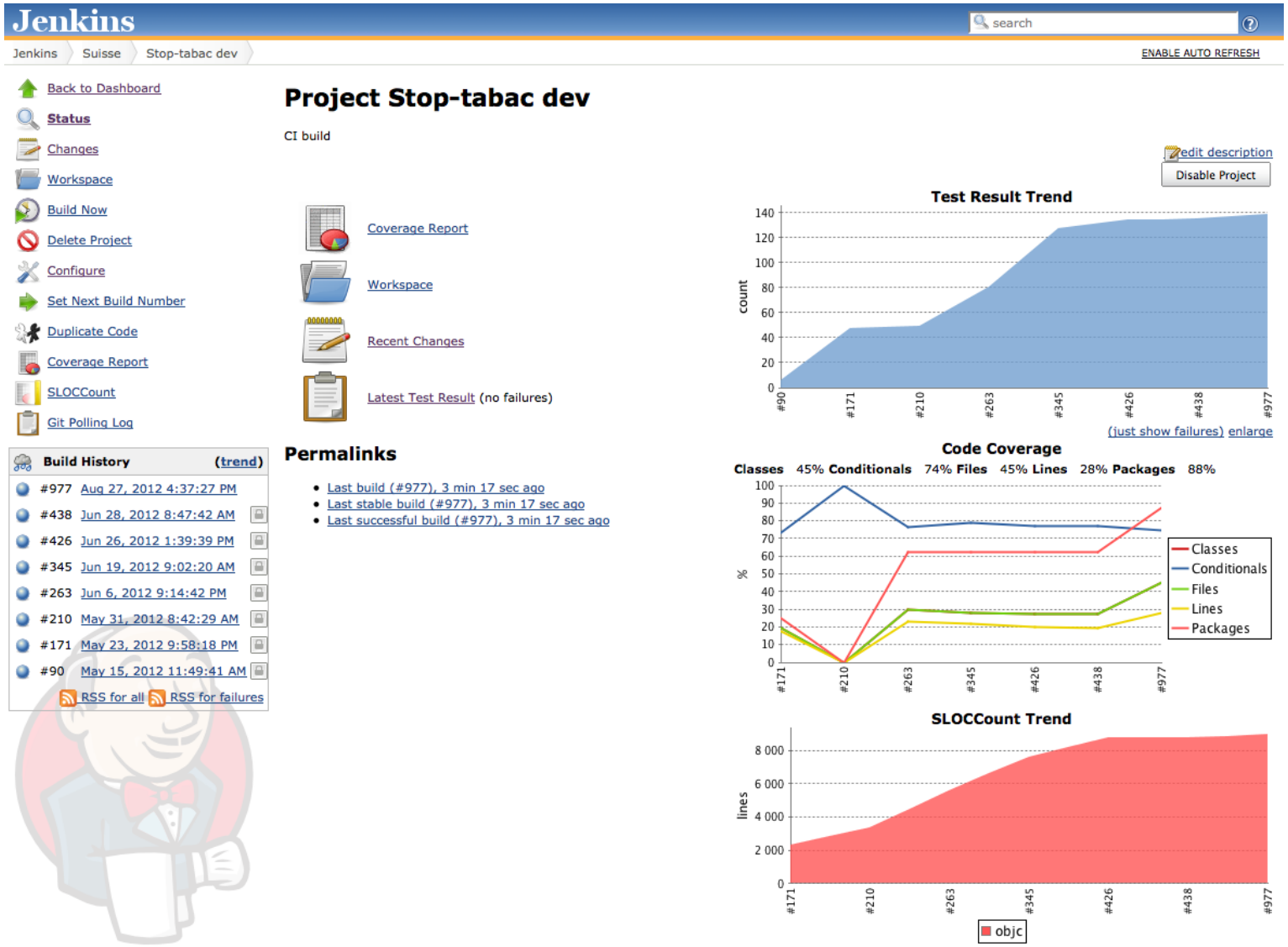
НЕПРЕРЫВНАЯ ИНТЕГРАЦИЯ

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

НЕПРЕРЫВНАЯ ИНТЕГРАЦИЯ

- Hudson
- TeamCity
- Jenkins
- BuildBot
- Travis

НЕПРЕРЫВНАЯ ИНТЕГРАЦИЯ



НЕПРЕРЫВНАЯ ИНТЕГРАЦИЯ

- Проект забирается из указанного репозитория
- Собирается в указанном окружении
- После сборки проходит несколько шагов:
  - Проверка на прохождение тестов
  - Проверка на покрытие тестами
  - Проверка статическим анализатором кода
  - Проверка на возможные утечки памяти (динамический анализ с помощью valgrind)
- В зависимости от статуса сборки (успешна/нет) – выполнение дополнительных шагов (рассылка информационных писем о сборке, автоматический деплой, любые другие дополнительные действия)

РАЗВЕРТЫВАНИЕ

- Пакеты (rpm/tgz)
- Ansible – платформа для удаленного развертывания ПО, выполнения задач и конфигурирования. Чистый python + ssh.

РАБОТА ТЕСТИРОВЩИКОВ

Любыми способами сломать то, что мы им отдали, используя метод “черного ящика”.

Для каждого проекта есть утвержденное описание всех его тест-кейсов, полностью проверяется его функциональность.

Тестировщик нашел ошибку – заводится баг в TFS, разработчик забирает задачу, чинит баг, тестер проверяет заново. Повторять до получения результата.

МОНИТОРИНГ СЕРВИСОВ

В режиме реального времени постоянно производится мониторинг всех сервисов на предмет их работоспособности.

В случае отказа чего-либо – рассылаются уведомления людям, связанным с работой этого сервиса.

МОНИТОРИНГ СЕРВИСОВ

- Nagios
- Zabbix
- Cacti
- Icinga

Вопросы?

Y U NO WANT THESE LECTIONS

- Map-Reduce, Message Queues: Hadoop, своя распределенная реализация на базе ZeroMQ
- Selenium, автоматическое тестирование сторонних продуктов
- NoSQL (Cassandra)