itseez

Александр Сморкалов

Автоматическое тестирование в Computer Vision

26 марта 2016 г.

План

- Intro: процесс разработки в CV
- QA инструменты
- QA сервисы
- CI
- Анализ

Задача

Разработка библиотек компьютерного зрения и пользовательских приложений на их базе для широкого спектра платформ и ОС с полной интеграцией в платформу

Аппаратные платформы: x86+extra, arm-v7a+extra, arm-v8a+extra

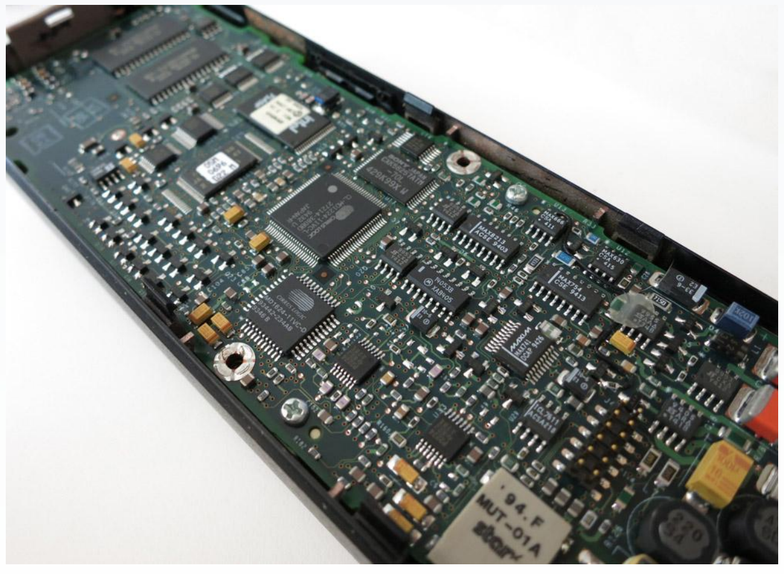
Программные платформы: Windows, Linux, Android*, QNX*.

Часть 1

Intro: процесс разработки в CV

Процесс разработки CV приложения

- Исследование
- PoC
- Стабилизация
- Портирование
- Профилировка
- Оптимизация
- Продуктизация

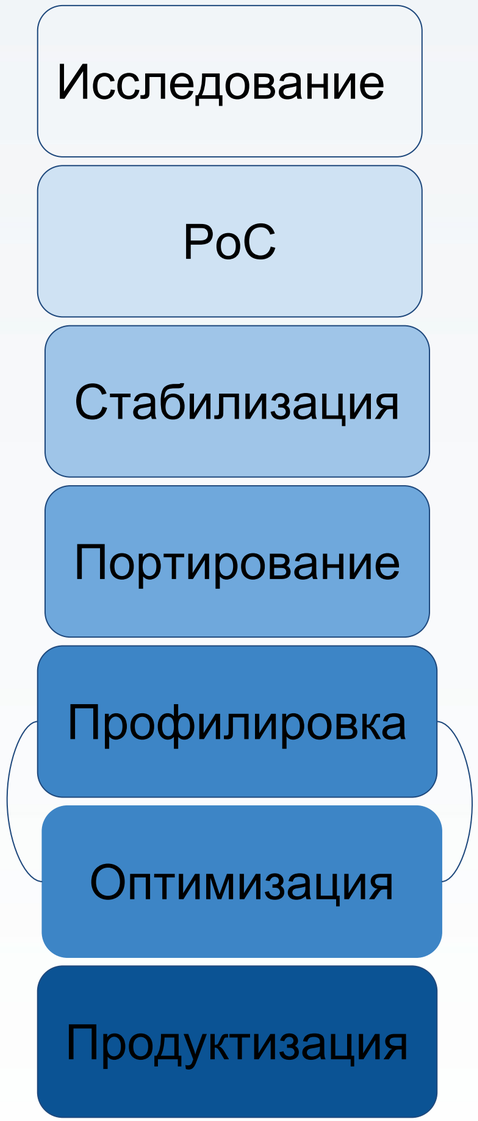


Изображения с http://cdn.durdom.com/ibm-simon/lower-part-case-angle.jpg и http://www.hasandilsiz.com.tr/wp-content/uploads/ibm-pc.jpg

Процесс разработки - инструменты

- Хранение кода: *Git*
- Исследования: *MatLab, Python, C++*
- Построение:
  - *CMake* + *make* для Linux
  - *CMake* + *ninja* / *MS Build* для Windows
- Тестирование: Расширенный *GTest*
- Пакеты и инсталяторы: Расширенный *CPack*

Процесс разработки - в профиль



Тест для оценки качества алгоритма

Тест для оценки качества алгоритма

Тесты на точность, регрессионные тесты

Тесты производительности

Интеграционные и приёмочные тесты

Постулаты

- Высокое качество != качественное тестирование релизов. Качество закладывается на каждом этапе
- Процессы разработки и тестирования в команде должны быть предсказуемыми и на 100% повторимыми
- Тесты должны быть предсказуемыми и на 100% повторимыми
- Максимально эффективная автоматизация. Автоматизация — метод повышения производительности

Часть 2

QA инструменты

C/C++ компилятор

- Кодирование без предупреждений
- Максимально строгая проверка на уровне компилятора: *-Wall* и не только

Планы:

- *Clang* & *GCC* санитайзеры

Тесты

- *Gtest* — базовый фреймворк для тестов
- Тесты полностью детерминированы, результат не зависит от порядка тестов*:
  - Явное управление генерацией случайных чисел
  - Внимание к многопоточности
- Точечные проверки на аккуратность во всех тестах производительности
- *Gcov+Lcov* — инструмент анализа покрытия
  - Нет цели получить 100% покрытие
  - Используется для поиска больших непокрытых кусков и особых случаев

Статический анализ

- *Vera*++ — утилита проверка стиля, форматирования, базовых конструкций
- *Cppcheck* — полноценный статический анализатор

Run-time анализ

- *Valgrind* — набор утилит для проверки обращений к памяти
- *CUDA-memcheck* — набор утилит для проверки обращений к памяти для GPU
- Стресс-тестирование*

Тестирование сценариев

- *V4l2loopback* — модуль ядра эмулирующий работу V4L2 устройств, в том числе web-камер
- *GStreamer* — фреймворк для декодирования видео, доступа к камерам и проч. Позволяет конфигурировать конвейер, использовать тестовые источники данных, например videotestsrc
- Тестирование предзаписанных сценариев поведения пользователя с использованием функциональности самого приложения:
  - Алгоритм детерминирован -> одинаковый результат при одинаковом вводе
  - PNG вместо JPEG и видео для предсказуемости

Документация

- *Doxygen* со строгой проверкой предупреждений
- Компиляция всех примеров и сниппетов кода
  - Все кусочки кода в документации сведены в C/C++ файл и компилируются вместе с проектом
  - В качестве примера используется тело некоторых тестов
- *Linkchecker* — утилита проверки ссылок для консистентной документации

Пакетирование

- Платформы:
  - *CPack -> *.deb* для Linux
  - *CPack+NSIS -> *.exe* для Windows
  - *CPack -> *.zip* для Android, etc
- Тесты — релиз артефакт и пакуются как основной продукт
- Тестирование из пакетов: тесты запускаются так же, как на стороне пользователя

Пакетирование

- *Lintian* — инструмент проверки Debian пакетов на соответствие требованиям платформы
- *dpkg-sig* — утилита проверки цифровых подписей
- *ABI compliance checker* — улилита проверки бинарной совместимости и совместимости на уровне исходного кода
- *Docker* как средство проверки пакетной совместимости
  - Установка в “чистой комнате”
  - Установка совместно с популярными пакетами

Часть 3

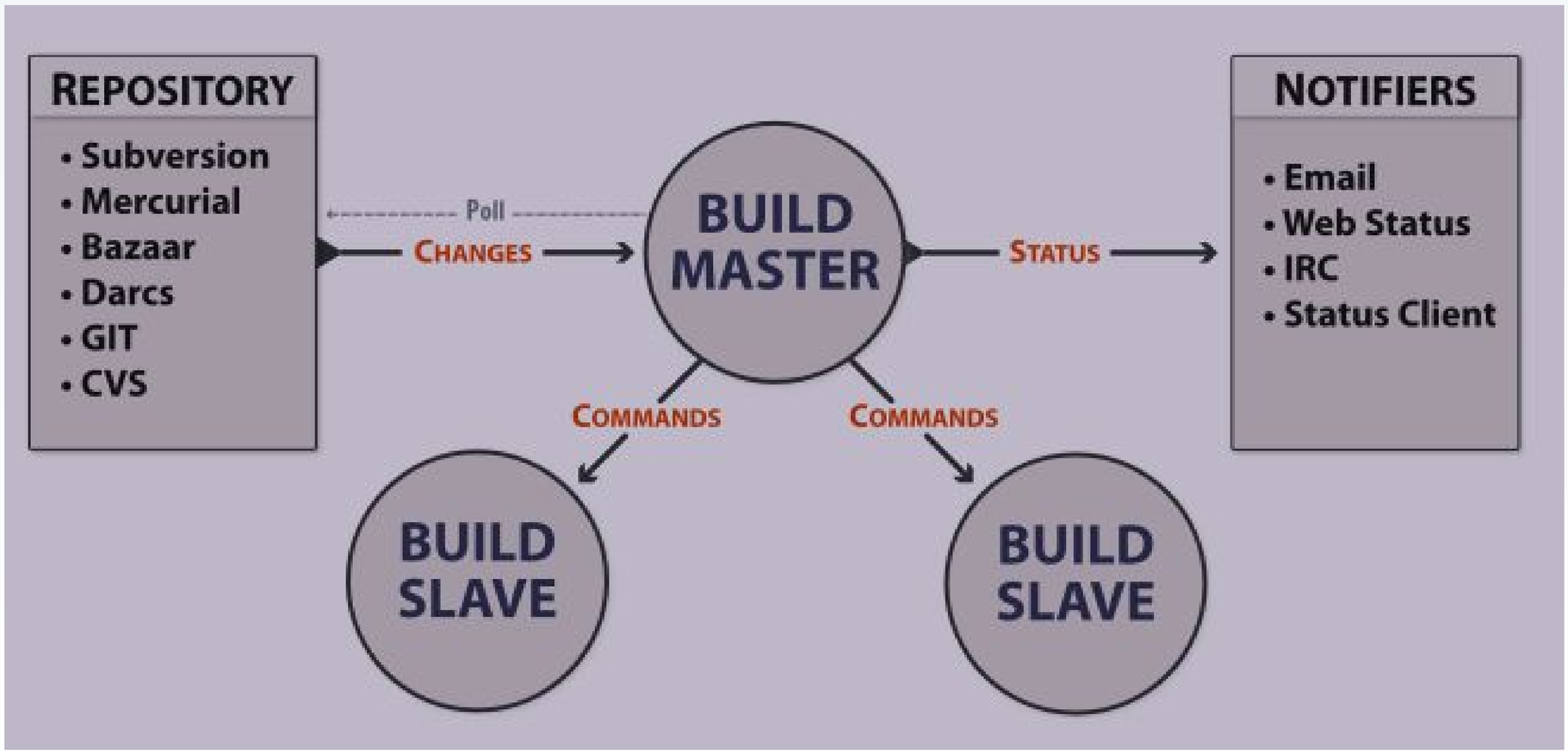
QA сервисы

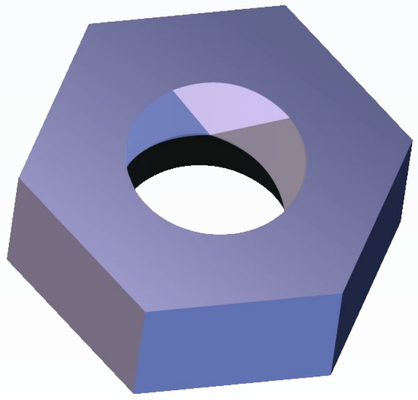
GitLab

- Хостинг для *Git*
- Инструмент управления проектами и пользователями
- Инструмент ревью кода и непрерывной интеграции

BuildBot

- Кросс-платформенный CI фреймворк на *Python*
- Построен на основе *Twisted*





Изображение с http://buildbot.net/#/basics

Сервис pre-commit тестирования

- Автоматическое тестирование всех запросов на интеграцию (MR, PR) параллельно с ревью
- Интеграция *BuildBot* с *Gitlab* & *Github*
- Алгоритм: merge -> build -> test

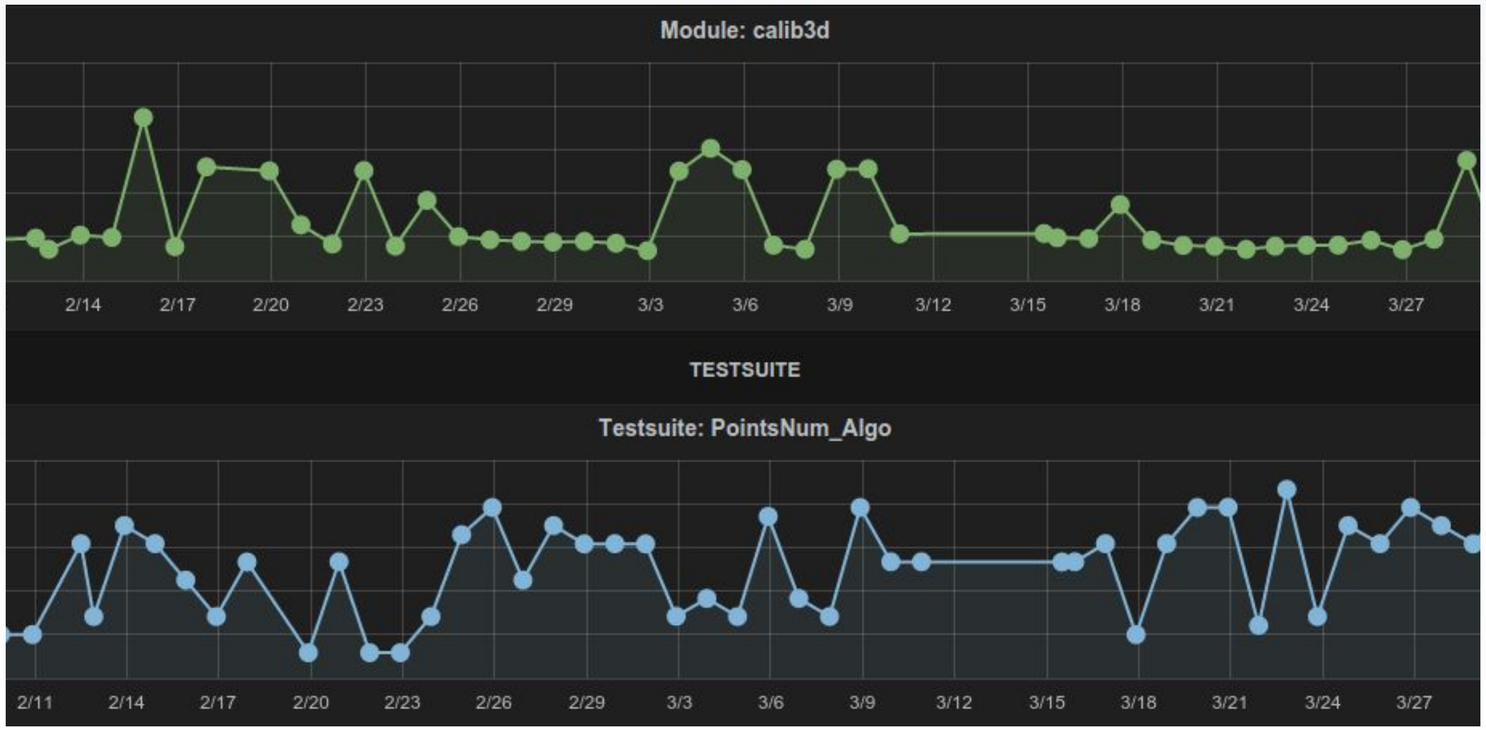
Active pull requests (61) GitHub OpenCV repo

Main Buildbot

Id	Author	Title & description	Linux x64	Win7 x64 VS2013	Win10 x64 VS2015	Mac	Android armeabi-v7a	OpenCL	OpenCL Intel	Linux x64 Debug	Docs	iOS
filter	filter	PR name filter	Builder filter									
6329 master	paroj / None	highgui: add CV_GUI_* flags to enum in cpp so they appear in bindings Required builds passed	5675 success	5246 success	988 success	5152 success	5396 success	5135 success	not_queued	3490 success	5487 success	1785 success
6327 2.4	terfendail / None	Added randomization seed initialization to feature_homography python … Required builds passed **What does this PR change?** findHomography use RANSAC so we have to reset random seed to ensure test stability.	5674 success	5245 success	987 success	5151 success	5395 success	5134 success	not_queued	3489 success	5486 success	1784 success
6326 master	sovrasov / None	Replace of some synthetic scenes in python samples Required builds passed **What does this PR change?** Replace some synthetic scenes in python samples to more informative	5673 success	5244 success	986 success	5150 success	5394 success	5133 success	not_queued	3488 success	5485 success	1783 success

Сервис анализа производительности

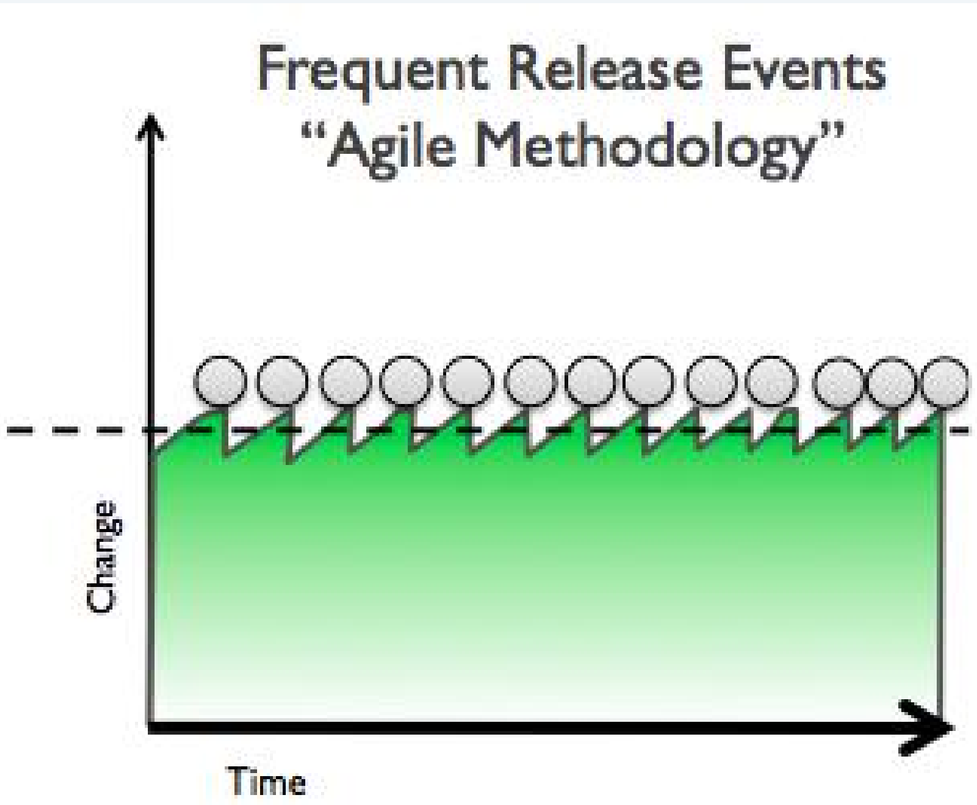
- *InfluxDB* + *Grafana* + *BuildBot*
- История всех замеров производительности
- Автоматический поиск регрессий

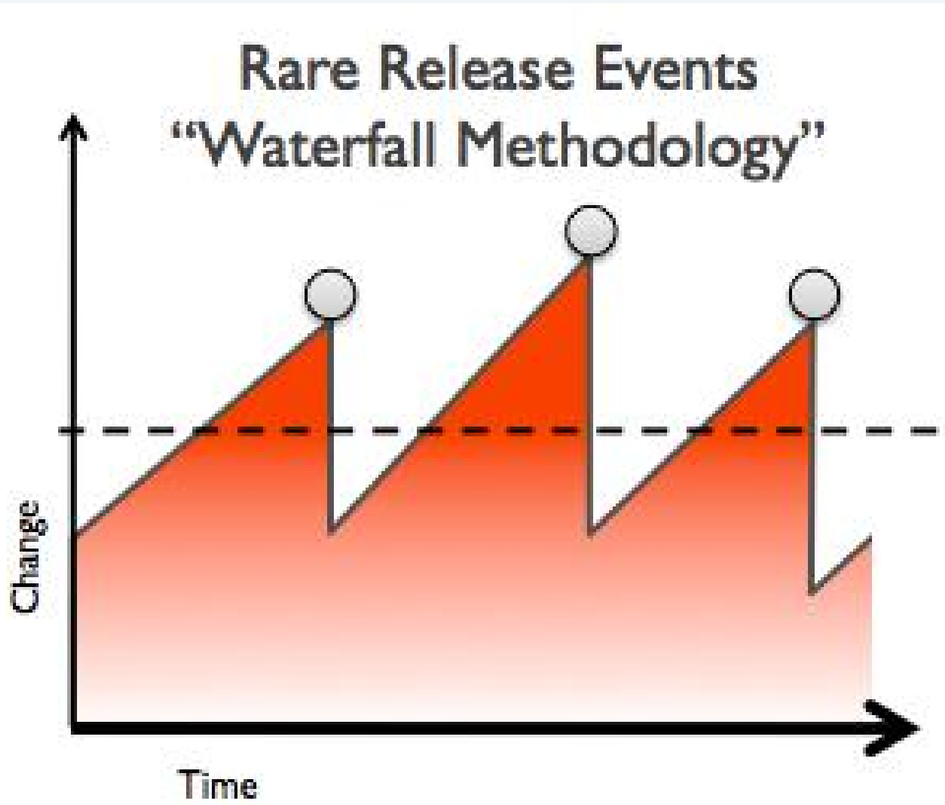


Часть 4

In CI we trust

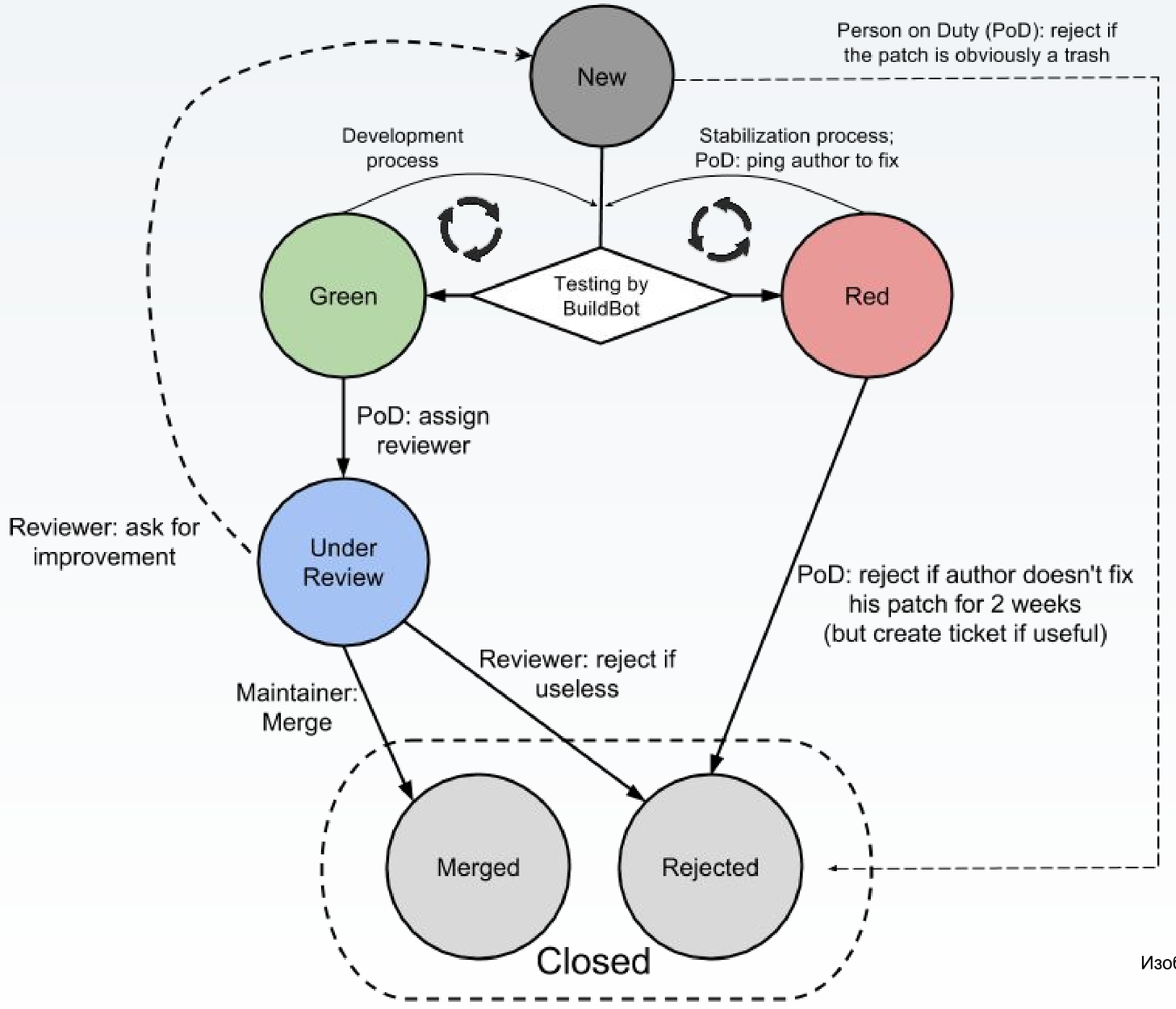
Agile





Изображение с https://upload.wikimedia.org/wikipedia/commons/1/1c/Agile-vs-iterative-flow.jpg

Шаг 1. Разработка новой функциональности



Изображение с https://github.com/Itseez/opencv/wiki/How_to_contribute

Шаг 2. Ежедневная работа

- Проверка и анализ статуса ночных и других регулярных билдов
- Анализ результатов тестов производительности
- Проверка состояния оборудования и инфраструктуры
- Проверка известных проблем, поиск связей и закономерностей при падении
- Поиск ошибок в процедурах построения

Шаг 3. Релиз

- Запуск всех билдов для построения артефактов на BuildBot
- Запуск всех вспомогательных билдов: отчёты о покрытии, производительнсти, etc
- Ручное тестирование базовых сценариев
- Выкладка артефактов для релиза

Часть 5

Анализ

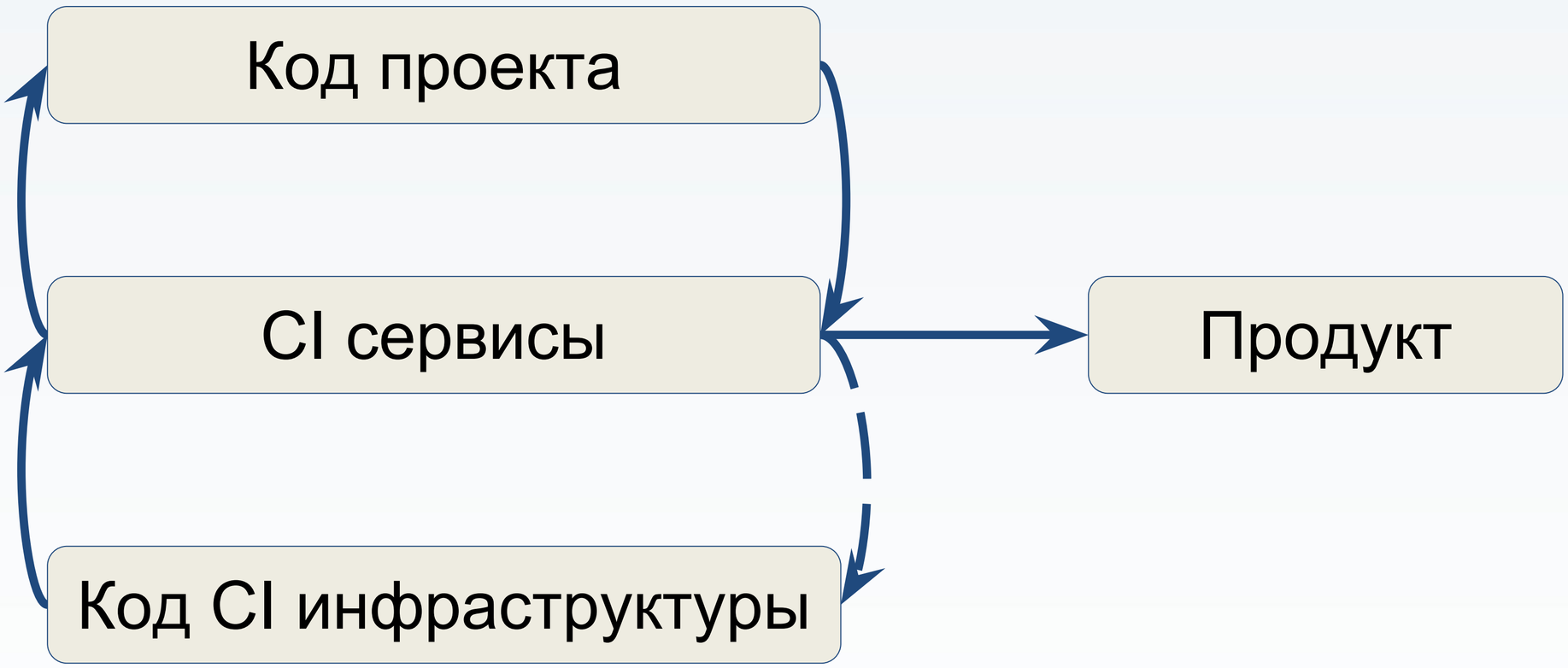
Профит

- Устойчивый и масштабируемый процесс
- Масштабируемая поддержка платформ
- Возможно расширение тестового покрытия без постоянного привлечения ресурсов
- Быстрое детектирование ошибок, в том числе в платформах
- Нет выделенного QA инженера. Тесты реализует разработчик функции
- Один DevOps инженер, он же ментейнер инфраструктуры

Trust?

- Время разработчиков заменяется машинным временем. В больших проектах требуется существенное количество оборудования
- Регулярное тестирование и code review занимают существенное время, что может сказаться на производительности команды
- Необходима устойчивая работа CI инфраструктуры
- Комада ДОВЕРЯЕТ инструментам и результатам CI
- Необходима регулярная проверка тестов и умное тестирование
- Необходима регулярная проверка сборочных процедур и умная автоматизация

Тестирование инфраструктуры



Инструменты для инфраструктуры

- *Docker* обеспечивает переносимые сервисы и конфигурации
- *Ansible* обеспечивает автоматическое развертывание
- Разработка инфраструктуры по тому же процессу, что и разработка основного продукта

Ссылки

- Сервис pre-commit тестирования: http://pullrequest.opencv.org/#/summary/
- Код сервиса: https://github.com/alalek/buildbot-pullrequest-sample
- Расширенный GTest: https://github.com/Itseez/opencv/tree/master/modules/ts

itseez future you!

Спасибо за внимание!