

Непрерывный статический анализ кода

Иван Пономарёв, КУРС/МФТИ

ponomarev@corchestra.ru

 [@inponomarev](https://twitter.com/inponomarev)

Льём в прод



Кодим



Delivery Pipeline



Покупайте лучшие анализаторы!



Что такое статанализ?

Что такое статанализ?

- Wikipedia: «Анализ программного обеспечения, производимый без реального выполнения исследуемых программ».

Что такое статанализ?

- Wikipedia: «Анализ программного обеспечения, производимый без реального выполнения исследуемых программ».
- Здравый смысл: Любая проверка исходного кода, требующая только исходный код (без тестов).

Чего в принципе не может СА?

Чего в принципе не может СА?

Зависнет или остановится?

```
def halts(f):  
    # false, если программа зависает  
    # true, если завершается за конечное время
```


Чего в принципе не может СА?

Зависнет или остановится?

```
def halts(f):  
    # false, если программа зависает  
    # true, если завершается за конечное время  
def g():  
    if halts(g):  
        while True:  
            pass
```

Теорема Райса

Вычисляет ли функция квадрат числа?

```
def is_a_squaring_function(f):  
    # true, если функция вычисляет квадрат  
    # false, если не вычисляет
```


Теорема Райса

Вычисляет ли функция квадрат числа?

```
def is_a_squaring_function(f):  
    # true, если функция вычисляет квадрат  
    # false, если не вычисляет  
  
def halts(f):  
    def t(n):  
        f()  
        return n * n  
    return is_a_squaring_function(t)
```

Статанализ не найдёт

- `Все object has no attribute` (если динамическая типизация)

Статанализ не найдёт

- Все object has no attribute (если динамическая типизация)
- Все NPE (если нет Null Safety)

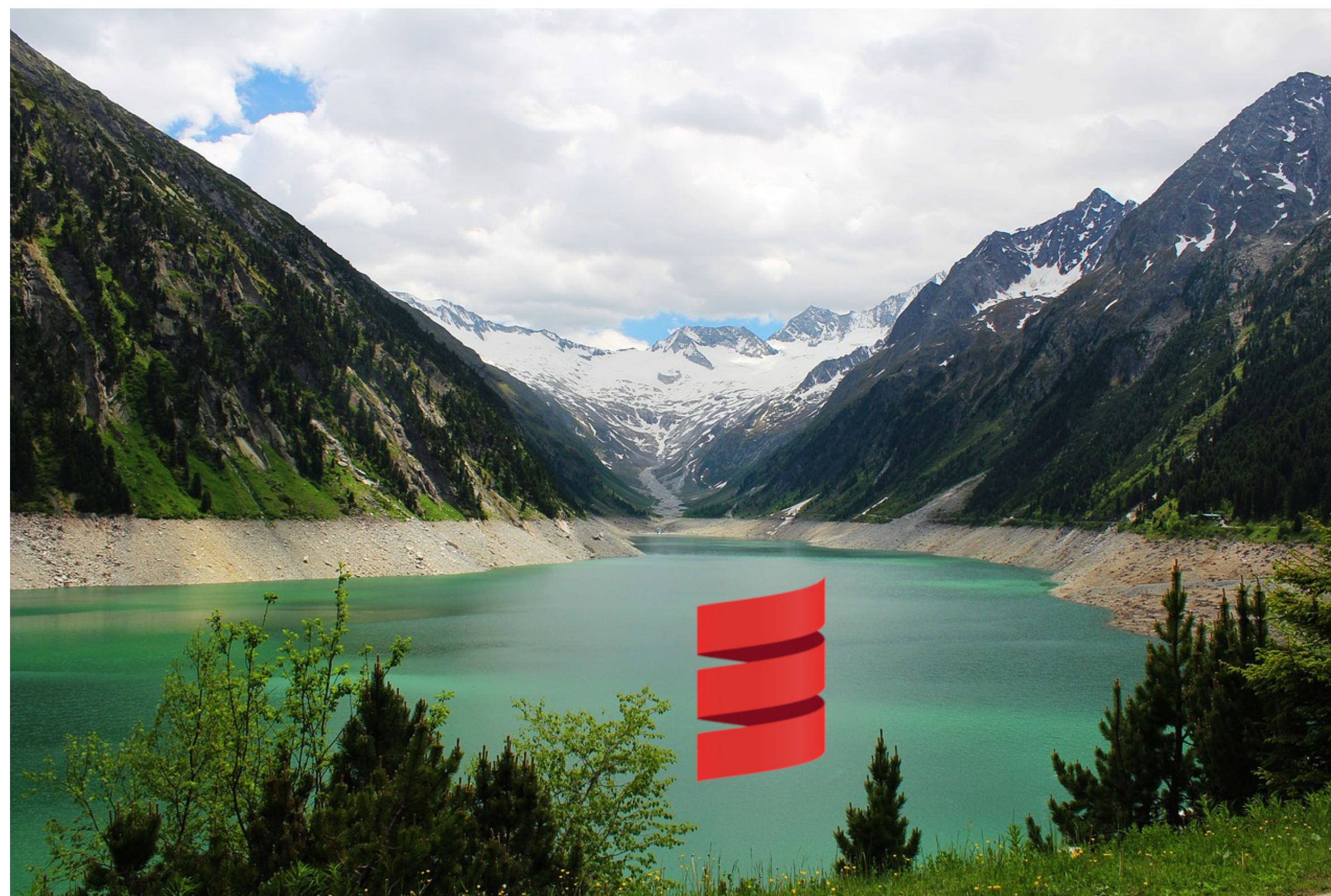
Статанализ не найдёт

- Все object has no attribute (если динамическая типизация)
 - Все NPE (если нет Null Safety)
- ...не говоря уж о менее тривиальных вещах.

Чистый горный источник

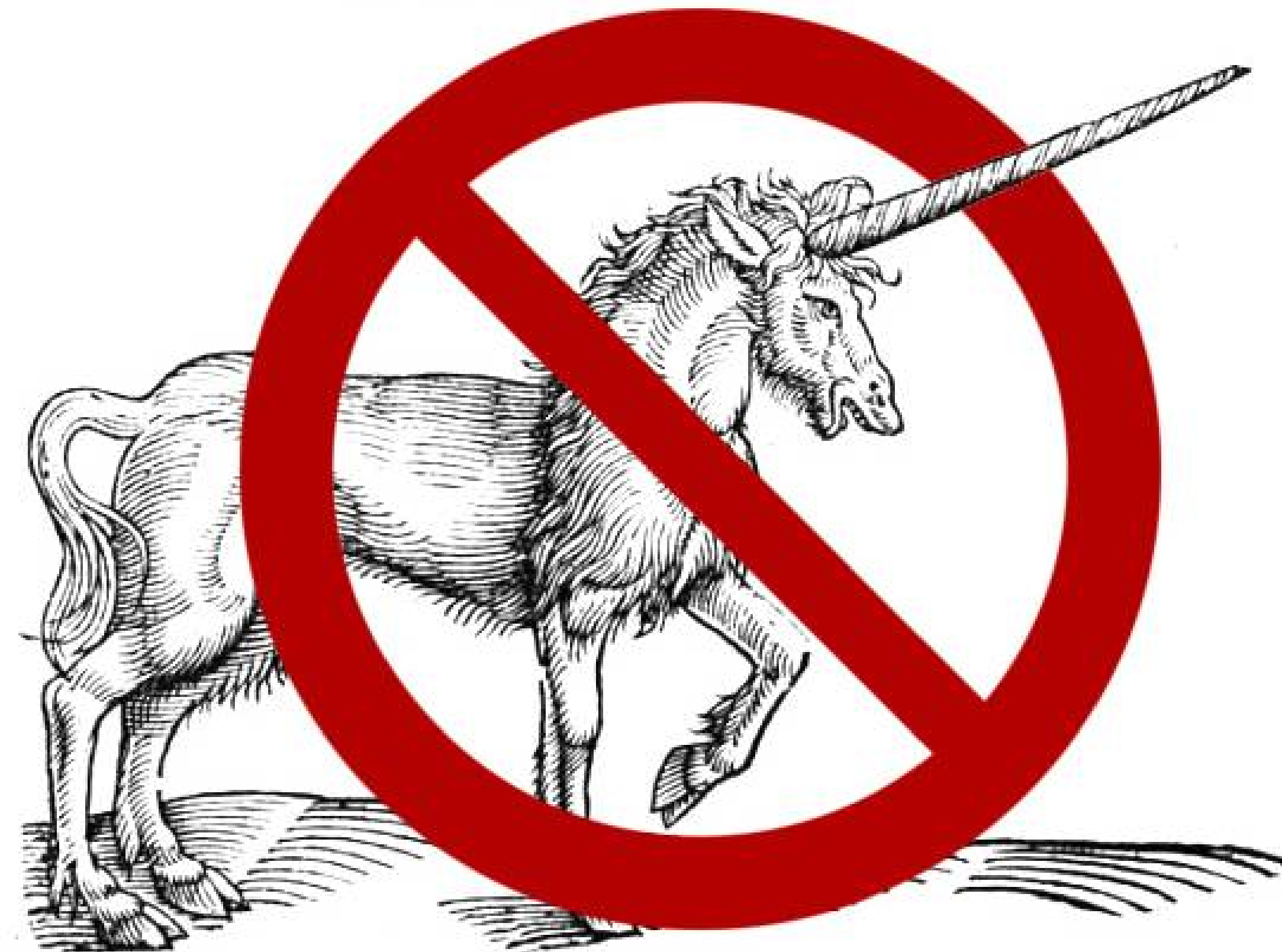


Чистый горный источник



А что же статанализ?

Никакого волшебства!



Обыкновенная лошадь



Разнообразие средств статанализа

Разнообразие средств статанализа

- Проверка стиля кодирования (checkstyle, flake8)

Разнообразие средств статанализа

- Проверка стиля кодирования (checkstyle, flake8)
- Поиск характерных ошибок в коде (spotbugs, IDEA, PVS-Studio)

Разнообразие средств статанализа

- Проверка стиля кодирования (checkstyle, flake8)
- Поиск характерных ошибок в коде (spotbugs, IDEA, PVS-Studio)
- Проверка валидности ресурсных файлов (xmllint, YAMLLint, JSONLint)

Разнообразие средств статанализа

- Проверка стиля кодирования (checkstyle, flake8)
- Поиск характерных ошибок в коде (spotbugs, IDEA, PVS-Studio)
- Проверка валидности ресурсных файлов (xmllint, YAMLLint, JSONLint)
- Компиляция/парсинг (`ansible --syntax-check`, `terraform validate`)

Разнообразие средств статанализа

- Проверка стиля кодирования (checkstyle, flake8)
- Поиск характерных ошибок в коде (spotbugs, IDEA, PVS-Studio)
- Проверка валидности ресурсных файлов (xmllint, YAMLLint, JSONLint)
- Компиляция/парсинг (`ansible --syntax-check`, `terraform validate`)
- Предупреждения компиляторов

Разнообразие средств статанализа

- Проверка стиля кодирования (checkstyle, flake8)
- Поиск характерных ошибок в коде (spotbugs, IDEA, PVS-Studio)
- Проверка валидности ресурсных файлов (xmllint, YAMLLint, JSONLint)
- Компиляция/парсинг (`ansible --syntax-check`, `terraform validate`)
- Предупреждения компиляторов
- Проверка правописания

Разнообразие средств статанализа

- Проверка стиля кодирования (checkstyle, flake8)
- Поиск характерных ошибок в коде (spotbugs, IDEA, PVS-Studio)
- Проверка валидности ресурсных файлов (xmllint, YAMLLint, JSONLint)
- Компиляция/парсинг (`ansible --syntax-check`, `terraform validate`)
- Предупреждения компиляторов
- Проверка правописания
- Конфигурационные тесты

Больше анализаторов!

- Open Google
- <Your Language> static analyzer
- <Your Language> linter

Кто программирует на Bash?

koalaman / shellcheck

Watch 367 Star 14,503 Fork 734

Code Issues 400 Pull requests 7 Projects 0 Wiki Insights

ShellCheck, a static analysis tool for shell scripts <https://www.shellcheck.net>

1,293 commits 4 branches 21 releases 76 contributors GPL-3.0

```
aaronkilik@tecmint ~/bin $ shellcheck test.sh
```

```
In test.sh line 24:
```

```
E_NOTROOT=50
```

```
^-- SC2034: E_NOTROOT appears unused. Verify it or export it.
```

```
In test.sh line 25:
```

```
E_MINARGS=100
```

```
^-- SC2034: E_MINARGS appears unused. Verify it or export it.
```

```
In test.sh line 30:
```

```
echo $E_NONROOT
```

```
^-- SC2153: Possible misspelling: E_NONROOT may not be assigned, but E_NOTROOT is.
```

```
^-- SC2086: Double quote to prevent globbing and word splitting.
```

```
aaronkilik@tecmint ~/bin $
```

IntelliJ IDEA в текстовой консоли

- `bin/format.sh` — форматирование кода
- `bin/inspect.sh` — инспекции (с выводом в .xml)

Запускаем инспекции IntelliJ IDEA на Jenkins

Что из этого использовать?

Что из этого использовать?

Всё!

Как это всё использовать?

Как это всё использовать?

- Однократное применение анализа бессмысленно

Как это всё использовать?

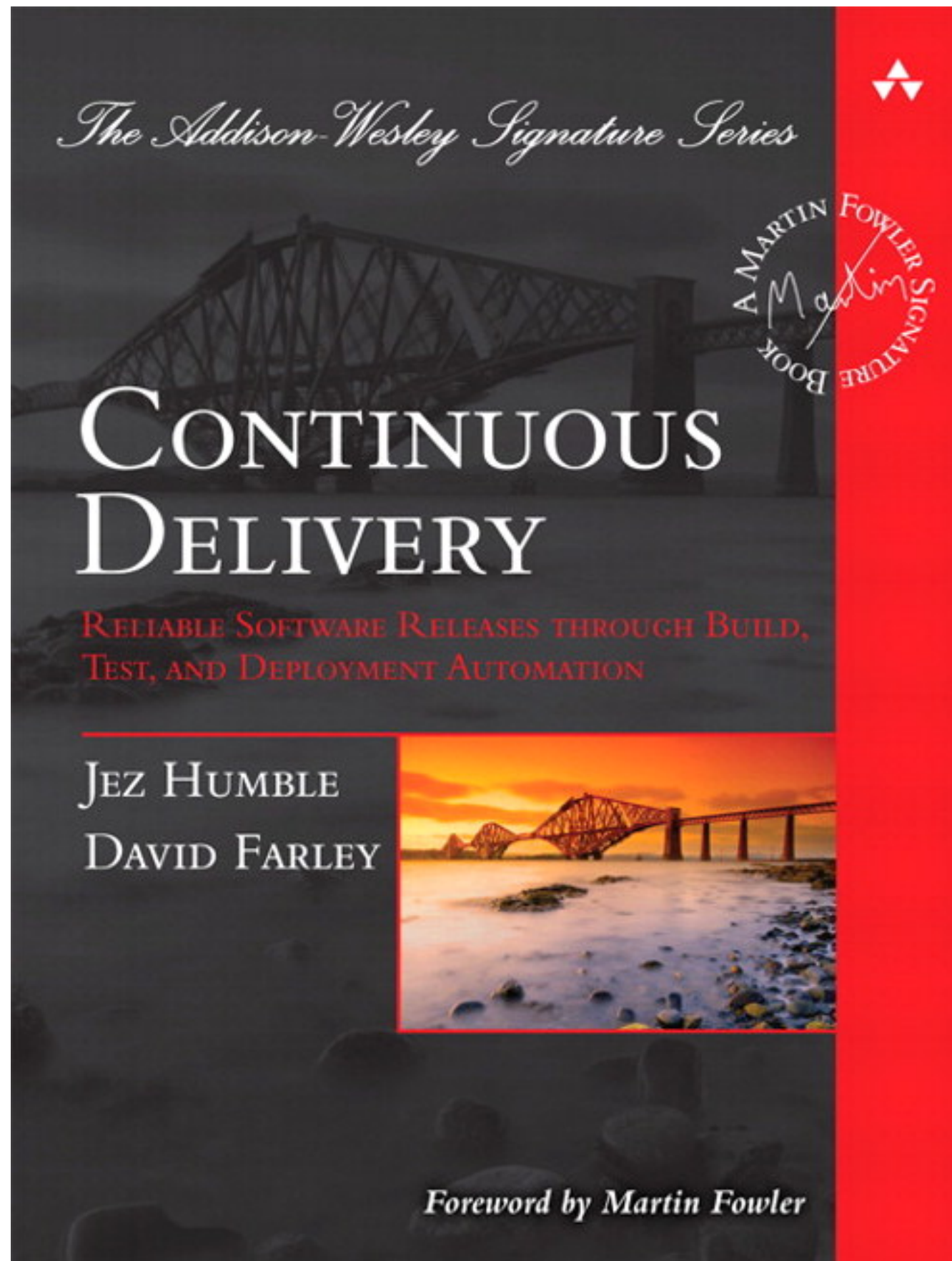
- Однократное применение анализа бессмысленно
- Анализ должен производиться **непрерывно** и **автоматически**

Как это всё использовать?

- Однократное применение анализа бессмысленно
- Анализ должен производиться **непрерывно** и **автоматически**
- Результаты анализа должны определять **quality gates**

Роль и место СА в конвейере поставки

Continuous Delivery Book



Jez Humble, David Farley. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley, 2011

Типовой конвейер сборки



Типовой конвейер сборки

«Фильтрующая способность»



Типовой конвейер сборки

Сложность, стоимость,
время работы, вероятность сбоя



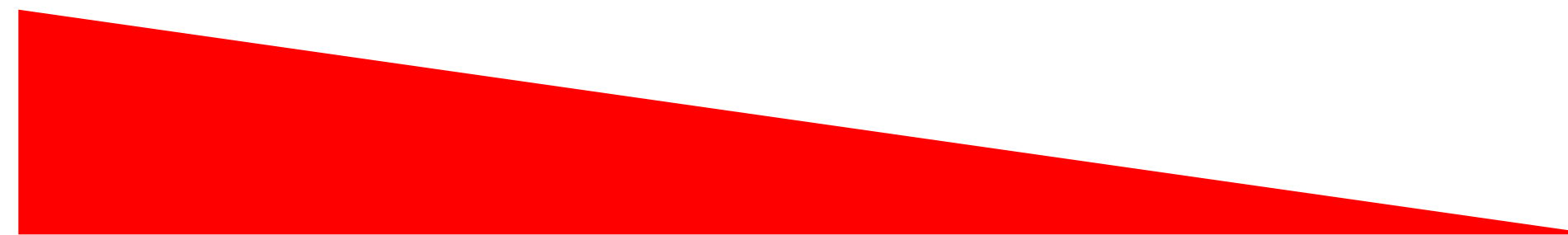
Многоступенчатый фильтр



Источник: [Wikimedia Commons](#)

Многоступенчатый фильтр

Размер пропускаемого загрязнения
Пропускная способность



Многоступенчатый фильтр

Сложность, стоимость



Вывод

Вывод

- Статанализ — «фильтр грубой очистки» в начале каскада фильтров

Вывод

- Статанализ — «фильтр грубой очистки» в начале каскада фильтров
- В отдельности от других — не работает

Вывод

- Статанализ — «фильтр грубой очистки» в начале каскада фильтров
- В отдельности от других — не работает
- Другие без него работают хуже

Случай из практики: долгий отклик

resource.json

```
{  
  "key": "value with "unescaped quotes" "  
}
```


Случай из практики: долгий отклик

resource.json

```
{  
  "key": "value with "unescaped quotes" "  
}
```

- Все UI тесты падают.

Случай из практики: долгий отклик

resource.json

```
{  
  "key": "value with "unescaped quotes" "  
}
```

- Все UI тесты падают.
- Но это происходит **спустя дни**.

Случай из практики: лечение

- Добавляем JSONLint в начало пути

```
find . -name \\.json -print0 | xargs -0 -n1 -t jsonlint -q
```

Случай из практики: лечение

- Добавляем JSONLint в начало пути

```
find . -name \\.json -print0 | xargs -0 -n1 -t jsonlint -q
```

- Отклик на проблему идёт **сразу**

Случай из практики: лечение

- Добавляем JSONLint в начало пути

```
find . -name \\.json -print0 | xargs -0 -n1 -t jsonlint -q
```

- Отклик на проблему идёт **сразу**

• PROFIT

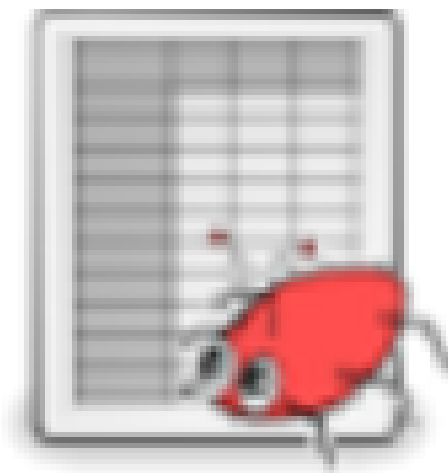
Внедрение в legasy-проект

Внедрение в legacy-проект

Знакомая картина?



Checkstyle: 2,496 warnings from one analysis.



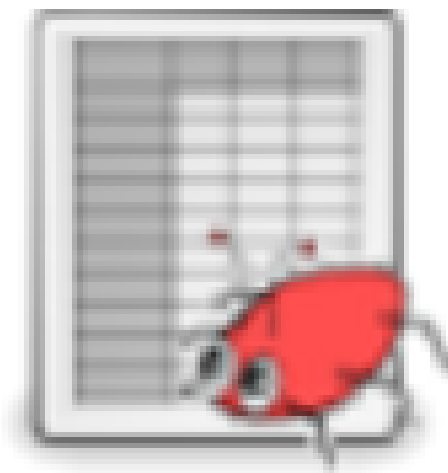
FindBugs: 130 warnings from one analysis.

Внедрение в legacy-проект

Знакомая картина?



Checkstyle: 2,496 warnings from one analysis.



FindBugs: 130 warnings from one analysis.

Оставить нельзя пофиксить!

Пофиксить автоматически?

Google + Stackoverflow:

- 'sed remove trailing spaces'

```
find . -name '*.py' -print0 | xargs -0 -n1 -t \  
    sed -i -r 's/\s+$//'
```

- 'bash add a newline to the end of a file'

```
find . -name '*.java' -print0 | xargs -0 -L1 bash \  
    -c 'test "$(tail -c 1 "$0")" && printf "\r\n" >> $0'
```

- etc etc

Автофикс

- Javascript: `eslint --fix`

Spotless: идемпотентный автоформаттер

Spotless can format

<java | kotlin | scala | sql | groovy | javascript | flow | typeScript | css | scss | less | jsx | vue
| graphql | json | yaml | markdown | license headers | anything>

using

<gradle | maven | anything>

Quality Gates

Пороговое значение находок

Пороговое значение находок

- «Если меньше 100 находок, то код ОК»

Пороговое значение находок

- «Если меньше 100 находок, то код ОК»
- ДАНО: в коде 90 находок и код ОК.

Пороговое значение находок

- «Если меньше 100 находок, то код ОК»
- ДАНО: в коде 90 находок и код ОК.
- Добавляем Null Pointer Dereference.

Пороговое значение находок

- «Если меньше 100 находок, то код ОК»
- ДАНО: в коде 90 находок и код ОК.
- Добавляем Null Pointer Dereference.
- У нас 91 находка, код всё ещё ОК?

Пороговое значение находок

- «Если меньше 100 находок, то код ОК»
- ДАНО: в коде 90 находок и код ОК.
- Добавляем Null Pointer Dereference.
- У нас 91 находка, код всё ещё ОК?

Вывод: не используйте данный метод!

Suppression Profile

- Старые находки — в игнор
- Новые находки — не пропускаем

Suppression Profile

Наивный подход:

```
<file name="AppProperties.java">  
  <error line="31" column="5" message="Missing a Javadoc comment."/>  
  <error line="36" column="5" message="Missing a Javadoc comment."/>  
</file>
```

Suppression Profile

Наивный подход:

```
<file name="AppProperties.java">  
  <error line="31" column="5" message="Missing a Javadoc comment."/>  
  <error line="36" column="5" message="Missing a Javadoc comment."/>  
</file>
```

Добавляем текст в начало файла...

Suppression Profile

Наивный подход:

```
<file name="AppProperties.java">  
  <error line="31" column="5" message="Missing a Javadoc comment."/>  
  <error line="36" column="5" message="Missing a Javadoc comment."/>  
</file>
```

Добавляем текст в начало файла...

...номера строк "уползли" и все находки снова появились.

Suppression Profile

Подход PVS-Studio --- хеши строк:

```
{  
  "FileName": "CelestaParser.java",  
  "ErrorCode": "V6021",  
  "CodePrev": -1464702071,  
  "CodeCurrent": -1679070819,  
  "CodeNext": 35764079  
}
```

Suppression Profile

Вывод: метод хорош, но труднодоступен

Проверка правописания

GNU Aspell



Проверка документации:

```
for f in $(find . -name '*.adoc'); do \  
  cat $f | aspell --master=ru \  
  --personal=./dict list; done \  
| sort | uniq
```

Проверка литералов и комментариев:

```
for f in $(find . -name '*.java'); do \  
  cat $f | aspell --mode=ccpp \  
  --master=ru --personal=./dict list; done \  
| sort | uniq
```

Проверка правописания

Проверка правописания

- Храните пользовательский словарь в проекте

Проверка правописания

- Храните пользовательский словарь в проекте
- Quality Gate: **не должно быть незнакомых спелчекеру слов.**

Упавшая проверка

Stage Logs (Spellcheck)

Shell Script -- true -- (self time 263ms)

Print Message -- The following words are probaly misspelled: -- (self time 5ms)

Print Message -- вфыва длфыовфыа фыв фыжв ыавдыфоа ыффыва -- (self time 4ms)

вфыва

длфыовфыа

фыв

фыжв

ыавдыфоа

ыффыва

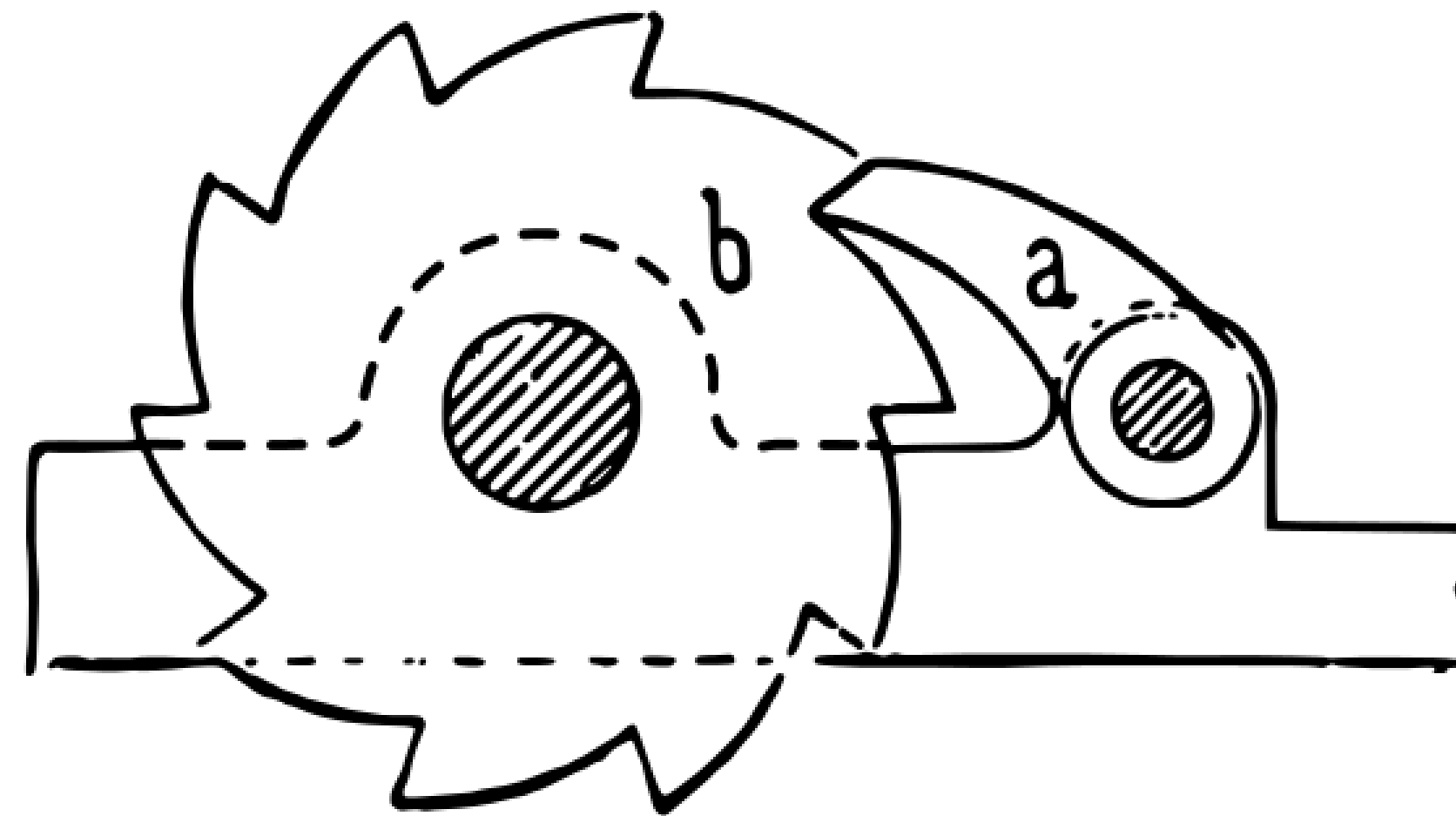
Error signal (self time 5ms)

Проверка правописания

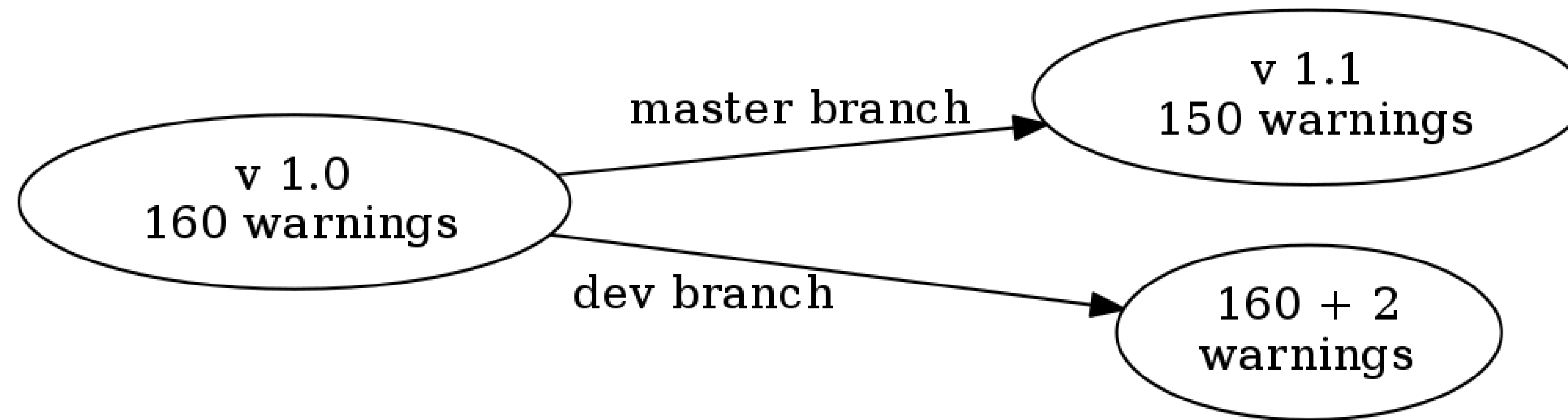
Вывод: spellchecker может быть частью пайплайна

Храповик

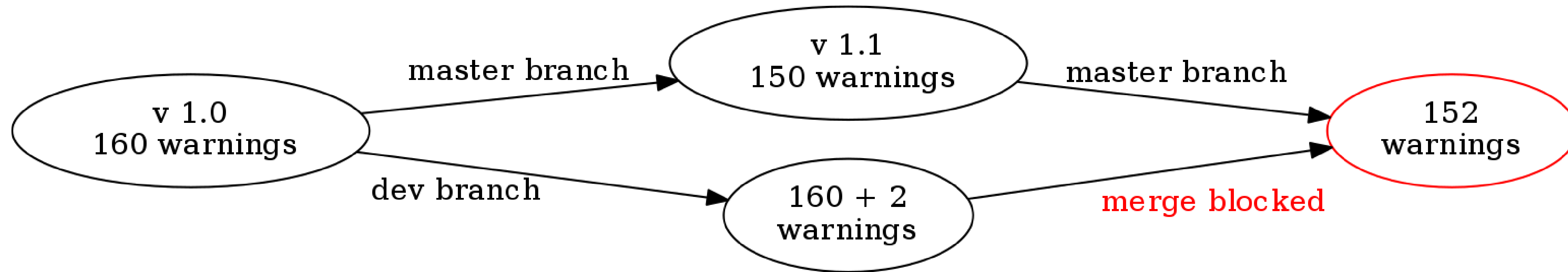
Штука, позволяющая движение
только в нужную сторону



Принцип работы



Принцип работы



Full project Number of flake8 warnings 1597 is greater than previous 1596.



Stage View

Average stage times:

| | Clone | Static analysis |
|----|-------|-----------------|
| #1 | 7s | 23s |

Mar 30 03:24 No Changes

03:24

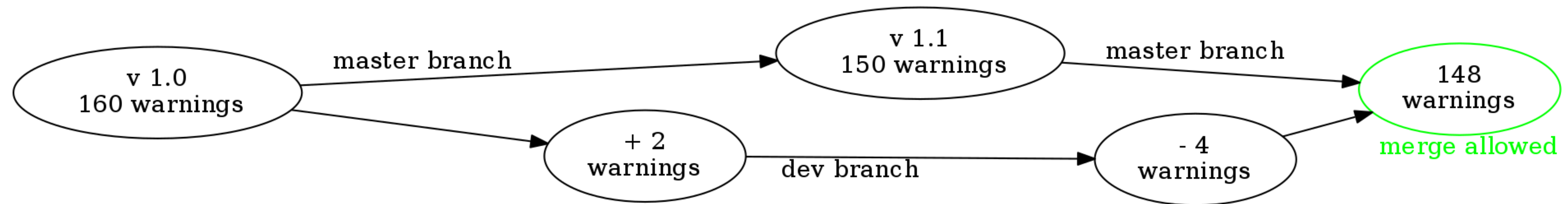
23s

816ms

failed

ror(s)

Принцип работы





Много модулей/инструментов

Вид метаданных:

```
# warnings.yml
celestasql:
  checkstyle: 434
  spotbugs: 45
celestacore:
  checkstyle: 206
  spotbugs: 13
celestamavenplugin:
  checkstyle: 19
  spotbugs: 0
celestasunit:
  checkstyle: 0
  spotbugs: 0
```

Упавшая проверка

```
= celesta-sql.checkstyle: 399->399
= celesta-sql.findbugs: 41->41
= celesta-core.checkstyle: 185->185
= celesta-core.findbugs: 13->13
+ celesta-maven-plugin.checkstyle: 19->20
= celesta-maven-plugin.findbugs: 0->0
= celesta-system-services.checkstyle: 18->18
= celesta-system-services.findbugs: 0->0
= dbschemasync.checkstyle: 3->3
= dbschemasync.findbugs: 0->0
= celesta-unit.checkstyle: 0->0
= celesta-unit.findbugs: 0->0
```

Error signal (self time 3ms)

Как это реализовано у нас

- Jenkins scripted pipeline
- Jenkins shared libraries in Groovy
- JFrog Artifactory для хранения метаданных о сборках

Подсчёт основан на XML-выводе анализаторов

```
private Map countModule(prefix) {  
    def count = [:]  
    def f = new File("${prefix}/target/checkstyle-result.xml")  
    if (f.exists()) {  
        def checkstyle = new XmlSlurper().parseText(f.text)  
        count.put("checkstyle", checkstyle.file.error.size())  
    }  
    . . .  
    count  
}
```

Скачиваем данные о последней сборке

```
def server = Artifactory.server 'ART'
def downloadSpec = """
    {"files": [
      {
        "pattern": "warn/${project}/*/warnings.yml",
        "build": "${project} :: dev/LATEST",
        "target": "previous.yml",
        "flat": "true"
      }
    ]}
"""
server.download spec: downloadSpec
oldWarnings = readYaml file: 'previous.yml'
```


Шаг храповика

```
stage ( 'Ratcheting' ) {  
  def warningsMap = countWarnings()  
  writeYaml file: 'target/warnings.yml', data: warningsMap  
  compareWarningMaps oldWarnings, warningsMap  
}
```

Jenkins Warnings NG Plugin

Собирает и читает отчёты всех известных анализаторов

```
def checkstyle
    = scanForIssues tool: checkStyle(pattern: '**/cs.xml')

def spotbugs
    = scanForIssues tool: spotBugs(pattern: '**/spotbugs.xml')

def idea
    = scanForIssues tool: ideaInspection(pattern: 'target/idea_inspections/*')

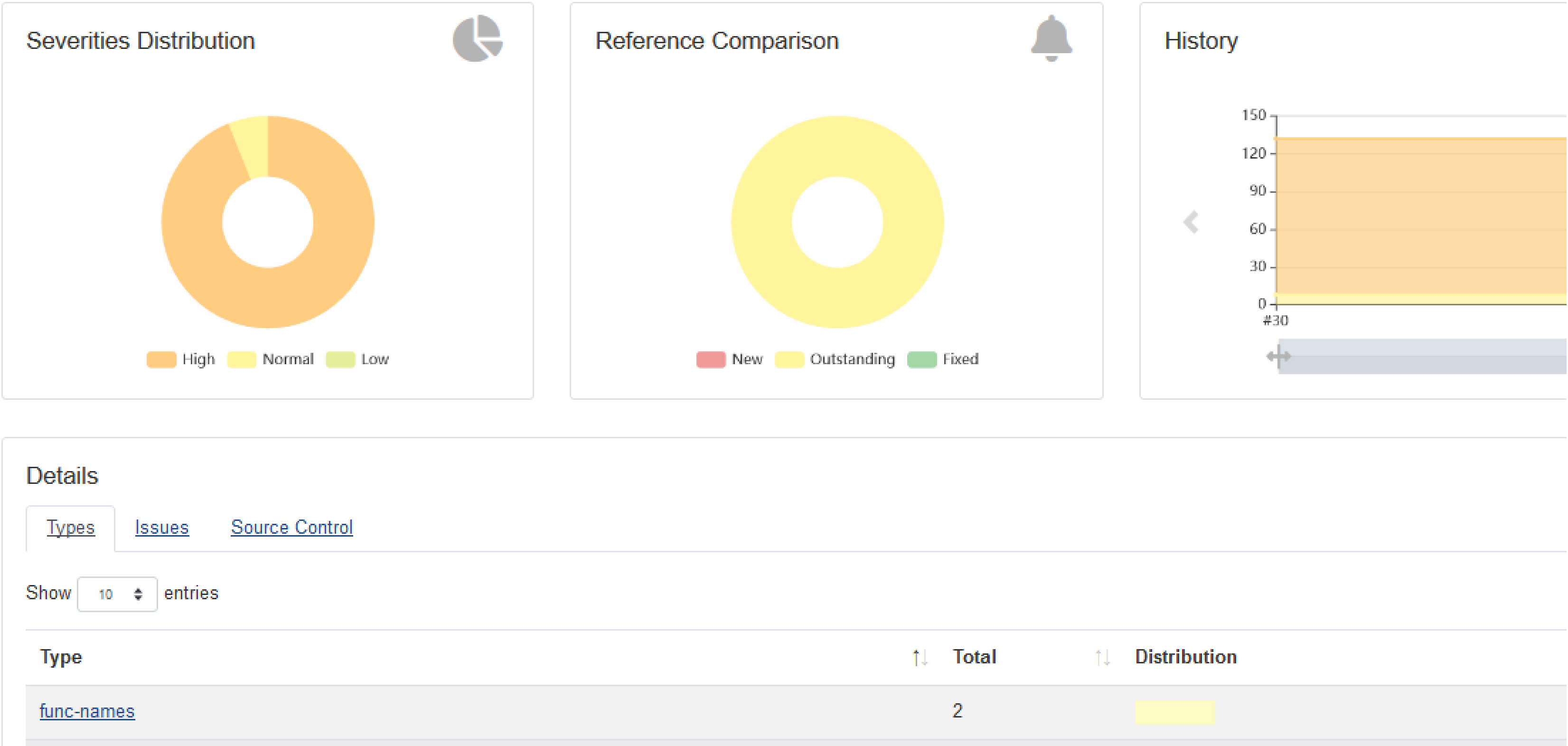
def eslint
    = scanForIssues tool: esLint(pattern: '**/eslint.xml')

publishIssues issues: [checkstyle, spotbugs, idea, eslint]
. . .
```

Jenkins Warnings NG Plugin

Красиво отображает

ESLint Warnings

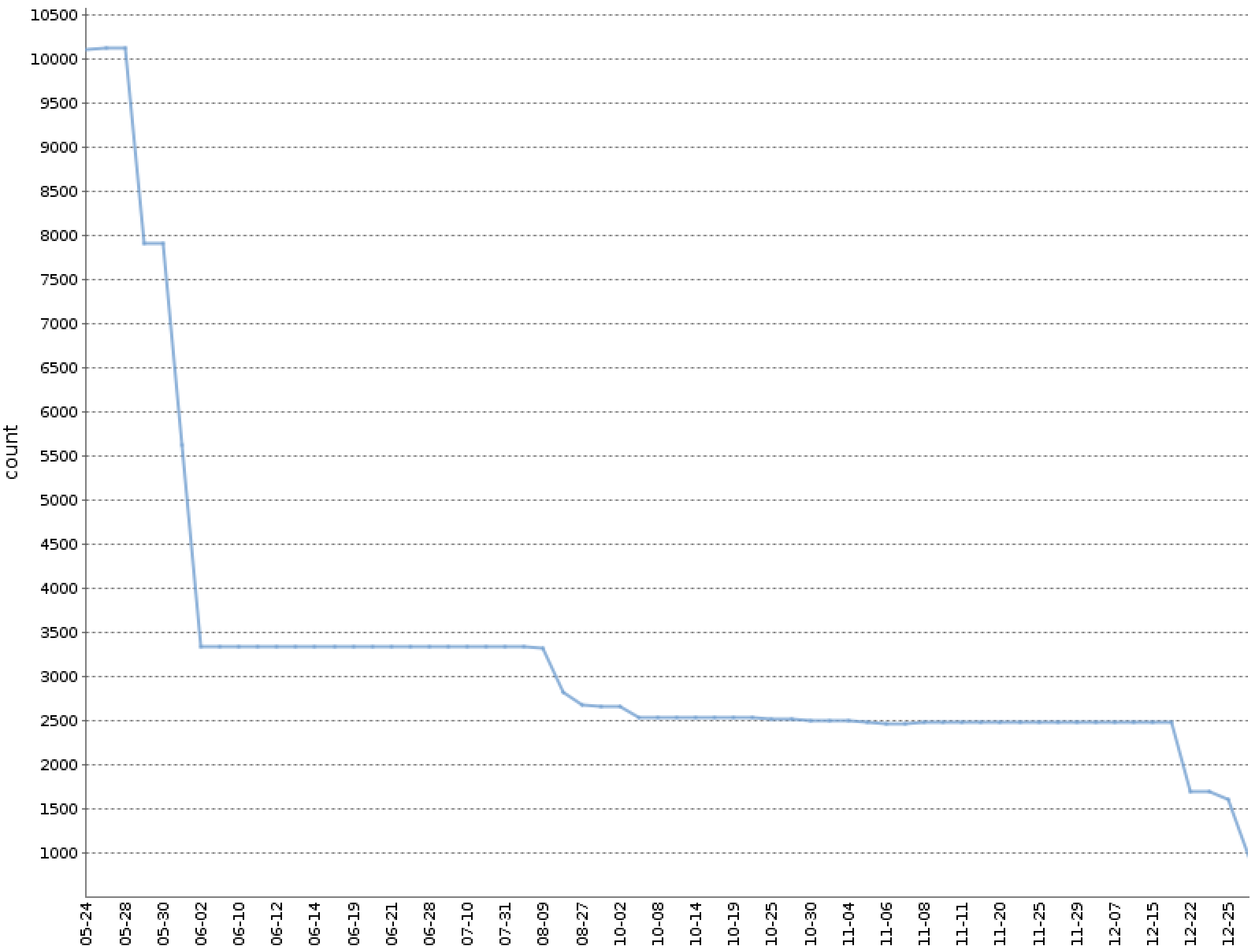


Jenkins Warnings NG Plugin

Можно программировать Quality Gates, в т. ч. в виде разницы с reference build:

```
recordIssues tool: java(pattern: '*.log'),  
  qualityGates: [[threshold: 1, type: 'TOTAL',  
                                unstable: true]]
```

Храповик: работа за полгода



Checkstyle warnings, [Celesta project](#)

Случай из практики

Кто здесь видит проблему?

```
#.travis.yml
```

```
install:
```

- pip install yamllint
- pip install ansible-lint

```
script:
```

```
. . .
```

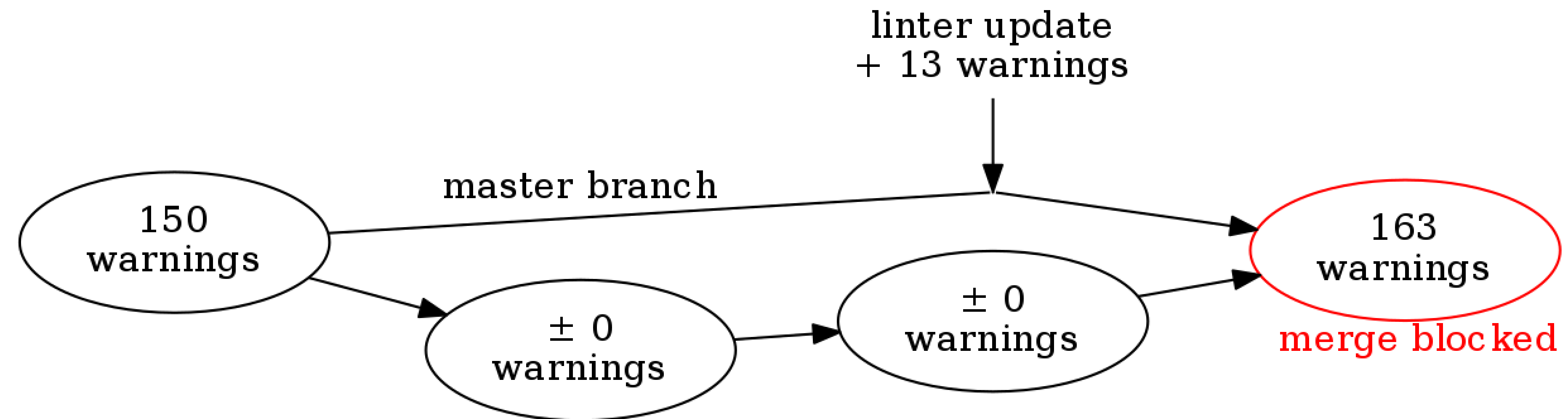
```
# Check YAML validity
```

- yamllint -c yamllint.yml .

```
# Ansible code static analysis
```

- ansible-lint . . .
- ansible-lint . . .
- ansible-lint . . .

Невоспроизводимая сборка



Фиксируем версии всего!

#.travis.yml

install:

- pip install yamllint==1.13.0
- pip install ansible-lint==3.5.1

Выводы

Выводы

- Статический анализ многолик

Выводы

- Статический анализ многолик
- Статический анализ бесполезен при нерегулярном применении

Выводы

- Статический анализ многолик
- Статический анализ бесполезен при нерегулярном применении
- Внедряйте анализ в качестве первого звена конвейера интеграции

Выводы

- Статический анализ многолик
- Статический анализ бесполезен при нерегулярном применении
- Внедряйте анализ в качестве первого звена конвейера интеграции
- Устанавливайте quality gates. Используйте метод храповика



Выводы

- Статический анализ многолик
- Статический анализ бесполезен при нерегулярном применении
- Внедряйте анализ в качестве первого звена конвейера интеграции
- Устанавливайте quality gates. Используйте метод храповика
- Не забывайте про воспроизводимость сборок

Ссылки

- **Humble, Jez; Farley, David (2011).** Continuous Delivery: reliable software releases through build, test, and deployment automation.
- **Иван Пономарев** [Внедряйте статический анализ в процесс, а не ищите с его помощью баги](#)
- **Алексей Кудрявцев** [Анализ программ: как понять, что ты хороший программист](#)

На этом всё!

-  @inponomarev
-  ponomarev@corchestra.ru
- **Спасибо!**