

# Непрерывный статический анализ кода

## Содержание

1. Льём в прод	3
2. Кодим	3
3. Delivery Pipeline	4
4. Покупайте лучшие анализаторы!	5
5. Что такое статанализ?	5
6. Чего в принципе не может СА?	6
7. Чего в принципе не может СА?	6
8. Теорема Райса	6
9. Статанализ не найдёт то, что решается на уровне языка	6
10. Чистый горный источник	7
11. А что же статанализ?	7
12. Разнообразие средств статанализа	7
13. Разнообразие средств статанализа	8
14. Больше анализаторов!	8
15. Обязательно найдете	8
16. Кто программирует на Bash?	8
17. IntelliJ IDEA в CI	9
18. Что из этого использовать?	9
19. Как это всё использовать?	9
20. Роль и место СА в конвейере поставки	10
21. Типовой конвейер сборки	10
22. Типовой конвейер сборки	10
23. Типовой конвейер сборки	10
24. Многоступенчатый фильтр	11
25. Многоступенчатый фильтр	11
26. Многоступенчатый фильтр	11
27. Вывод	12
28. Случай из практики: долгий отклик	12
29. Случай из практики: лечение	12
30. Внедрение в legacy-проект	12
31. Внедрение в legacy-проект	13
32. С легаси нелегко	13
33. Пофиксить автоматически?	13
34. Автофикс	14

35. Spotless: идемпотентный автоформаттер . . . . .	14
36. Теперь полегче . . . . .	14
37. Quality Gates . . . . .	15
38. Пороговое значение находок . . . . .	15
39. Suppression Profile . . . . .	15
40. Suppression Profile . . . . .	15
41. Suppression Profile . . . . .	16
42. Suppression Profile . . . . .	16
43. Проверка правописания . . . . .	16
44. Проверка правописания . . . . .	16
45. Проверка правописания . . . . .	17
46. Упавшая проверка . . . . .	17
47. Проверка правописания . . . . .	17
48. Храповик . . . . .	17
49. Принцип работы . . . . .	18
50. Принцип работы . . . . .	18
51. Упавшая проверка . . . . .	19
52. Принцип работы . . . . .	19
53. Принцип работы . . . . .	20
54. Много модулей/инструментов . . . . .	21
55. Упавшая проверка . . . . .	21
56. Как это реализовано у нас . . . . .	21
57. Парсинг XML-вывода анализаторов . . . . .	22
58. Скачиваем данные о последней сборке . . . . .	22
59. Шаг храповика . . . . .	22
60. Jenkins Warnings NG Plugin . . . . .	23
61. Jenkins Warnings NG Plugin . . . . .	23
62. Jenkins Warnings NG Plugin . . . . .	23
63. Храповик: ожидание . . . . .	24
64. Храповик: реальность . . . . .	24
65. Случай из практики . . . . .	24
66. Невоспроизводимая сборка . . . . .	25
67. При замене фильтров бывает грязно! . . . . .	26
68. Фиксируем версии всего! . . . . .	26
69. Выводы . . . . .	26
70. Статанализ разнообразен . . . . .	26
71. Статанализ бесполезен при нерегулярном применении . . . . .	27
72. Фильтр грубой очистки ставится в начале каскада . . . . .	27
73. Используйте храповик . . . . .	28
74. Не забывайте про повторяемость сборок . . . . .	28
75. Ссылки . . . . .	29

[ponomarev@corchestra.ru](mailto:ponomarev@corchestra.ru)

 @inponomarev

## 1. Льём в прод



## 2. Кодим



### 3. Delivery Pipeline



## 4. Покупайте лучшие анализаторы!



## 5. 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).

## 6. Чего в принципе не может СА?

## 7. 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
```

## 8. 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)
```

## 9. Статанализ не найдёт то, что решается на уровне языка

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

## 10. Чистый горный источник



Близкое динамическое болото



Труднодоступное статическое озеро

## 11. А что же статанализ?



## 12. Разнообразие средств статанализа



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

# 13. Разнообразие средств статанализа



- Компиляция/парсинг (`ansible --syntax-check`, `terraform validate`)
- Предупреждения компиляторов
- Проверка правописания
- Конфигурационные тесты

# 14. Больше анализаторов!

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

# 15. Обязательно найдете



# 16. Кто программирует на Bash?

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

```
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 $
```

## 17. IntelliJ IDEA в CI

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

IntelliJ IDEA Inspections Warnings



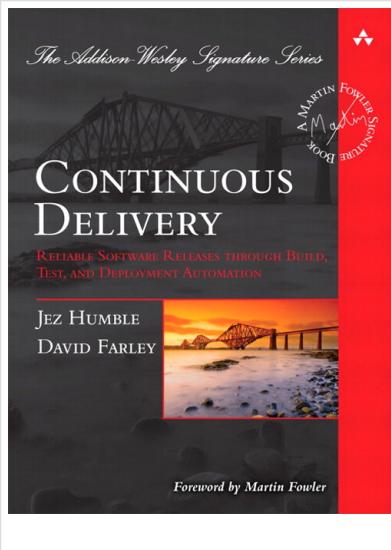
## 18. Что из этого использовать?

Всё!

## 19. Как это всё использовать?

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

## 20. Роль и место СА в конвейере поставки



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

## 21. Типовой конвейер сборки



## 22. Типовой конвейер сборки

«Фильтрующая способность»



## 23. Типовой конвейер сборки

Сложность, стоимость,  
время работы, вероятность сбоя





## 24. Многоступенчатый фильтр



## 25. Многоступенчатый фильтр

Размер пропускаемого загрязнения  
Пропускная способность



## 26. Многоступенчатый фильтр

Сложность, стоимость





## 27. Вывод

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

## 28. Случай из практики: долгий отклик

`resource.json`

```
{  
  "key": "value with \"unescape quotes\" "  
}
```

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

## 29. 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

## 30. Внедрение в legacy-проект

## 31. Внедрение в legacy-проект

Знакомая картина?



Checkstyle: [2,496 warnings](#) from one analysis.



FindBugs: [130 warnings](#) from one analysis.

Оставить нельзя пофиксить!

## 32. С легаси нелегко



## 33. 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

## 34. Автофикс

- Javascript: `eslint --fix`

## 35. 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>

## 36. Теперь легче



## 37. Quality Gates



## 38. 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!

## 39. Suppression Profile

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

## 40. 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.

## 41. Suppression Profile

Подход PVS-Studio --- хеши строк:

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

## 42. Suppression Profile

Вывод: метод хорош, но труднодоступен

## 43. Проверка правописания

## 44. 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
```

## 45. Проверка правописания

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

## 46. 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)

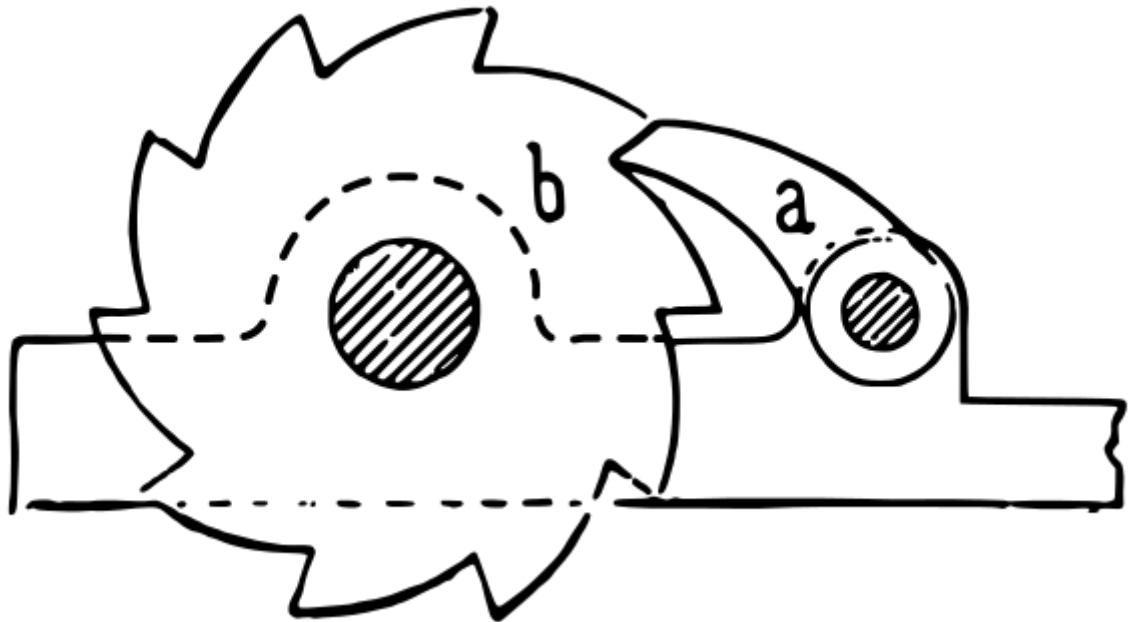
вфыва
длфыовфыа
фыв
фыжв
ыавдыфоа
ыффыва

Error signal (self time 5ms)
```

## 47. Проверка правописания

Вывод: spellchecker может быть частью пайплайна

## 48. Храповик



## 49. Принцип работы

Failed to generate image: Could not find the 'dot' executable in PATH; add it to the PATH or specify its location using the 'graphvizdot' document attribute

```
digraph G {
    rankdir="LR";
    graph [ dpi = 150 ];
    ri[label="v 1.0\n 160 warnings"];
    ri -> r1 [label="master branch"];
    r1 -> r2 [label="master branch", style="invis"];

    ri ->b1 [ xlabel = "dev branch"];
    r1[label="v 1.1\n 150 warnings"];
    b1[label="160 + 2\nwarnings"];

    b1->r2[style="invis"];
    r2[label="152\nwarnings", style="invis"];
}
```

## 50. Принцип работы

```

Failed to generate image: no implicit conversion of nil into String
digraph G {
    rankdir="LR";
    graph [ dpi = 150 ];
    ri[label="v 1.0\n 160 warnings"];
    ri -> r1 [label="master branch"];
    r1 -> r2 [label="master branch"];

    ri ->b1 [ xlabel = "dev branch"];
    r1[label="v 1.1\n 150 warnings"];
    b1[label="160 + 2\nwarnings"];

    b1->r2;
    r2[label="152\nwarnings", color="red", xlabel=<<font color="red">merge
blocked</font>>];
}

```

## 51. Упавшая проверка

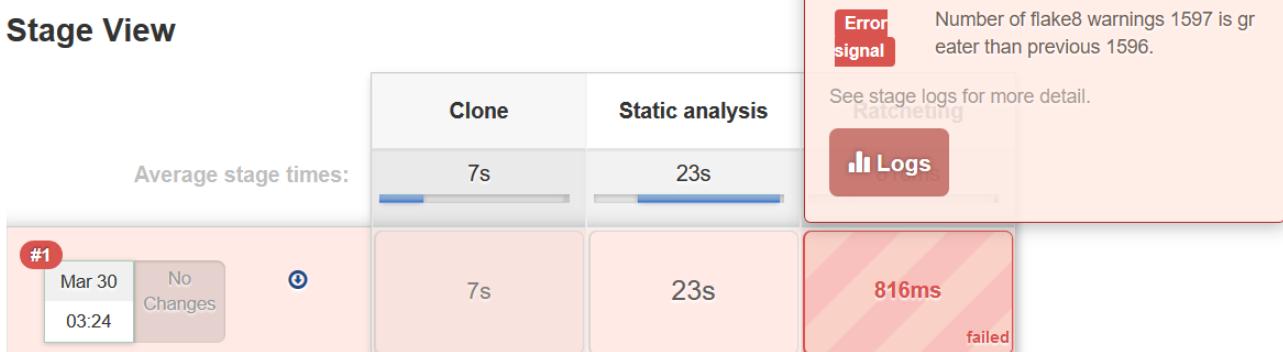
### Pull

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



rror(s)

### Stage View



## 52. Принцип работы

```
Failed to generate image: no implicit conversion of nil into String
digraph G {
    rankdir="LR";
    graph [ dpi = 150 ];
    ri[label="v 1.0\n 160 warnings"];
    ri -> r1 [label="master branch"];
    r1 -> r2 [label="master branch"];

    ri ->b1;
    r1[label="v 1.1\n 150 warnings"];
    b1[label="+ 2\nwarnings"];

    b1->b2[xlabel="dev branch"];
    b2->r2
    b2[label="- 4\nwarnings"]
    r2[label="148\nwarnings", color="green", xlabel=<<font color="green">merge
allowed</font>>]
}
```

## 53. Принцип работы

```
Failed to generate image: no implicit conversion of nil into String
digraph G {
    rankdir="LR";
    graph [ dpi = 150 ];
    ri[label="v 1.0\n 160 warnings"];
    ri -> r1 [label="master branch"];
    r1 -> r2 [label="master branch"];

    r1[label="v 1.1\n 150 warnings"];

    r2[label="v 1.2\n148 warnings"];
    r2->r3;

    b2->r3;

    b2[label="+ 1\nwarning"]
    r3[label="149\nwarnings", color="red", xlabel=<<font color="red">merge
blocked</font>>]
}
```

## 54. Много модулей/инструментов

Вид метаданных:

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

## 55. Упавшая проверка

```
= celestasql.checkstyle: 399->399
= celestasql.findbugs: 41->41
= celestacore.checkstyle: 185->185
= celestacore.findbugs: 13->13
+ celestamavenplugin.checkstyle: 19->20
= celestamavenplugin.findbugs: 0->0
= celestasystemservices.checkstyle: 18->18
= celestasystemservices.findbugs: 0->0
= dbschemasync.checkstyle: 3->3
= dbschemasync.findbugs: 0->0
= celestauit.checkstyle: 0->0
= celestauit.findbugs: 0->0
```

Error signal (self time 3ms)

## 56. Как это реализовано у нас

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

## 57. Парсинг XML-вывода анализаторов

```
<checkstyle>
  <file name="...">
    <error line="1" severity="..." message="..."/>
  </file>
  ...
</checkstyle>
```

```
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
}
```

## 58. 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'
```

## 59. Шаг храповика

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

## 60. 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/*.xml')

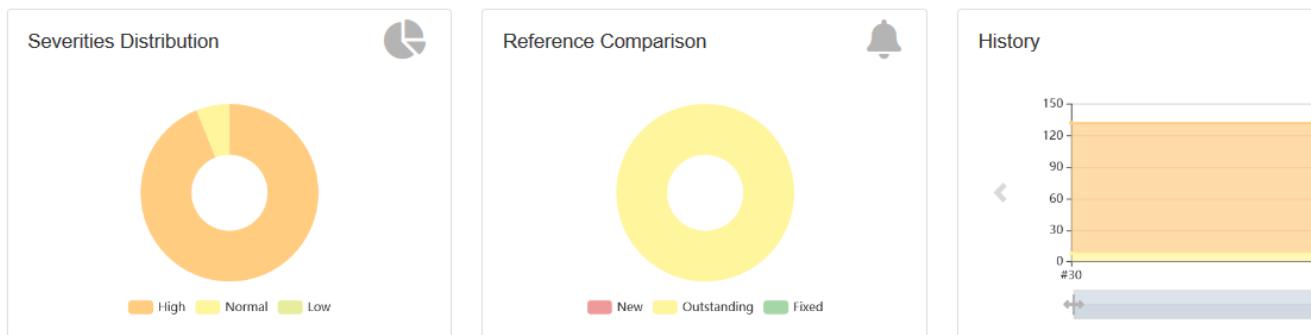
def eslint
= scanForIssues tool: esLint(pattern: '**/eslint.xml')

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

## 61. Jenkins Warnings NG Plugin

Красиво отображает

ESlint Warnings

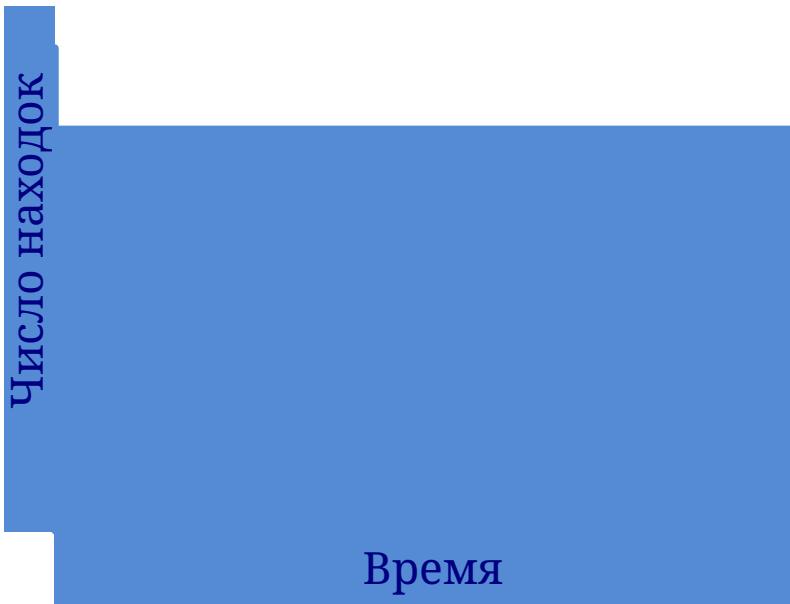


## 62. Jenkins Warnings NG Plugin

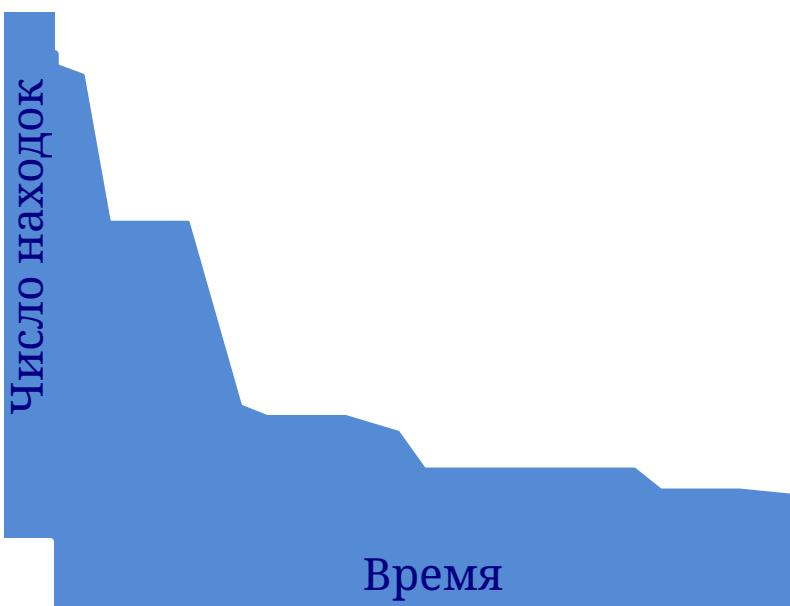
Можно программировать Quality Gates, в т. ч. в виде разницы с reference build:

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

## 63. Храповик: ожидание



## 64. Храповик: реальность



## 65. 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 . .

```

## 66. Невоспроизводимая сборка

```

Failed to generate image: no implicit conversion of nil into String
digraph G {
    rankdir="LR";
    graph [ dpi = 150 ];

    u -> r0;
    u[shape=plaintext; label="linter update\n+ 13 warnings"]
    r0[shape=point, width = 0]
    r1 -> r0[ arrowhead = none, label="master branch" ];

    r0-> r2 [];
    b1 -> b4;

    r1->b1
    r1[label="150\nwarnings"]
    b1[label="\u00b1 0\nwarnings"]

    b4[label="\u00b1 0\nwarnings"]
    b4->r2
    r2[label="163\nwarnings", color="red", xlabel=<<font color="red">merge
blocked</font>>]
    {rank = same; u; r0; b4;}
}

```

## 67. При замене фильтров бывает грязно!

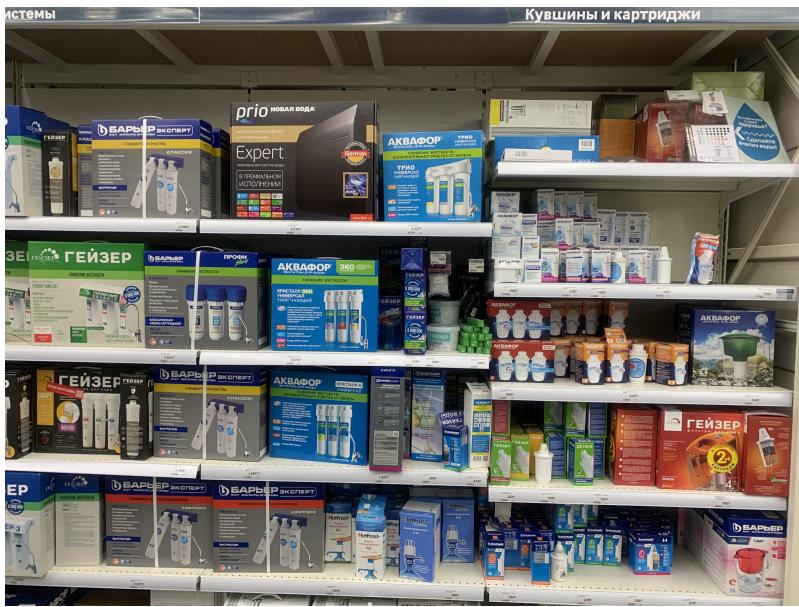


## 68. Фиксируем версии всего!

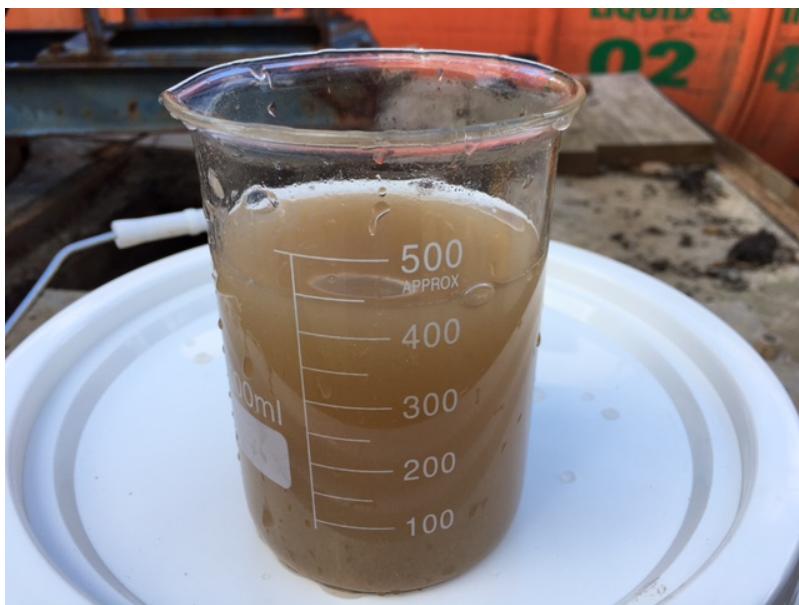
```
#.travis.yml
...
install:
  - pip install yamllint==1.13.0
  - pip install ansible-lint==3.5.1
```

## 69. Выводы

## 70. Статанализ разнообразен



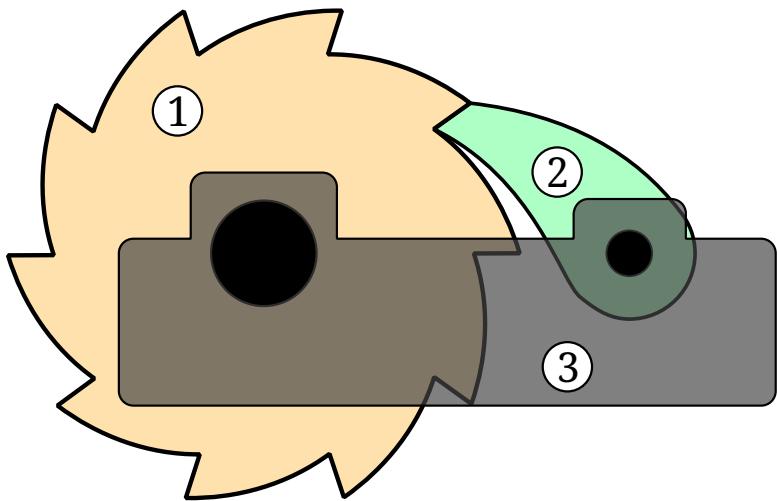
## 71. Статанализ бесполезен при нерегулярном применении



## 72. Фильтр грубой очистки ставится в начале каскада



## 73. Используйте храповик



## 74. Не забывайте про повторяемость сборок



## 75. 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
- **Иван Пономарев** Запускаем инспекции IntelliJ IDEA на Jenkins
- **Алексей Курячев** Анализ программ: как понять, что ты хороший программист

## 76. На этом всё!

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

