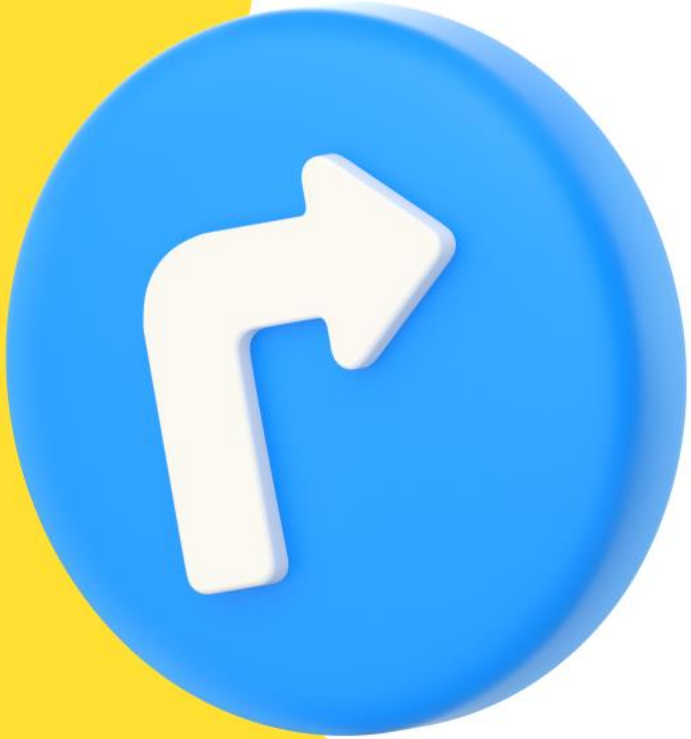
Яндекс Еда

Как статический анализ помогает улучшить код автотестов на Kotlin

Недосейкин Николай
Инженер мобильной платформы Яндекс Еды

Содержание



Наш стек автоматизации

Проблематика

Какие проблемы хотелось бы решить

01 О простых или неочевидных ошибках узнать ещё локально (Fail Fast)

Не беспокоиться о некоторых классах проблем

Какие проблемы хотелось бы решить

01 О простых или неочевидных ошибках узнать ещё локально (Fail Fast)

Не беспокоиться о некоторых классах проблем

02 Сэкономить ресурсы CI на сборку кода, который заведомо придется исправлять (Quality Gates)

Какие проблемы хотелось бы решить

01 О простых или неочевидных ошибках узнать ещё локально (Fail Fast)

Не беспокоиться о некоторых классах проблем

02 Сэкономить ресурсы CI на сборку кода, который заведомо придется исправлять (Quality Gates)

03 Реже переключать контекст ревьюеров и экономить их время

Статический анализ

01 Detekt используем почти в любом Kotlin коде, в том числе в UI-тестах

Статический анализ

01

Detekt используем почти в любом Kotlin коде, и том числе в UI-тестах

- длинные методы (тестовые)
- «магические числа»
- дубликаты строк тестовых данных

Статический анализ

01 **Detekt используем почти в любом Kotlin коде, и том числе в UI-тестах**

- длинные методы (тестовые)
- «магические числа»
- дубликаты строк тестовых данных

02 **При желании можно сделать автоисправление**

StringLiteralDuplication

hasTextColor — пример собственной проверки, которая работает с произвольными цветами, полученными от бэкенда и для которых могут не существовать Android-ресурсы

```
// Не рекомендуем
SomeScreen {
  element {
    title.hasTextColor("#0000FF")
    subTitle.hasTextColor("#0000FF")
  }
}
```

StringLiteralDuplication

hasTextColor — пример собственной проверки, которая работает с произвольными цветами, полученными от бэкенда и для которых могут не существовать Android-ресурсы

```
// Не рекомендуем
SomeScreen {
  element {
    title.hasTextColor("#0000FF")
    subTitle.hasTextColor("#0000FF")
  }
}

// Рекомендуем
SomeScreen {
  element {
    title.hasTextColor(PURE_BLUE)
    subTitle.hasTextColor(PURE_BLUE)
  }
}
```

Примеры наших правил

2.1 TestClassNamingRule

01 Название тестового класса в Junit4 должно содержать `Test`, иначе возникнут сложности с его запуском

02 Используется конвенция `**/*Test*.java` и её Kotlin аналоги

```
// Не рекомендуем
class BottomSheet : BaseTestCase()
```

2.1 TestClassNamingRule

01 Название тестового класса в Junit4 должно содержать `Test`, иначе возникнут сложности с его запуском

02 Используется конвенция `**/*Test*.java` и её Kotlin аналоги

```
// Не рекомендуем
class BottomSheet : BaseTestCase()

// Рекомендуем
class BottomSheetTest : BaseTestCase()
```

2.2 TestMethodNamingRule

01 Длина квалифицированного имени тестового метода должна быть не более рекомендуемого значения, чтобы с квалифицированным именем было удобно работать в Marathon файле

2.2 TestMethodNamingRule

01 Длина квалифицированного имени тестового метода должна быть не более рекомендуемого значения, чтобы с квалифицированным именем было удобно работать в Marathon файле

02 Название тестовой функции не должно содержать слово `test`, так как оно не несёт полезной нагрузки

Должно состоять более чем из одного слова

```
// Не рекомендуем
@Test fun testFeature42() { }
```

2.2 TestMethodNamingRule

01 Длина квалифицированного имени тестового метода должна быть не более рекомендуемого значения, чтобы с квалифицированным именем было удобно работать в Marathon файле

02 Название тестовой функции не должно содержать слово `test`, так как оно не несёт полезной нагрузки

Должно состоять более чем из одного слова

```
// Не рекомендуем
@Test fun testFeature42() { }

// Рекомендуем
@Test fun checkFooterForAvailableState() { }
```

2.3 IsVisibleUsageRule

01 Нередко при проверке отображения возникает желание использовать Kaspresso `isVisible`, но более подходящей проверкой в общем случае будет `isDisplayed`

02 Также это окажет влияние на стабильность за счёт flakySafety

```
// Не рекомендуем
SomeScreen {
    element {
        isVisible()
        click()
    }
}
```

2.3 IsVisibleUsageRule

01 Нередко при проверке отображения возникает желание использовать Kaspresso `isVisible`, но более подходящей проверкой в общем случае будет `isDisplayed`

02 Также это окажет влияние на стабильность за счёт flakySafety

```
// Не рекомендуем
SomeScreen {
    element {
        isVisible()
        click()
    }
}

// Рекомендуем
SomeScreen {
    element {
        isDisplayed()
        click()
    }
}
```

2.3 IsVisibleUsageRule

withEffectiveVisibility

```
public static Matcher<View> withEffectiveVisibility(ViewMatchers.Visibility visibility)
```

Returns a matcher that matches `Views` that have “effective” visibility set to the given value.

Effective visibility takes into account not only the view’s visibility value, but also that of its ancestors. In case of View.VISIBLE, this means that the view and all of its ancestors have `visibility-VISIBLE` . In case of GONE and INVISIBLE, it’s the opposite — any GONE or INVISIBLE parent will make all of its children have their effective visibility.

Note: Contrary to what the name may imply, view visibility does not directly translate into whether the view is displayed on screen (use `isDisplayed` for that). For example, the view and all of its ancestors can have `visibility=VISIBLE` , but the view may need to be scrolled to in order to be actually visible to the user. Unless you’re specifically targeting the visibility value with your test, use `isDisplayed`

2.4 RestrictedKeywordRule

Доклад с упоминанием `if` в тестах (холивар)

2.4 RestrictedKeywordRule

01 На уровне теста не должно быть ветвлений, поэтому запрещаем `if` `when`

2.4 RestrictedKeywordRule

01 На уровне теста не должно быть ветвлений, поэтому запрещаем `if` `when`

02 Неправильная обработка ошибок на уровне теста/экрана/сценария приводила к ожиданию до таймаута (у нас 20 секунд), поэтому запрещаем `try...catch` на этих уровнях

```
// Не рекомендуем
step("Нажимаем на кнопку если показалась") {
  try {
    SomeScreen {
      element {
        isDisplayed()
        click()
      }
    }
  } catch (e: NoMatchingViewException) { }
}
```

2.4 RestrictedKeywordRule

01 На уровне теста не должно быть ветвлений, поэтому запрещаем `if` `when`

02 Неправильная обработка ошибок на уровне теста/экрана/сценария приводила к ожиданию до таймаута (у нас 20 секунд), поэтому запрещаем `try...catch` на этих уровнях

```
// Не рекомендуем
step("Нажимаем на кнопку если показалась") {
  try {
    SomeScreen {
      element {
        isDisplayed()
        click()
      }
    }
  } catch (e: NoMatchingViewException) { }
}

// Рекомендуем улучшать testability и процессы
// Универсальный пример сложно показать
```

А ещё у нас есть такие правила

2.5 TestClassPrivateMemberRule

А ещё у нас есть такие правила

2.5 TestClassPrivateMemberRule

2.6 LargeScreenObjectRule

А ещё у нас есть такие правила

2.5 TestClassPrivateMemberRule

2.6 LargeScreenObjectRule

2.7 BaseScreenObjectRule

А ещё у нас есть такие правила

2.5 TestClassPrivateMemberRule

2.6 LargeScreenObjectRule

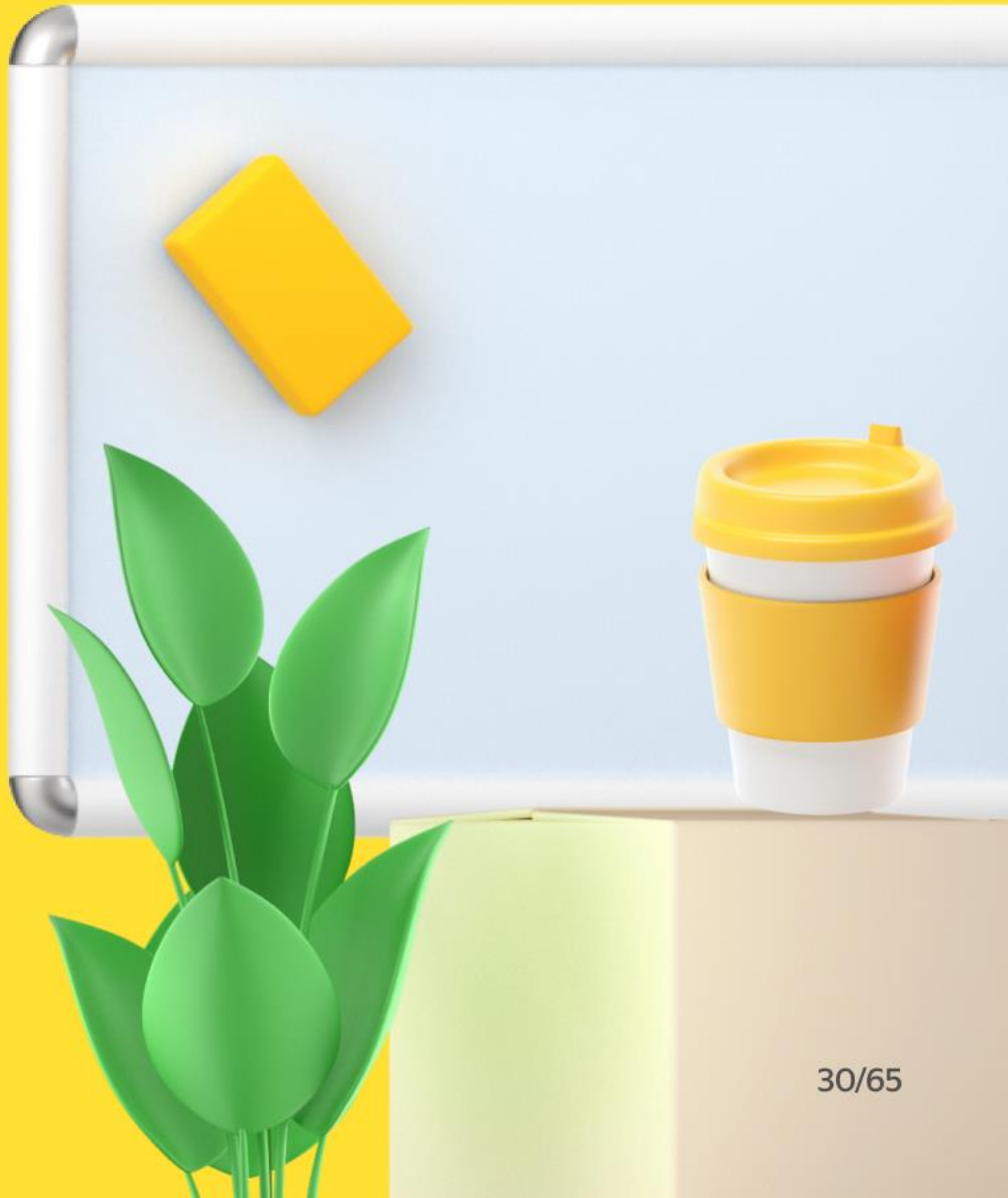
2.7 BaseScreenObjectRule

2.8 TmsLinkAnnotationRule

Идеи для новых правил

01 **Проверка лесенки аннотаций**

```
@Epic("e")
@Feature("f")
@Story("s")
```

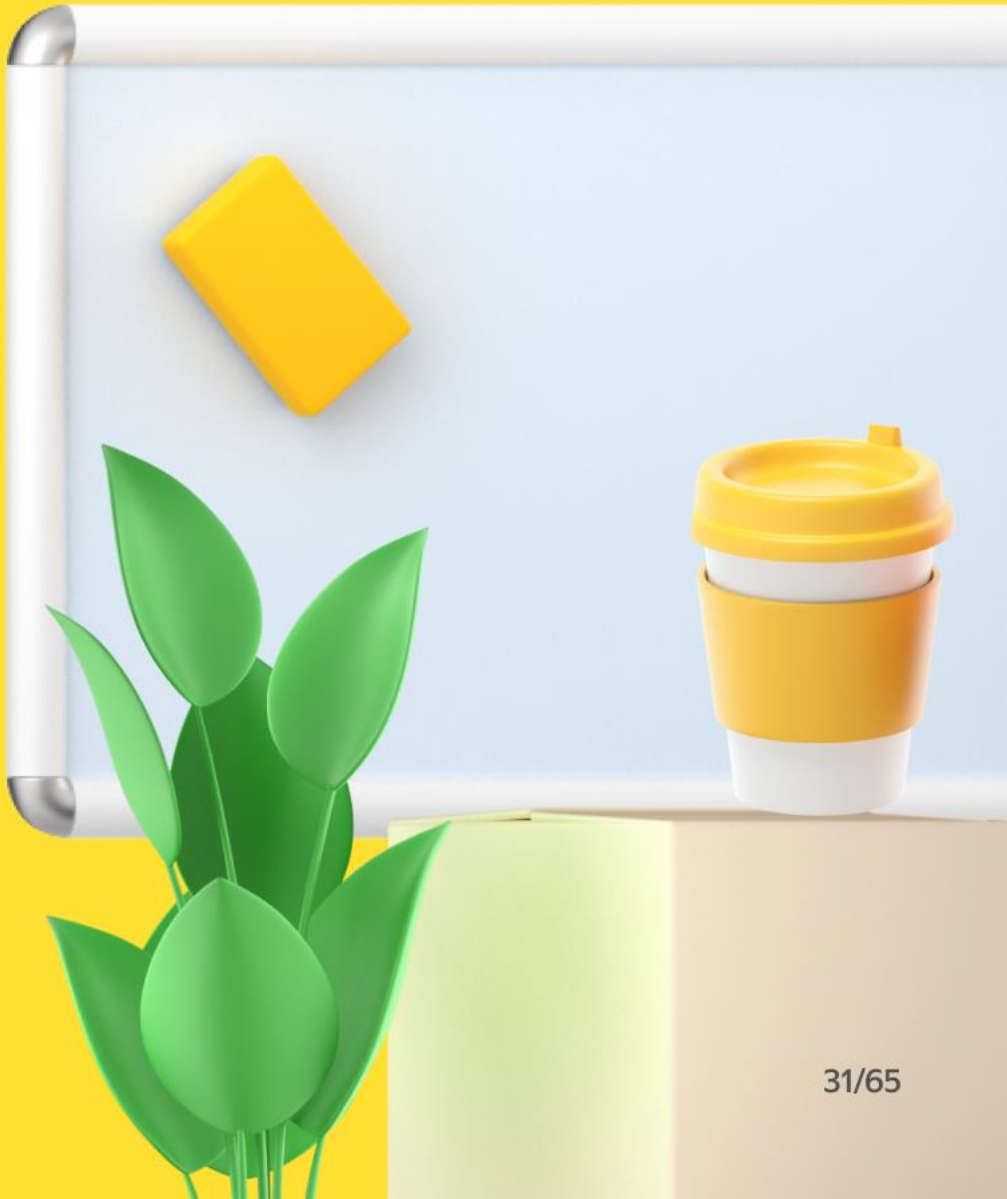


Идеи для новых правил

01 **Проверка лесенки аннотаций**

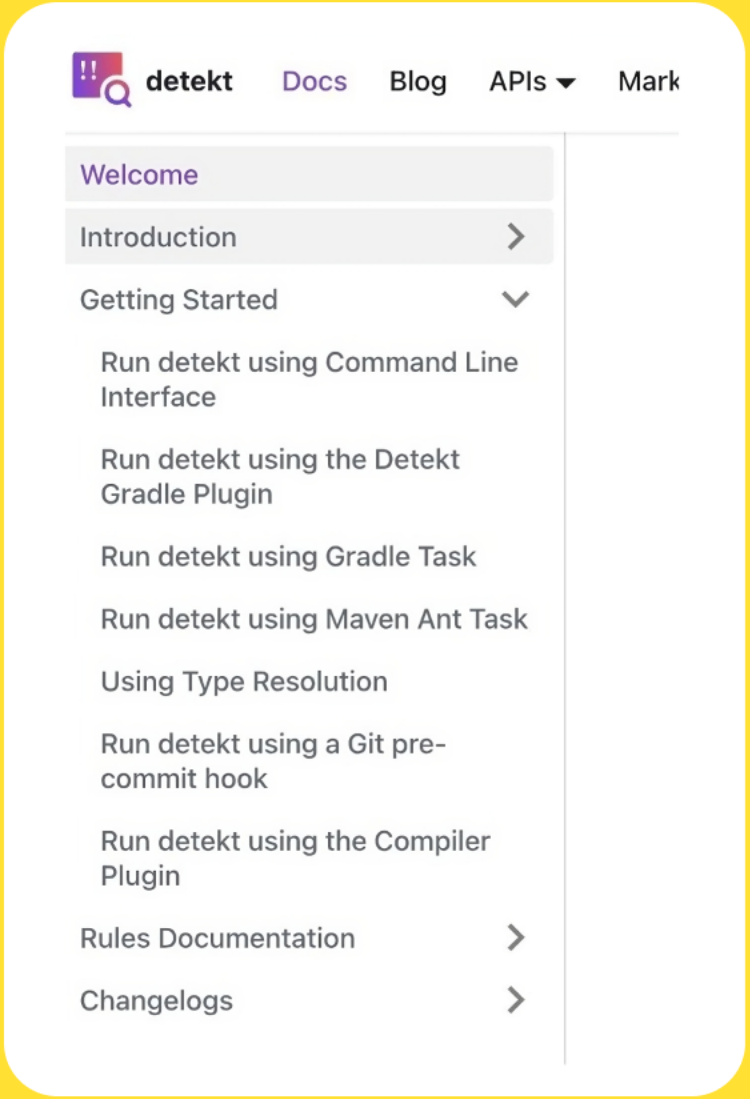
```
@Epic("e")
@Feature("f")
@Story("s")
```

02 **Запрет в тесте** `Thread.sleep` **и** `Screen.idle`



Алгоритм написания правила

Сначала коротко про подключение Detekt



Алгоритм написания правила

01 Донести до команды пользу правила

Алгоритм написания правила

01 Донести до команды пользу правила

02 Добавить правило в config.yml и свой RuleSetProvider

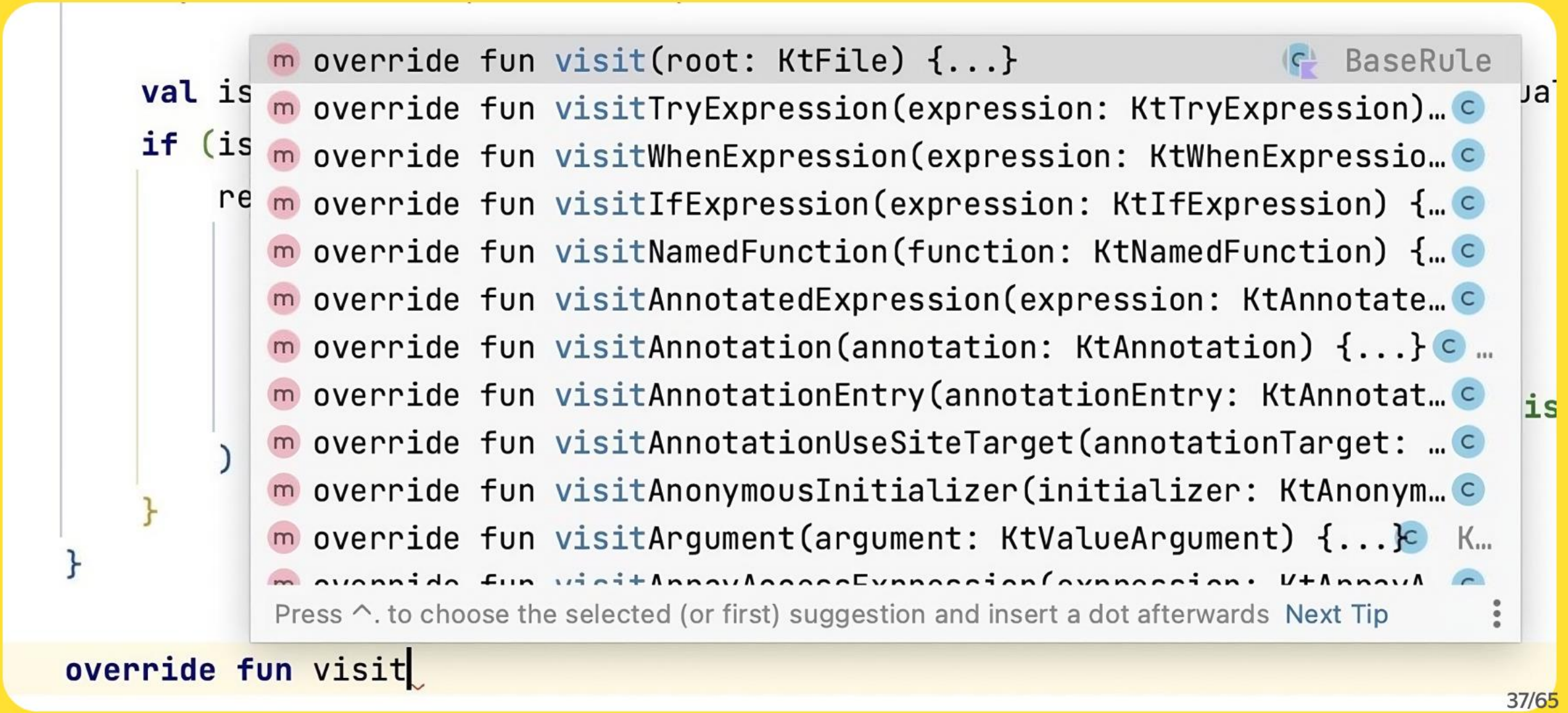
Алгоритм написания правила

01 Донести до команды пользу правила

02 Добавить правило в config.yml и свой RuleSetProvider

03 Написать первую наивную реализацию правила с использованием visit-функций

visit-функции для обхода кода



Особенности домена тестов

```
private const val TEST_ANNOTATION = "Test"

fun KtElement.inTestMethod(): Boolean =
    getNonStrictParentOfType<KtNamedFunction>()?.isTestMethod() ?: false

fun KtNamedFunction.isTestMethod(): Boolean =
    annotationEntries.any { it.shortName?.asString() == TEST_ANNOTATION }
```

Алгоритм написания правила

01 Донести до команды пользу правила

02 Добавить правило в config.yml и свой RuleSetProvider

03 Написать первую наивную реализацию правила с использованием visit-методов

04 Написать юнит-тесты к правилу

Юнит-тесты к правилу

```kotlin
@Test
fun containsUnexpectedCall() {
    // language="kotlin"
    val case =
        """
        class SomeTest : BaseTestCase() {
            @Test
            fun shouldDoSomething() {
                run {
                    step("should assert something") {
                        SomeScreen {
                            element.isVisible()
                        }
                    }
                }
            }
        }
        """.trimIndent()

    val findings = rule.lint(case)

    assertThat(findings).hasSize(1)
}
```

Алгоритм написания правила

01 Донести до команды пользу правила

02 Добавить правило в config.yml и свой RuleSetProvider

03 Написать первую наивную реализацию правила с использованием visit-методов

04 Написать юнит-тесты к правилу

05 Доработать наивную реализацию с учётом найденных нарушений и юнит-тестов

Алгоритм написания правила

01 Донести до команды пользу правила

02 Добавить правило в config.yml и свой RuleSetProvider

03 Написать первую наивную реализацию правила с использованием visit-методов

04 Написать юнит-тесты к правилу

05 Доработать наивную реализацию с учётом найденных нарушений и юнит-тестов

06 Добавить полученные нарушения в baseline.xml и создать задачи коллегам 😄

Либо исправить все нарушения самостоятельно в соответствии с «правилом бойскаутов» 😎

Писать правила несложно

```
class IsVisibleUsageRule(config: Config) : Rule(config) {

  override val issue = Issue(
    id = javaClass.simpleName,
    severity = Severity.Defect,
    description = "In general, 'Espresso isVisible' should not be used → use 'isDisplayed'",
    debt = Debt.FIVE_MINS,
  )

  override fun visitCallExpression(expression: KtCallExpression) {
    super.visitCallExpression(expression)

    val isVisibleAssertion = expression.getCalleeExpressionIfAny()?.text?.equals("isVisible") ?: false
    if (isVisibleAssertion) {
      report(
        CodeSmell(
          issue,
          Entity.from(expression),
          "In general, 'Espresso isVisible' should not be used → use 'isDisplayed'"
        )
      )
    }
  }
}
```

Хотя иногда нужно немало учесть

```
class TestClassPrivateMemberRule(config: Config) : Rule(config) {

    override val issue = Issue(
        id = javaClass.simpleName,
        severity = Severity.Defect,
        description = "Members of test class must use private modifier",
        debt = Debt(mins = 1)
    )

    private val baseTestClass: String by config(defaultValue = "TestCase")

    override fun visitObjectDeclaration(declaration: KtObjectDeclaration) {
        super.visitObjectDeclaration(declaration)

→       if (!declaration.inTestClass(baseTestClass)) return

→       if (declaration.isCompanion() &&
            !declaration.isPrivate()
        ) {
            reportIssue(declaration, "Add private modifier to companion object")
        }
    }

    override fun visitProperty(property: KtProperty) {
        super.visitProperty(property)

→       if (!property.inTestClass(baseTestClass)) return
→       if (property.annotationEntries.any { it.shortName?.asString() == "Rule" }) return

        val isNonLocalVariable = !property.isConstant() && !property.isLocal
→       if (isNonLocalVariable &&
            !property.isCompanionObjectProperty() &&
            !property.isPrivate()
        ) {
            reportIssue(property, "Add private modifier to non-local variable")
        }

→       if (!property.isConstant() &&
            property.isCompanionObjectProperty() &&
            property.isPrivate()
        ) {
            reportIssue(property, "Make property non-private and move it to private companion object")
        }

→       if (property.isConstant() &&
            property.isPrivate()
        ) {
            reportIssue(property, "Make constant non-private and move it to private companion object")
        }
    }

    override fun visitNamedFunction(function: KtNamedFunction) {
        super.visitNamedFunction(function)

→       if (!function.inTestClass(baseTestClass)) return

→       if (function.annotationEntries.isEmpty() &&
            !function.isOverride() &&
            !function.isPrivate()
        ) {
            reportIssue(function, "Add private modifier to function")
        }
    }

    private fun KtProperty.isCompanionObjectProperty(): Boolean {
        return getNonStrictParentOfType<KtObjectDeclaration>()?.isCompanion() ?: false
    }

    private fun reportIssue(element: PsiElement, message: String) =
        report(CodeSmell(issue, Entity.from(element), message))
}
```

Developer Experience

Порог входа когда всё настроено

gradle *nameOfCustomTask*

Как выглядит результат локально

```
ui-tests-linter - 6h 51min debt
    RestrictedKeywordRule - [Scenario must not contain 'try' expression] at /Users/nedoseykin/
    LargeScreenObjectRule - 313/110 - [Split a large ScreenObject into PageElement's and combine them on this SO] at
    RestrictedKeywordRule - [ScreenObject must not contain 'try' expression] at /Users/nedoseykin/
    TestClassPrivateMemberRule - [Make constant non-private and move it to private companion object] at /Users/nedose
    TestClassPrivateMemberRule - [Add private modifier to companion object] at /Users/nedoseykin/
    TestClassPrivateMemberRule - [Add private modifier to variable] at /Users/nedoseykin/
    TestClassPrivateMemberRule - [Add private modifier to function] at /Users/nedoseykin/
    RestrictedKeywordRule - [Test method must not contain 'try' expression] at /Users/nedoseykin/
    RestrictedKeywordRule - [Test method must not contain 'when' expression] at /Users/nedoseykin/
    RestrictedKeywordRule - [Test method must not contain 'if' expression] at /Users/nedoseykin/
    TmsLinkAnnotationRule - [Test class without @TmsLink annotation] at /Users/nedoseykin/
    IsVisibleUsageRule - [In general, 'isVisible' should not be used (use isDisplayed)] at /Users/nedoseykin/
    TmsLinkAnnotationRule - [Does not match pattern: @TmsLink\("(new_tests|retail_qa)/testcases/\d{1,6}"\)] at /Users
    TestClassNamingRule - [Test class names should match the pattern: [A-Z][a-zA-Z0-9]*(Test|Tests)] at /Users/nedose
    TestMethodNamingRule - [Test method name must not contain 'test'] at /Users/nedoseykin/
    TestMethodNamingRule - [Test method name should not consist of a single word] at /Users/nedoseykin/
    TestMethodNamingRule - 158/135 - [Test method name length should not be long (full qualifier)] at /Users/nedoseyk
    TestClassPrivateMemberRule - [Make property non-private and move it to private companion object] at /Users/nedose
```

Как выглядит результат в нашем CI

Success	[0] Configure Build Tree • Success
Failure	[1] Static analysis • Gradle exception (new); exit code 1 (Step: Static analyzers (Gradle)) (new)
Unknown	[2-1] Unit-tests (TIA) & Compile • Билд будет запущен автоматически! Для старта жду билды: [1] Static analysis
Unknown	[2-2] UI-tests (TIA) • Билд будет запущен автоматически! Для старта жду билды: [1] Static analysis
Unknown	[2-3] UI-tests (Custom) • Should be run manually
Failure	[3] Merger PR Check • Exit code 1 (Step: Curl check (Command Line))

robot at 11 Dec 2023, 22:40 Open issue ✓ Resolve Drop

Open full-report

Fix these Detekt violations

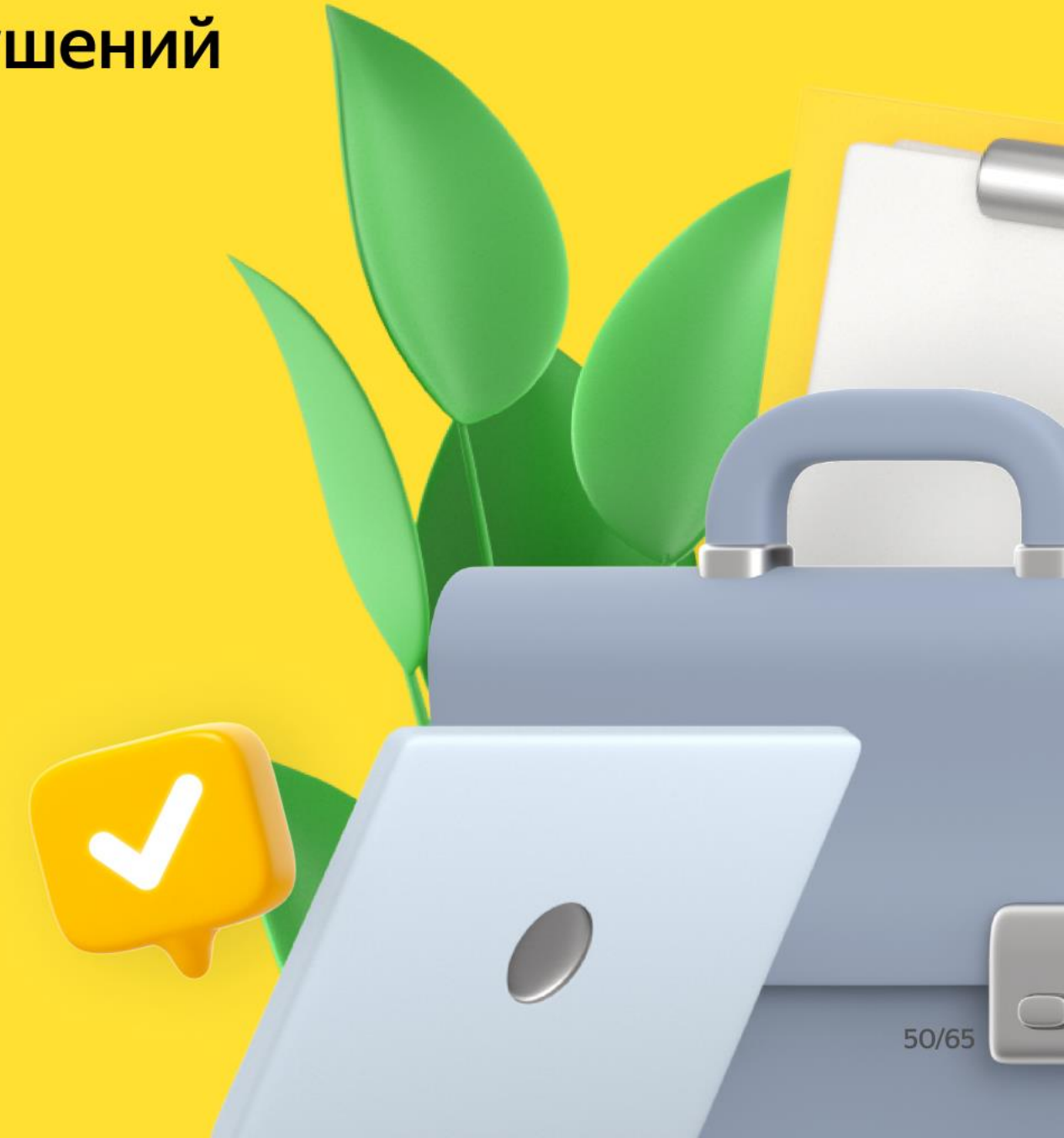
[TestMethodNamingRule] eats/src/androidTest/java/ru/eats/client/tests/cart_tests/pickup_cart/PickupCartTest.kt:43:5

Как выглядит результат в нашем CI

Способы игнорирования нарушений

и немного про стратегию работы с тех-долгом

01 Параметры правил в config.yml



config.yml

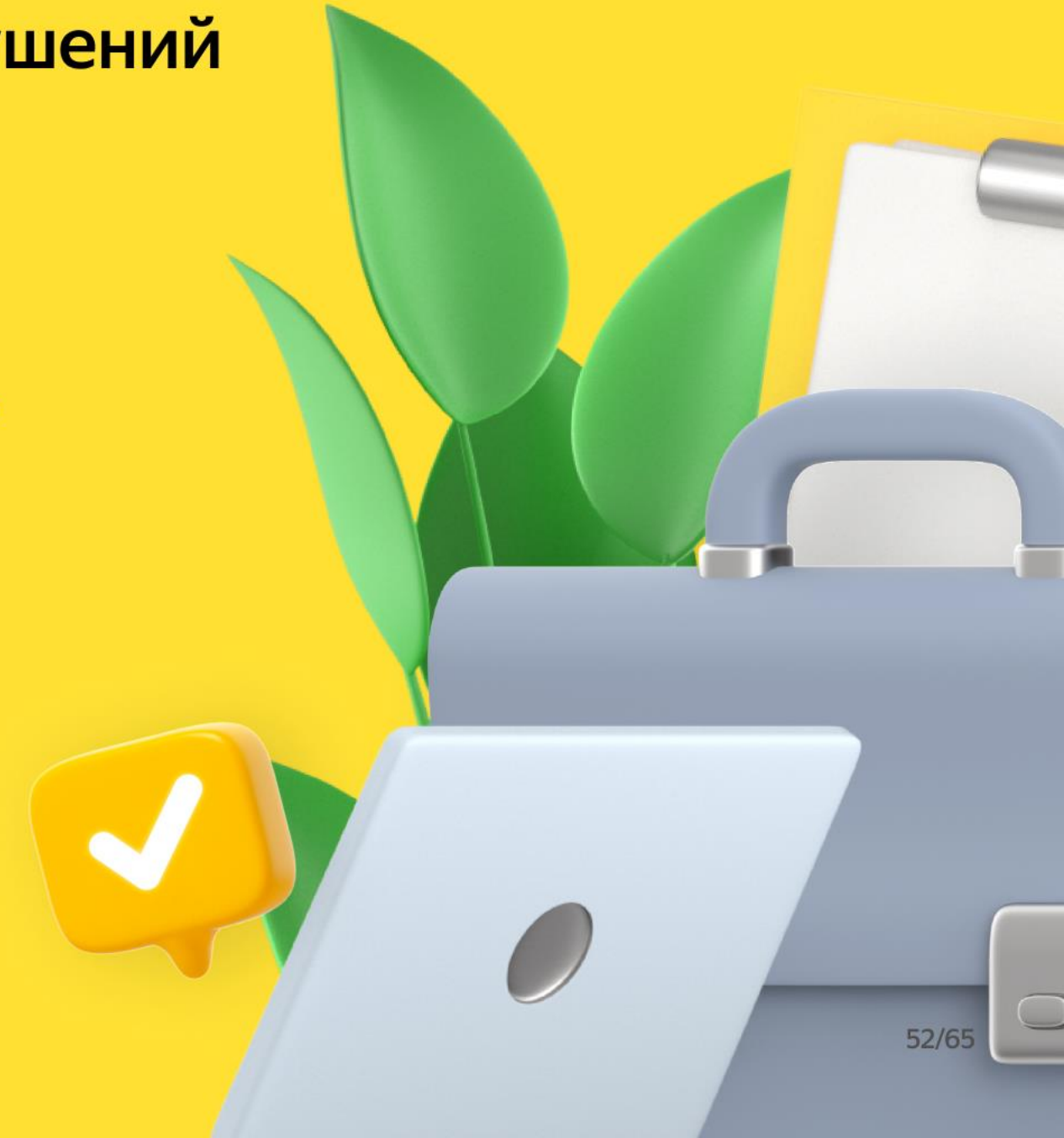
```
TestClassNamingRule:
  active: true
  includes: "**/androidTest/**/tests/**"
TestMethodNamingRule:
  active: true
  unexpectedWords: [ 'test' ]
  maxFullQualifierLength: 135
  includes: "**/androidTest/**/tests/**"
```

Способы игнорирования нарушений

и немного про стратегию работы с тех-долгом

01 Параметры правил в config.yml

02 Suppress-аннотации, но лучше их избегать



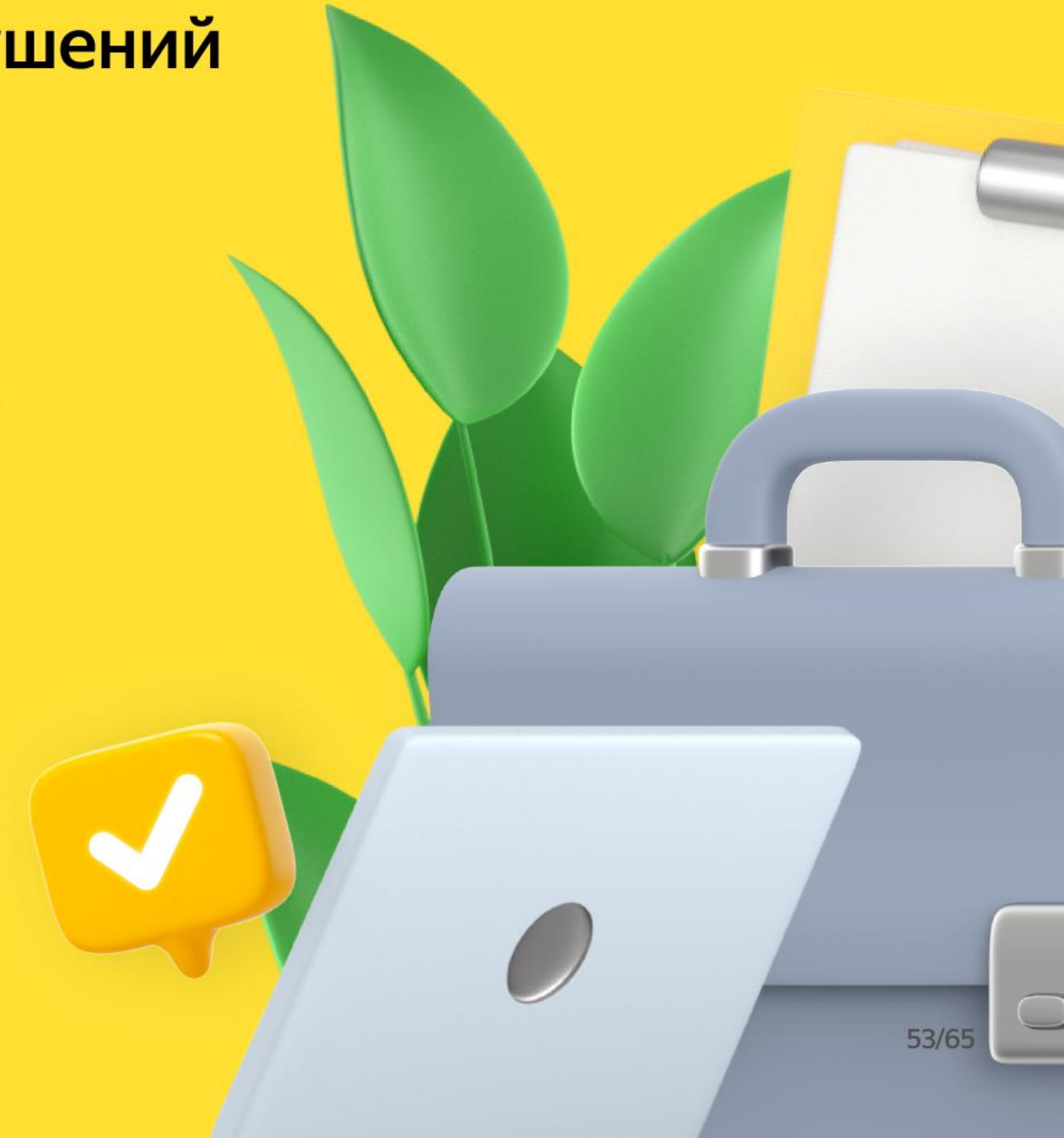
Способы игнорирования нарушений

и немного про стратегию работы с тех-долгом

01 Параметры правил в config.yml

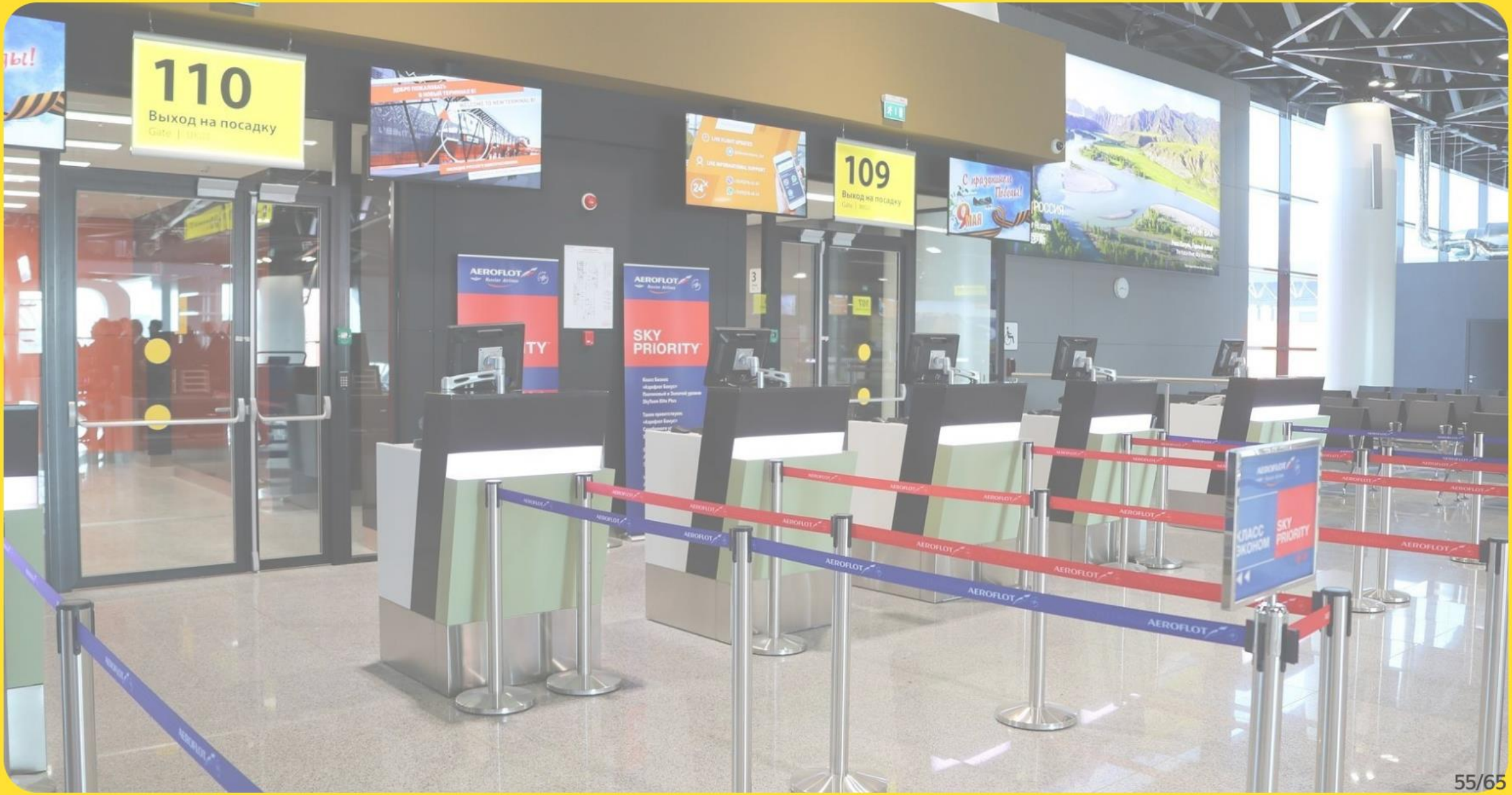
02 Suppress-аннотации, но лучше их избегать

03 baseline.xml для трудоёмкого в моменте



baseline.xml

```
<SmellBaseline>
  <ManuallySuppressedIssues>
  </ManuallySuppressedIssues>
  <CurrentIssues>
    <ID>TooManyFunctions:LargeClass.kt$io.gitlab.arturbosch.detekt.rules.complexity.LargeClass.kt</ID>
    <ID>ComplexMethod:DetektExtension.kt$DetektExtension$fun convertToArguments(): MutableList&lt;String&gt;</ID>
  </CurrentIssues>
</SmellBaseline>
```

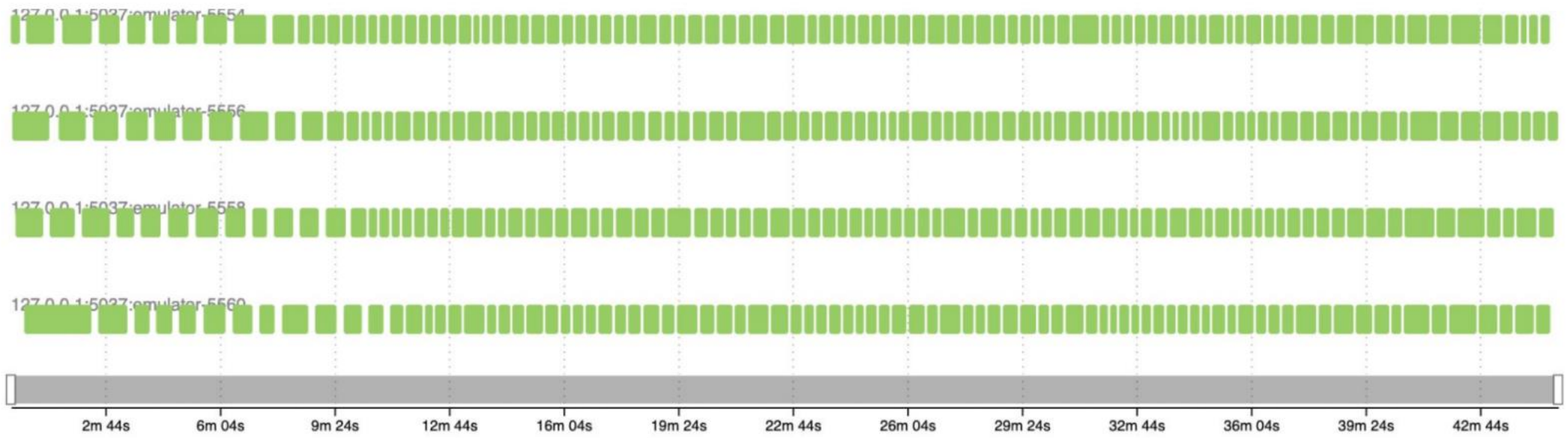


https://perm.rbc.ru/perm/freenews/5aeae7479a794779495d445d

Насколько ускорились

Этап	Дата	Число тестов	Длительность выполнения из Allure Report
До доработок	3.05.2023	261	*47m 03s*
Правим код и подкрепляем правилами	24.05.2023	261	*28m 15s*
Этап роста числа тестов	19.09.2023	326	*35m 41s*
Настоящее время	20.03.2024	368	*45m 01s*

Насколько стабилизировались



Выводы

01 Попробуйте в своём проекте статический анализ

Выводы

01 Попробуйте в своём проекте статический анализ

02 Анализ находит несоответствие кодстайлу и фактические ошибки

Выводы

01 Попробуйте в своём проекте статический анализ

02 Анализ находит несоответствие кодстайлу и фактические ошибки

03 Помогает сдвигать нахождение ошибки в коде с CI-этапа / ревью к локальной отладке

Выводы

01 Попробуйте в своём проекте статический анализ

02 Анализ находит несоответствие кодстайлу и фактические ошибки

03 Помогает сдвигать нахождение ошибки в коде с CI-этапа / ревью к локальной отладке

04 В нашем случае с его помощью смогли закрепить успех от ускорения и стабилизации тестов

Полезные ссылки

Хабр вариант доклада

Примеры правил

Доклад: непрерывный статический анализ кода

Полезные ссылки

Хабр вариант доклада

Примеры правил

Доклад: непрерывный статический анализ кода

Полезные ссылки

Хабр вариант доклада

Примеры правил

Доклад: непрерывный статический анализ кода

Яндекс Еда

Недосейкин Николай

Инженер мобильной платформы Яндекс Еды