Роман Аймалетдинов

Software engineer in Android

Как я code-coverage внедрял,

да gradle plugin для baseline писал

Contact:

aymaletdinov.job@email.com

linkedin.com/in/raymaletdinov/

Цель:

🔵 Вы сможете внедрить code-coverage в CI pipeline за несколько дней.

Ctrl+C/Ctrl+V из презентации, и все готово.

О проекте:

- 54 модуля
- 1838 unit теста
- N UI/Integration теста
- GitHub Actions

О чем мы будем поговорим?

1. что такое code-coverage
2. как засетапить kover
3. почему kover недостаточно?
4. что такое impact analysis и как его засетапить?

О чем мы будем поговорим?

5 напишем свой custom gradle plugin

6 свяжем kover + наш plugin + impact analysis

7 настроим CI

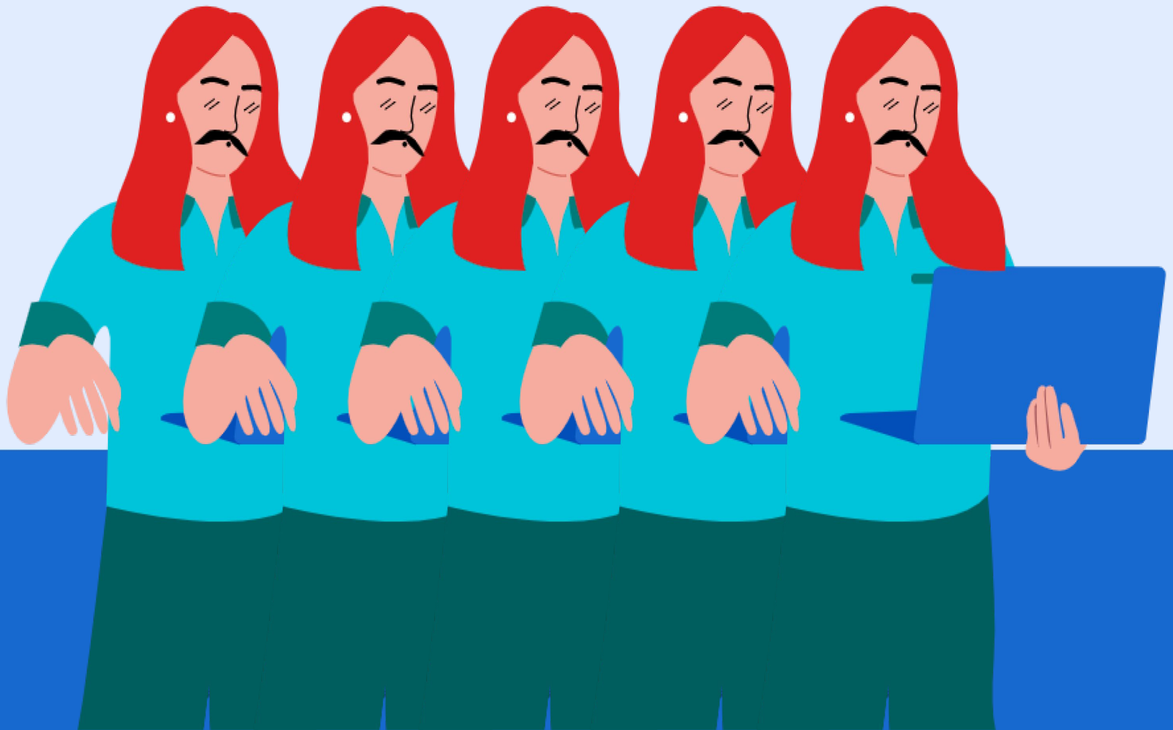
8 поговорим о результатах фичи

Что такое code-coverage?

Метрика отображающая соотношение написанного кода к количеству тестов проверяющих этот код.

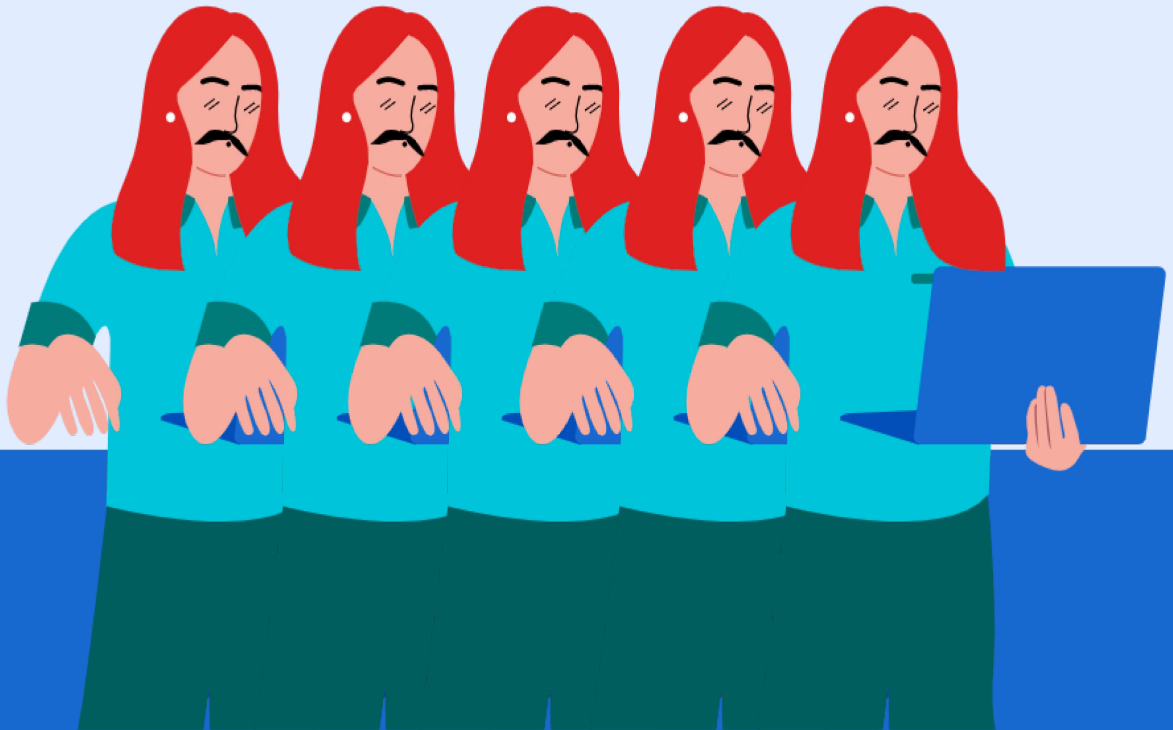
Что такое code-coverage?

Метрика отображающая соотношение написанного кода к количеству тестов проверяющих этот код.



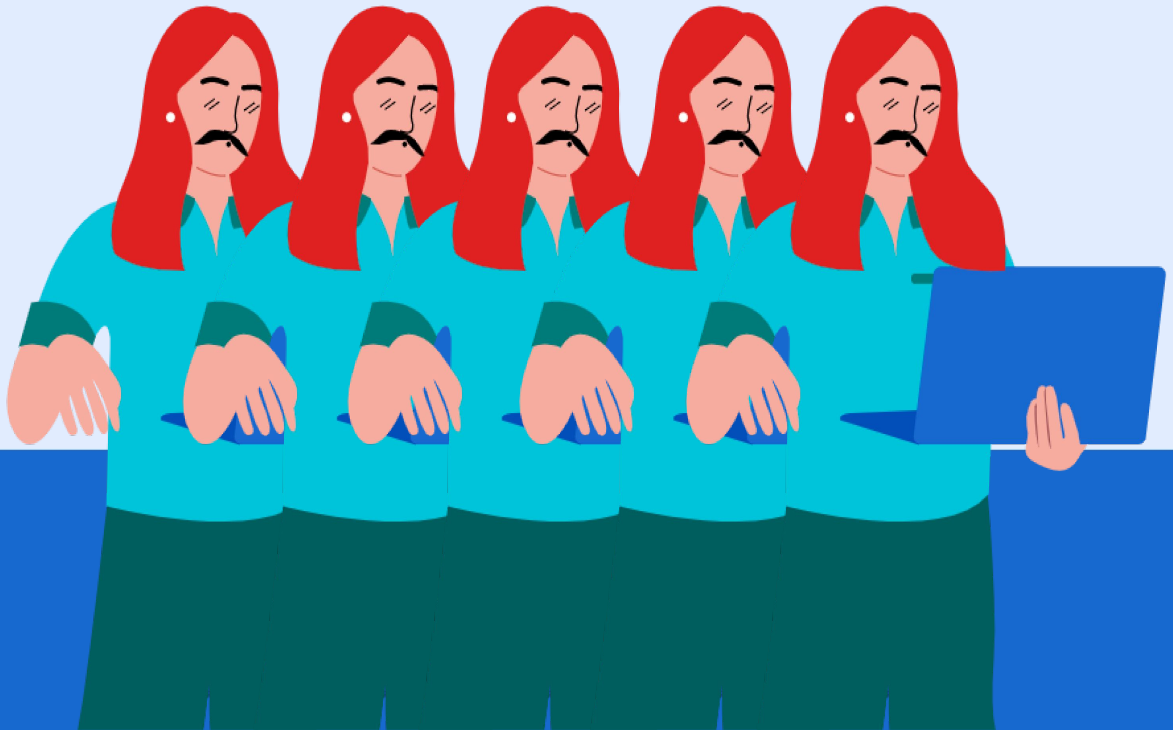
Что такое code-coverage?

Метрика отображающая соотношение написанного кода к количеству тестов проверяющих этот код.



Что такое code-coverage?

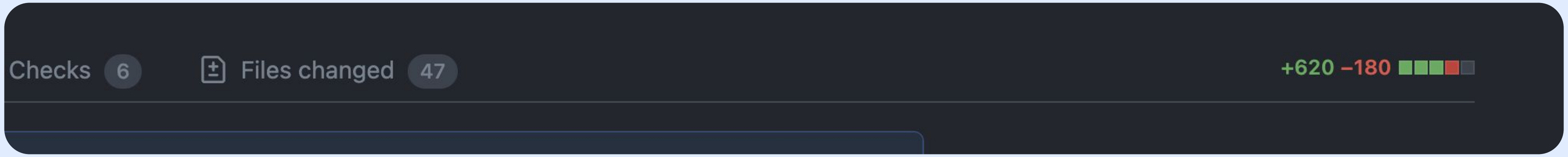
Метрика отображающая соотношение написанного кода к количеству тестов проверяющих этот код.



Что такое code-coverage?

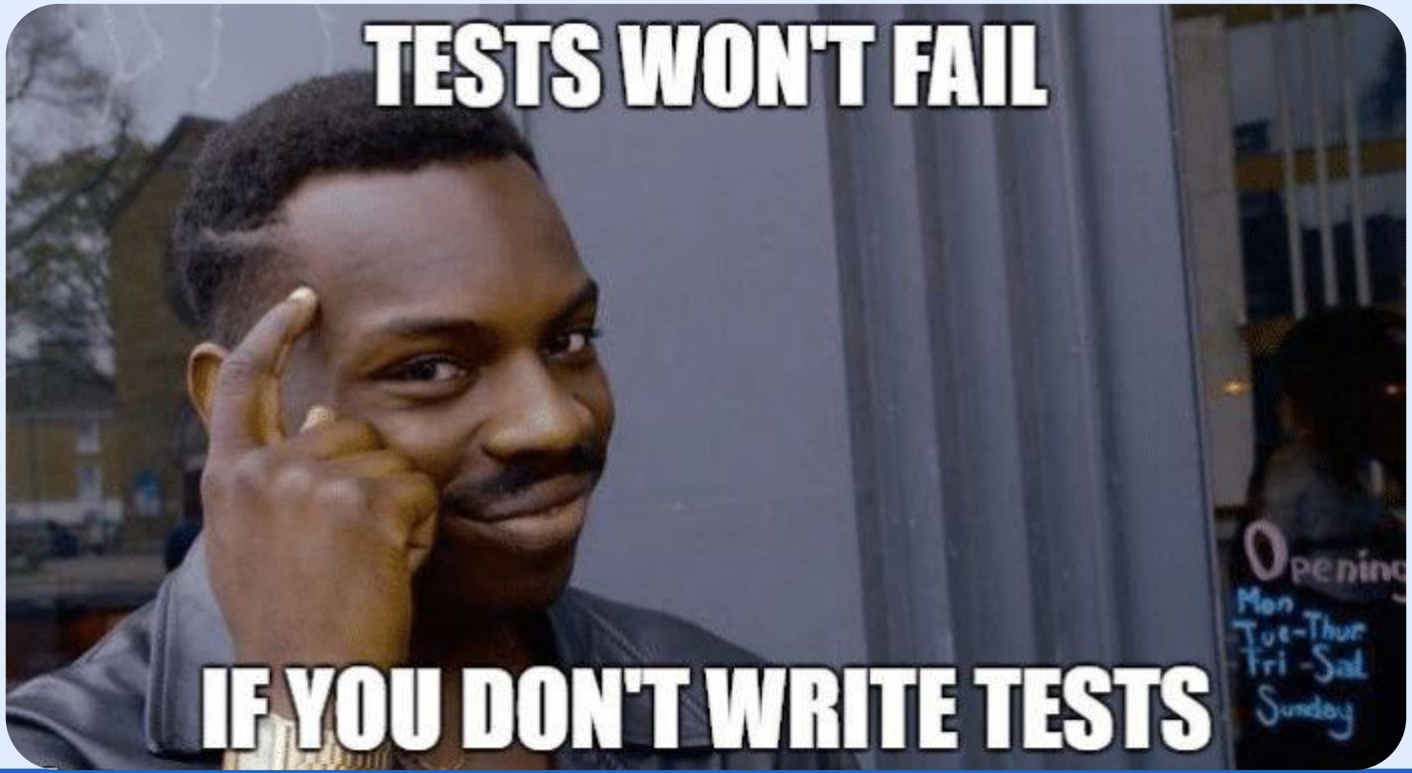
Как вы считаете?

Я тут на все классы написал тесты?



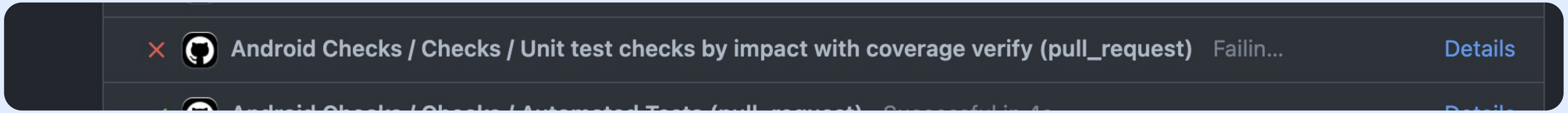
Что такое code-coverage?

На все, что вспомнил и на все, на что писать тесты было не лень 😅

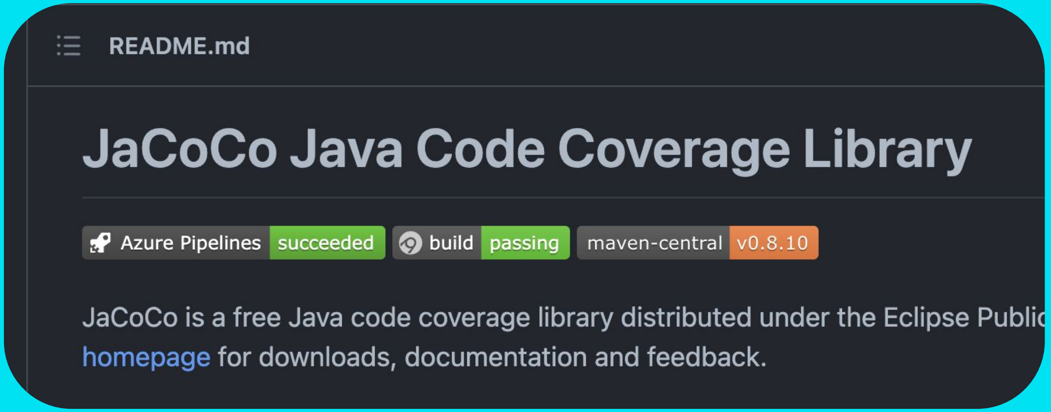


Что такое code-coverage?

Но написать все-таки пришлось..


Android Checks / Checks / Unit test checks by impact with coverage verify (pull_request) Failin... Details

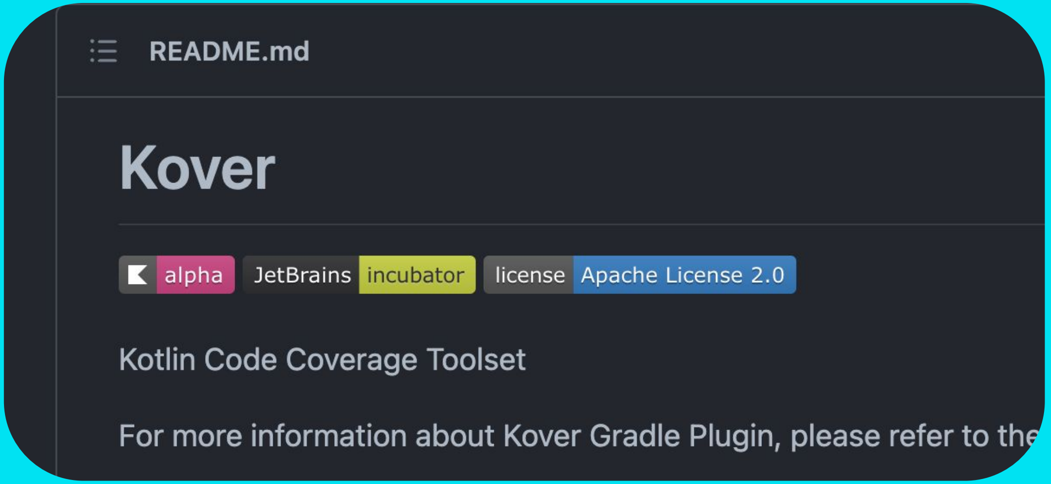
Как узнать свой code-coverage?


README.md

JaCoCo Java Code Coverage Library

Azure Pipelines succeeded | build passing | maven-central v0.8.10

JaCoCo is a free Java code coverage library distributed under the Eclipse Public
homepage for downloads, documentation and feedback.

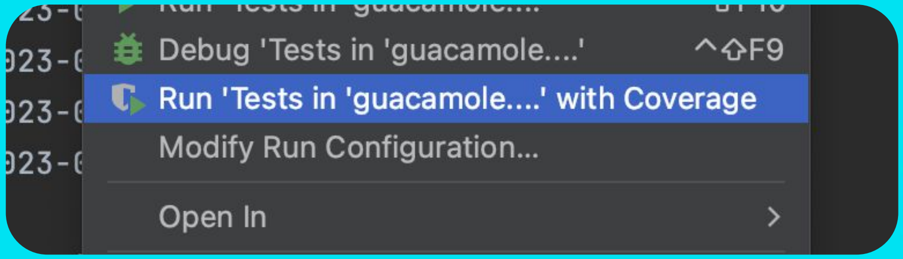

README.md

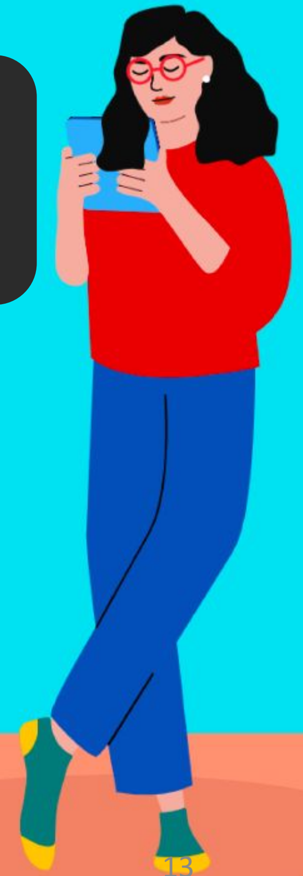
Kover

alpha | JetBrains incubator | license Apache License 2.0

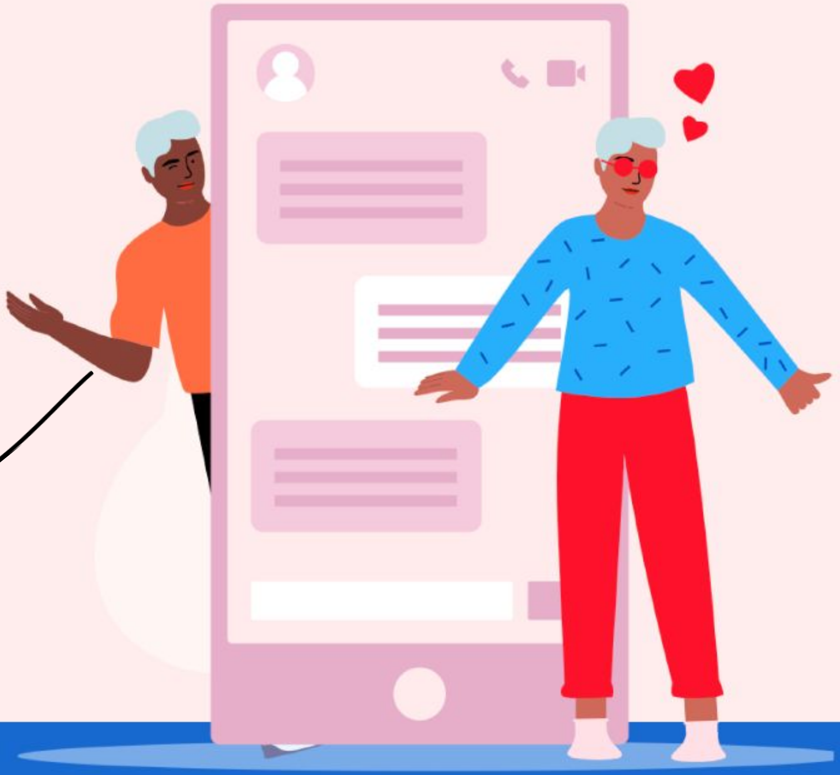
Kotlin Code Coverage Toolset

For more information about Kover Gradle Plugin, please refer to the


Debug 'Tests in 'guacamole....' ^⇧F9
Run 'Tests in 'guacamole....' with Coverage
Modify Run Configuration...
Open In



Требования к фиче



Мы можем посмотреть покрытие

Требования к фиче

Если модуль новый, вся логика должна быть покрыта тестами

Мы можем посмотреть покрытие

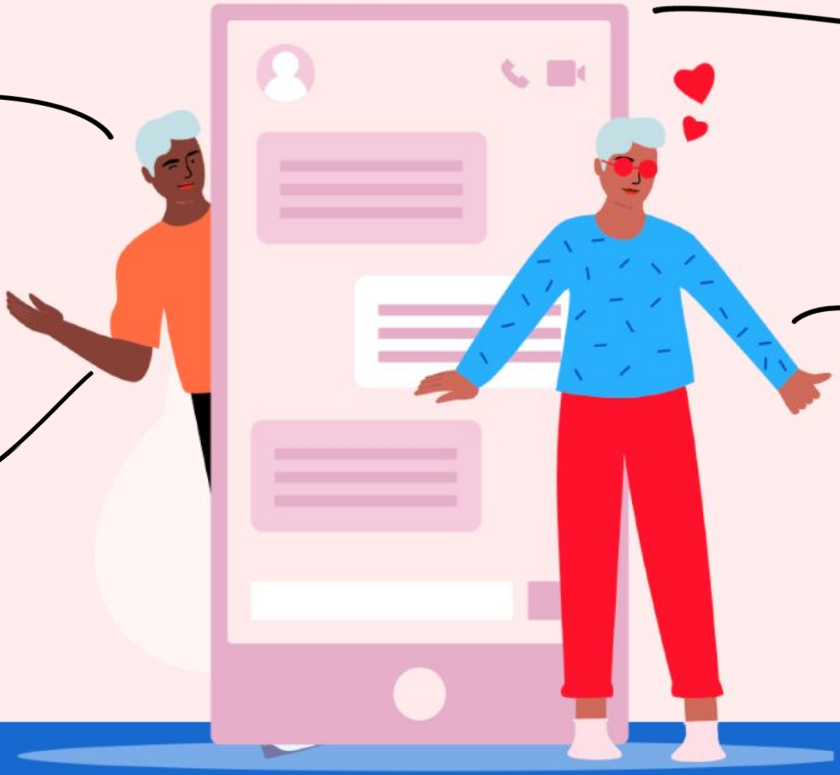
Требования к фиче

Если модуль новый, вся логика должна быть покрыта тестами

Мы можем посмотреть покрытие

Если модуль старый, то вся новая логика должна быть покрыта

Требования к фиче



Если модуль новый, вся логика должна быть покрыта тестами

Мы можем посмотреть покрытие

Хотелось бы что бы все это было под impact analysis

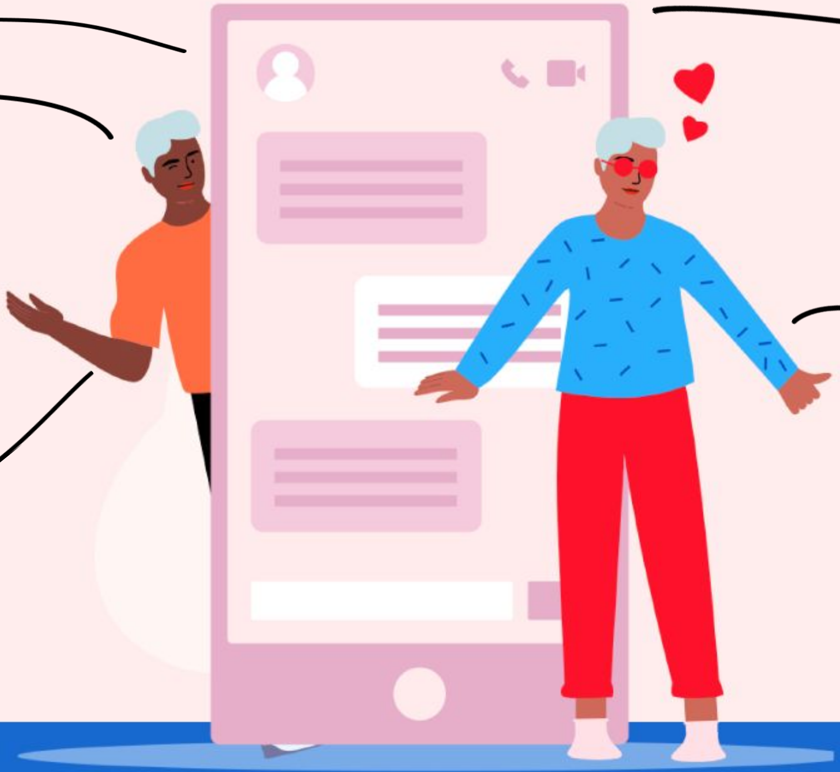
Если модуль старый, то вся новая логика должна быть покрыта

Требования к фиче

Процент покрытия проекта улучшается каждый PR

Если модуль новый, вся логика должна быть покрыта тестами

Мы можем посмотреть покрытие



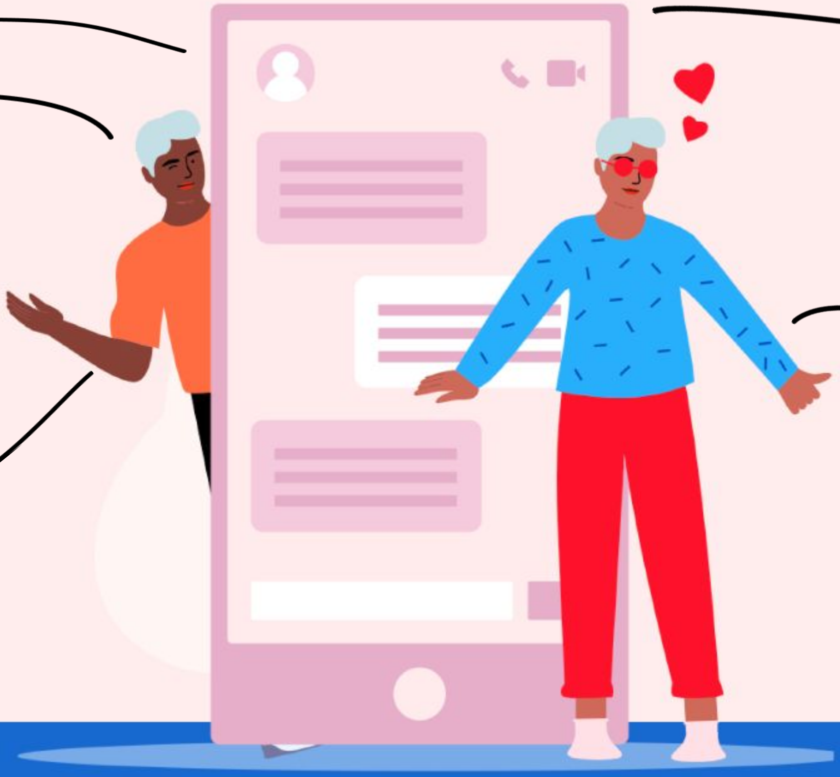
Хотелось бы что бы все это было под impact analysis

Если модуль старый, то вся новая логика должна быть покрыта

Требования к фиче

Процент покрытия проекта улучшается каждый PR

Соблюдение правил контролирует CI, а не человек



Если модуль новый, вся логика должна быть покрыта тестами

Хотелось бы что бы все это было под impact analysis

Мы можем посмотреть покрытие

Если модуль старый, то вся новая логика должна быть покрыта

Как засетапить ковер?

Добавляем в gradle строки:

```
plugins {
    // ..
    id 'org.jetbrains.kotlinx.kover'
}
```

Как засетапить ковер?

```
koverReport {
   filters {
       excludes {
           classes(
               // Android
               "*Application*",
               "*Application\$*",
               "*Activity*",
               // ..
           )
          }
   }
}
```

Настроим фильтр

Как засетапить ковер?

```
class AndroidModulePlugin : Plugin<Project> {

  override fun apply(project: Project) {
     // Add Kotlin
     project.plugins.apply("kotlin-android")

     // Add Kover support
     project.plugins.apply("org.jetbrains.kotlinx.kover")
     project.apply(
        from = "${project.rootProject.rootDir}/tools/test-coverage/kover_filter.gradle"
     )


     //..
```

Как засетапить ковер?

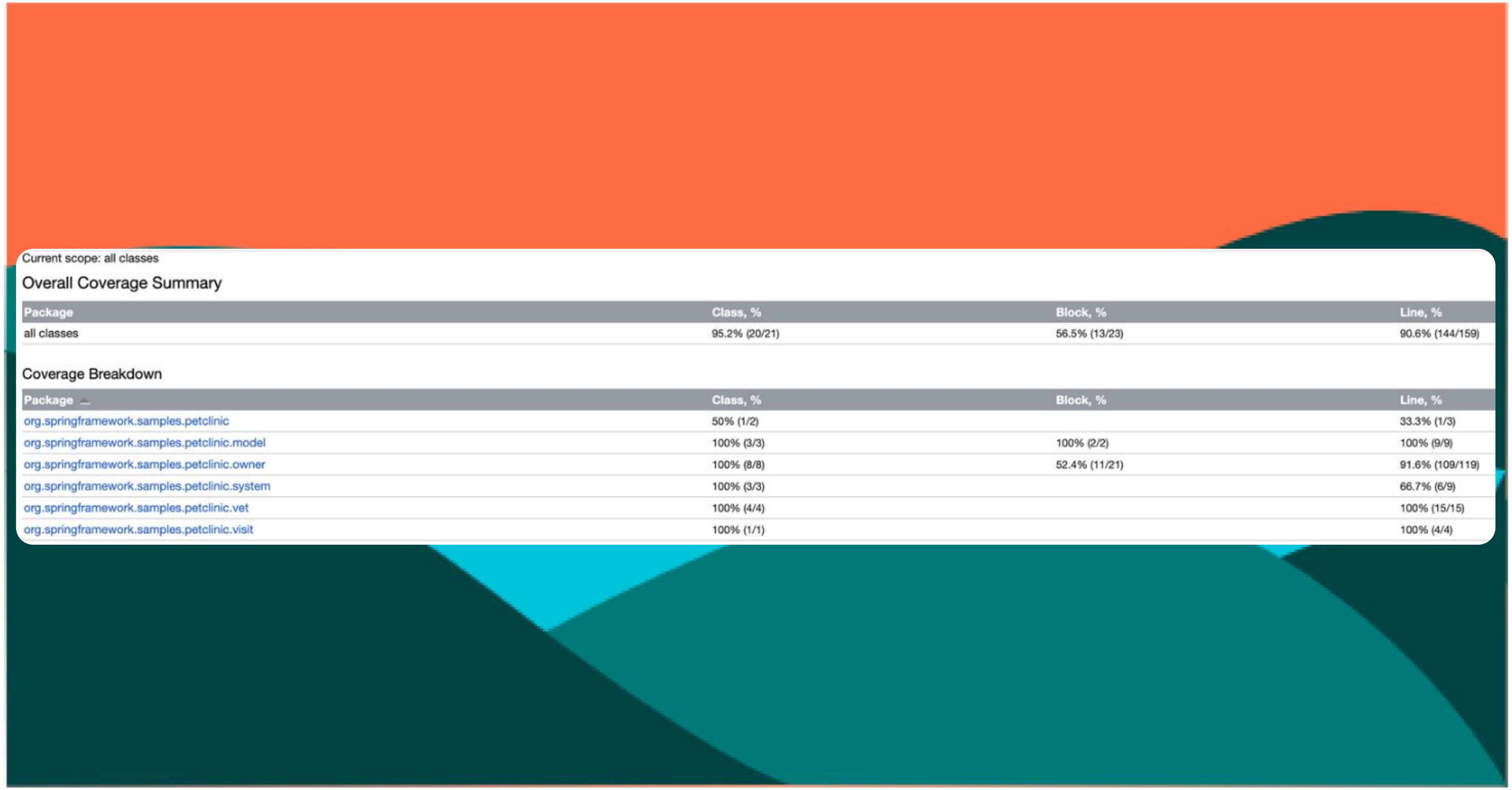
```kotlin
class AndroidModulePlugin : Plugin<Project> {

  override fun apply(project: Project) {
     // Add Kotlin
     project.plugins.apply("kotlin-android")

     // Add Kover support
     project.plugins.apply("org.jetbrains.kotlinx.kover")
     project.apply(
        from = "${project.rootProject.rootDir}/tools/test-coverage/kover_filter.gradle"
     )

     //..
```

Как засетапить ковер?

Готово

```
roman.aimaletdinov guacamole % ./gradlew core:koverHtmlReport
Configuration on demand is an incubating feature.

> Task :core:koverHtmlReport
Kover: HTML report for ':core'
file:///Users/roman.aimaletdinov/StudioProjects/global/android/guacamole/core/build/reports/kover/html/index
.html
```

Как засетапить ковер?



Current scope: all classes

Overall Coverage Summary

Package	Class, %	Block, %	Line, %
all classes	95.2% (20/21)	56.5% (13/23)	90.6% (144/159)

Coverage Breakdown

Package	Class, %	Block, %	Line, %
org.springframework.samples.petclinic	50% (1/2)		33.3% (1/3)
org.springframework.samples.petclinic.model	100% (3/3)	100% (2/2)	100% (9/9)
org.springframework.samples.petclinic.owner	100% (8/8)	52.4% (11/21)	91.6% (109/119)
org.springframework.samples.petclinic.system	100% (3/3)		66.7% (6/9)
org.springframework.samples.petclinic.vet	100% (4/4)		100% (15/15)
org.springframework.samples.petclinic.visit	100% (1/1)		100% (4/4)

Как засетапить ковер?

Что если мы не хотим тестировать функцию?

```
override fun error(e: Exception) {
   errorHandler?.invoke()
}
```

Или целый класс?

```
class DebugMenu @Inject internal constructor(
   private val schedulerProvider: SchedulerProvider,
...
```

Как засетапить ковер?

Что если мы не хотим тестировать функцию?

```
@ExcludeFromCoverageReport
override fun error(e: Exception) {
   errorHandler?.invoke()
}
```

Или целый класс?

```
@ExcludeFromCoverageReport
class DebugMenu @Inject internal constructor(
   private val schedulerProvider: SchedulerProvider,
...
```

Как засетапить ковер?

```
koverReport {
   filters {
       excludes {
           classes(
               // Android
               "*Application*",
               "*Application\$*",
               "*Activity*",
               // ..
           )

           annotatedBy("*ExcludeFromCoverageReport")
       }
   }
}
```

Включим аннотации

Как засетапить ковер?

```
koverReport {
   filters {
       excludes {
           classes(
               // Android
               "*Application*",
               "*Application\$*",
               "*Activity*",
               // ..
           )

           annotatedBy("*ExcludeFromCoverageReport")
       }
   }
}
```

Включим аннотации

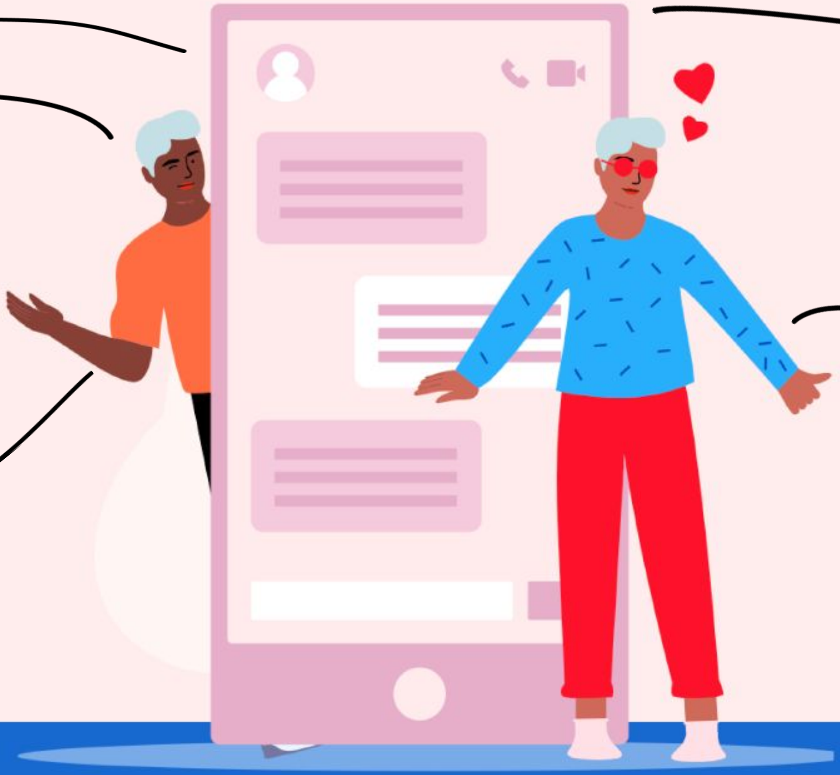
Требования к фиче

Процент покрытия проекта улучшается каждый PR

Соблюдение правил контролирует CI, а не человек

Если модуль новый, вся логика должна быть покрыта тестами

Хотелось бы что бы все это было под impact analysis

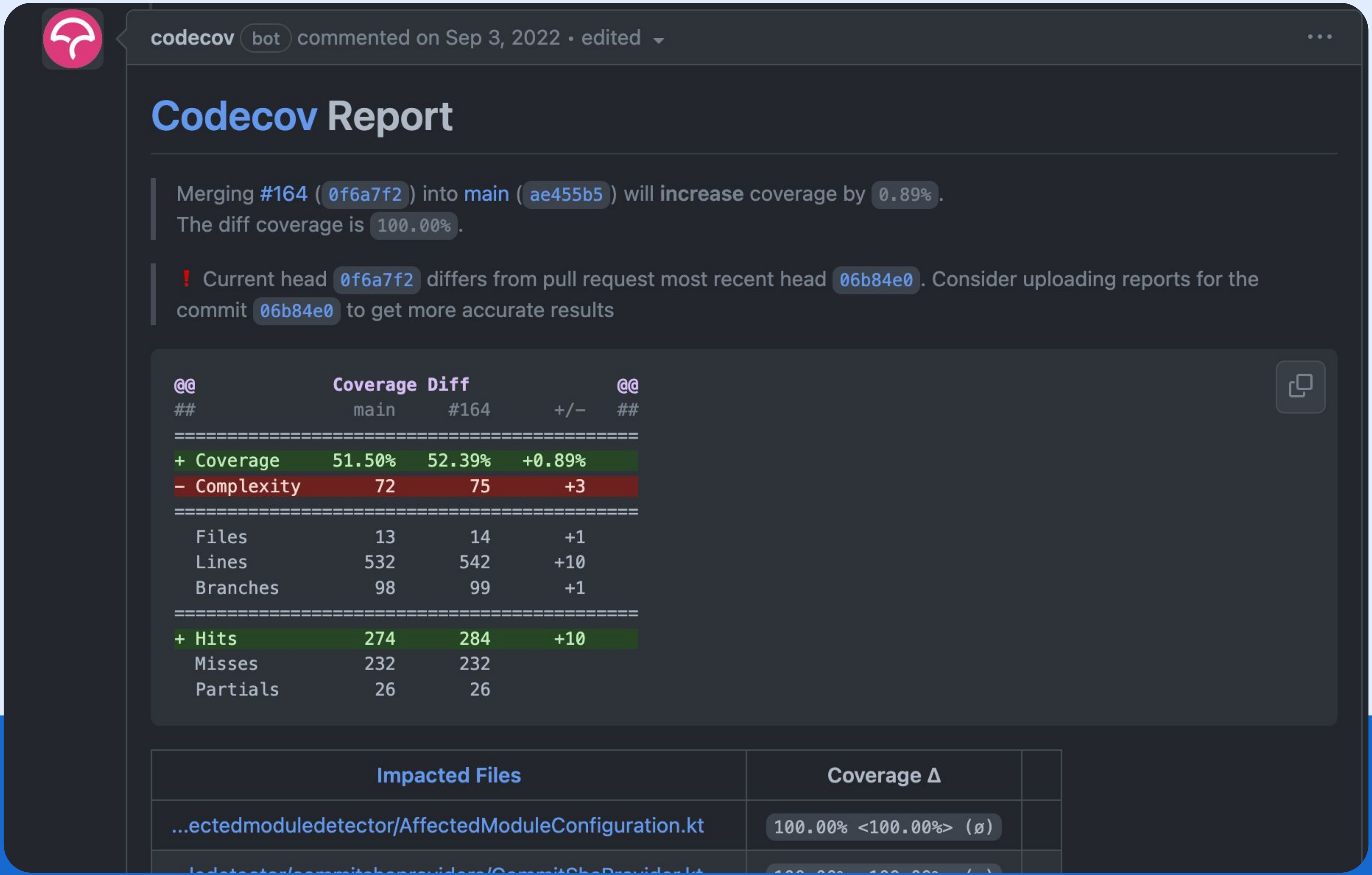


Мы можем посмотреть покрытие ✅

Если модуль старый, то вся новая логика должна быть покрыта

А как заставить команду писать тесты?

А как заставить команду писать тесты?



codecov bot commented on Sep 3, 2022 • edited

Codecov Report

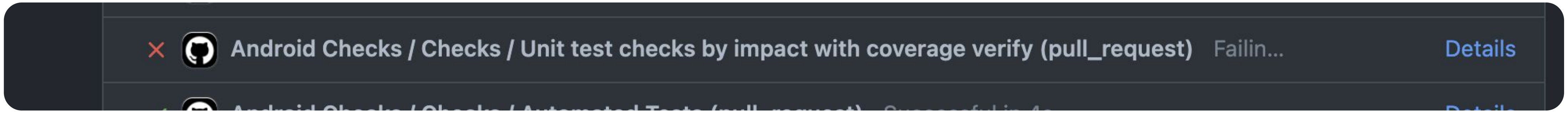
> Merging #164 (0f6a7f2) into main (ae455b5) will **increase** coverage by 0.89%.
> The diff coverage is 100.00%.

> ❗ Current head 0f6a7f2 differs from pull request most recent head 06b84e0. Consider uploading reports for the commit 06b84e0 to get more accurate results

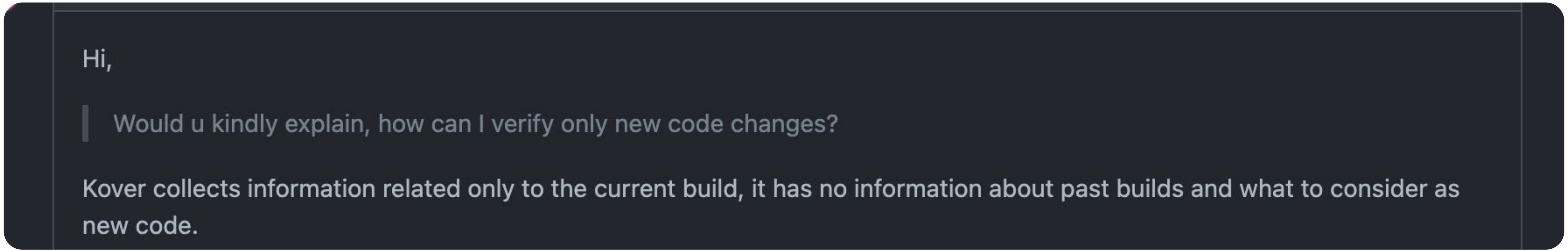
```
@@            Coverage Diff            @@
##             main     #164     +/-   ##
==========================================
+ Coverage   51.50%   52.39%   +0.89%
- Complexity     72       75       +3
==========================================
  Files          13       14       +1
  Lines         532      542      +10
  Branches       98       99       +1
==========================================
+ Hits          274      284      +10
  Misses        232      232
  Partials       26       26
```

Impacted Files	Coverage Δ	
...ectedmoduledetector/AffectedModuleConfiguration.kt	100.00% <100.00%> (ø)	

А как заставить команду писать тесты?


✕ Android Checks / Checks / Unit test checks by impact with coverage verify (pull_request) Failin... Details

Проверка только новых файлов



Hi,

> Would u kindly explain, how can I verify only new code changes?

Kover collects information related only to the current build, it has no information about past builds and what to consider as new code.

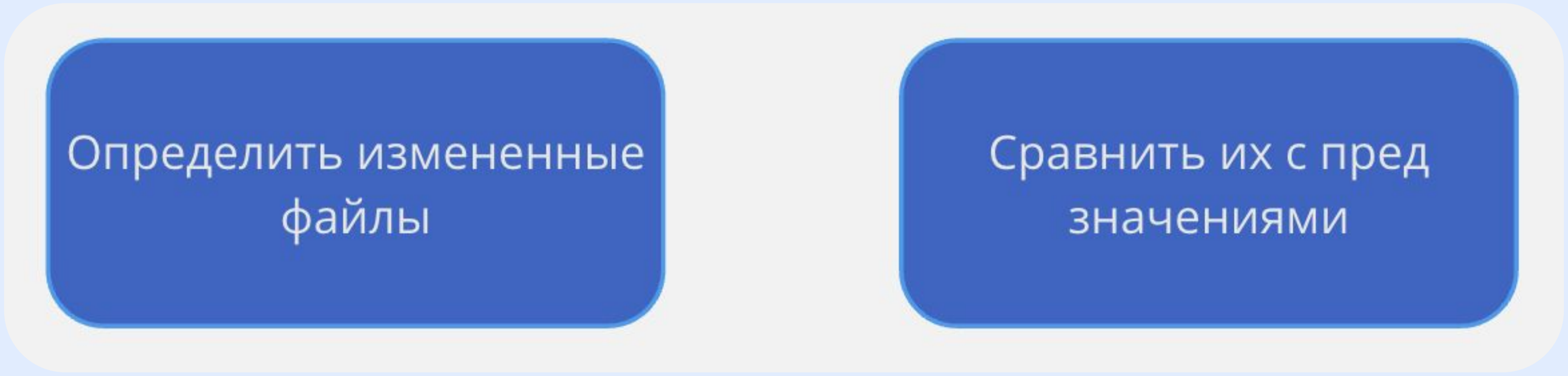
Custom Gradle plugin

А что это такое?

Gradle plugin - это некоторый контейнер хранящий инструкции к выполнению во время сборки проекта.

Требования к custom gradle plugin

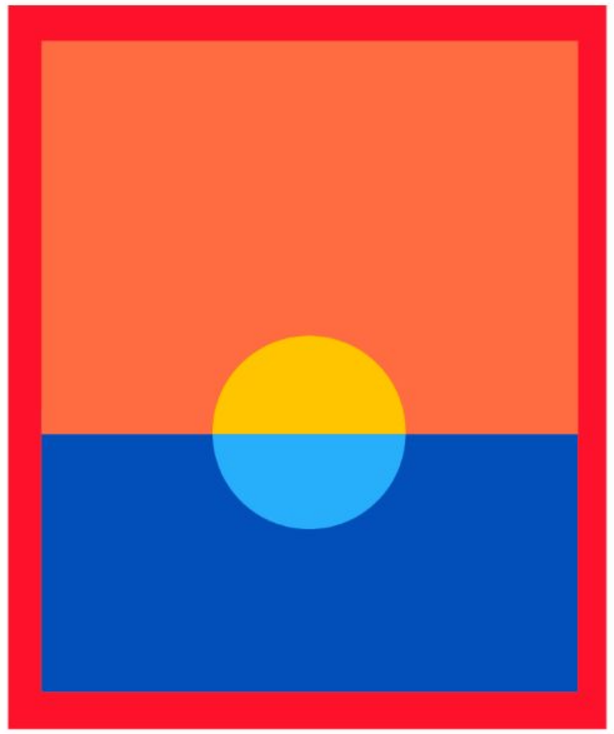
Какая функциональность нам нужна?



Требования к custom gradle plugin

Какие команды нам нужны?

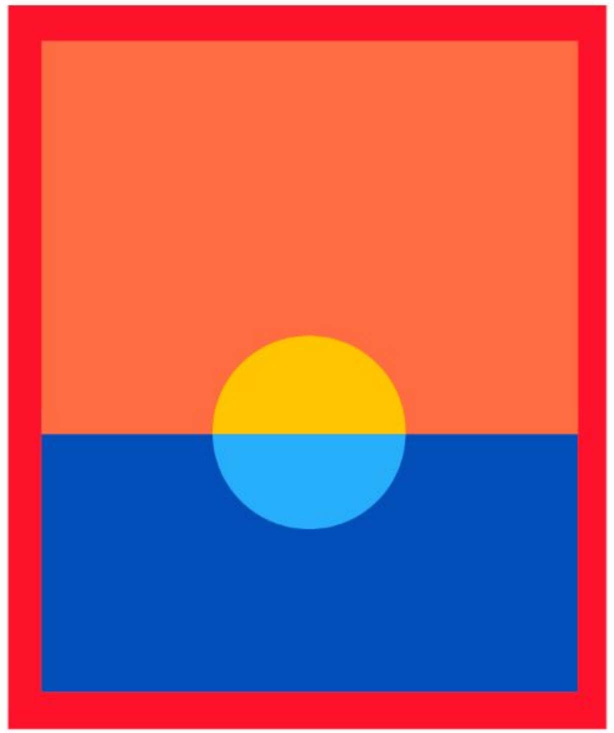
- Команда создания baseline
- Команда создания репорта
- Команда проверки покрытия



Требования к custom gradle plugin

Специфичные требования?

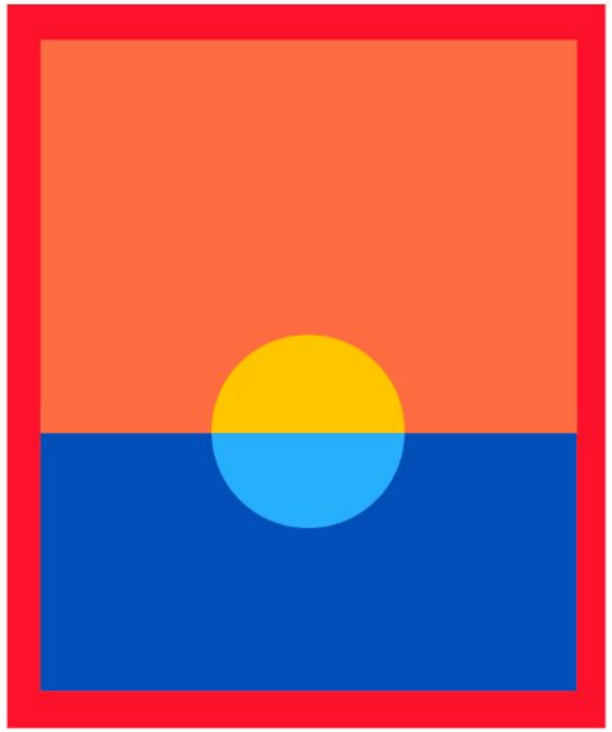
- Baseline должен быть удобочитаемым
- Плагин выкидывает исключение если покрытие уменьшилось
- Команды запускают kover вне зависимости от типа модуля и flavor



Требования к custom gradle plugin

Специфичные требования?

- Baseline должен быть удобочитаемым
- Плагин выкидывает исключение если покрытие уменьшилось
- *Команды запускают kover вне зависимости от типа модуля и flavor*



Требования к custom gradle plugin

```
./gradlew app:tasks

...
koverHtmlReport
koverHtmlReportCatDebug
koverHtmlReportCatRelease
koverHtmlReportDogDebug
koverHtmlReportDogRelease
```

модуль с flavor

просто jvm модуль без buildType & flavor

```
./gradlew util:tasks

...
koverHtmlReport
```

```
./gradlew feature:tasks

...
koverHtmlReport
koverHtmlReportDebug
koverHtmlReportRelease
```

модуль с buildType

Требования к custom gradle plugin

```
./gradlew app:tasks

...
koverHtmlReport
koverHtmlReportCatDebug
koverHtmlReportCatRelease
koverHtmlReportDogDebug
koverHtmlReportDogRelease
```

модуль с flavor

просто jvm модуль без buildType & flavor

```
./gradlew util:tasks

...
koverHtmlReport
```

```
./gradlew feature:tasks

...
koverHtmlReport
koverHtmlReportDebug
koverHtmlReportRelease
```

модуль с buildType

Требования к custom gradle plugin

```
./gradlew app:tasks

...
koverHtmlReport
koverHtmlReportCatDebug
koverHtmlReportCatRelease
koverHtmlReportDogDebug
koverHtmlReportDogRelease
```

модуль с flavor

просто jvm модуль без buildType & flavor

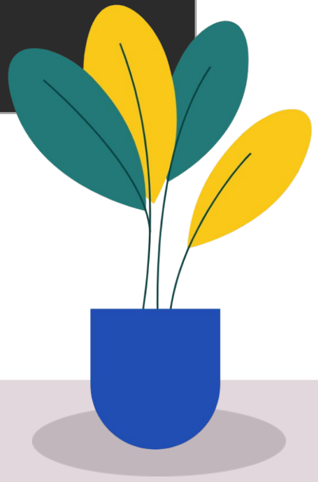
```
./gradlew util:tasks

...
koverHtmlReport
```

```
./gradlew feature:tasks

...
koverHtmlReport
koverHtmlReportDebug
koverHtmlReportRelease
```

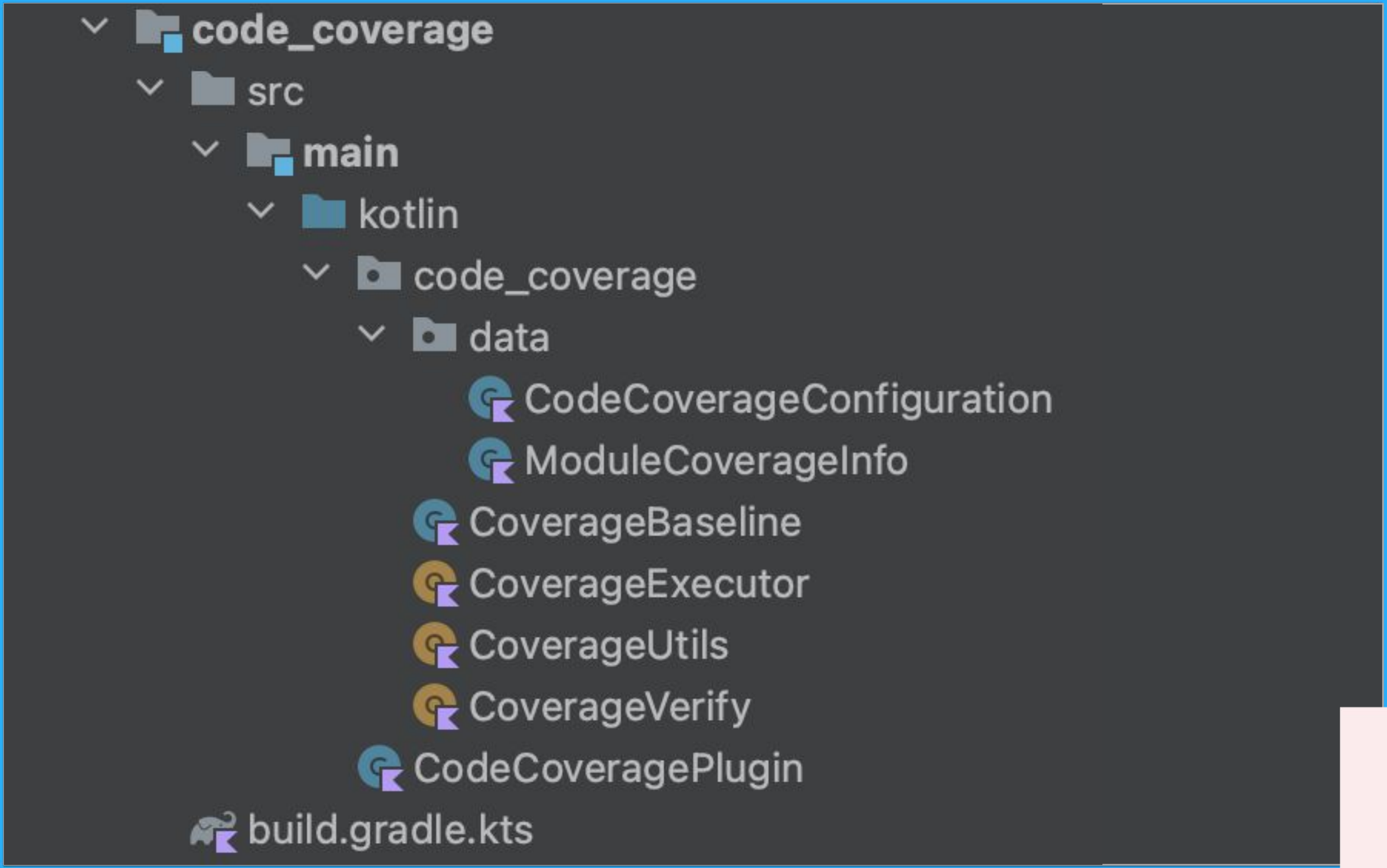
модуль с buildType

Требования к custom gradle plugin

```
./gradlew [any]:tasks

...
coverageHtmlReport
coverageVerify
coverageUpdateBaseline
```

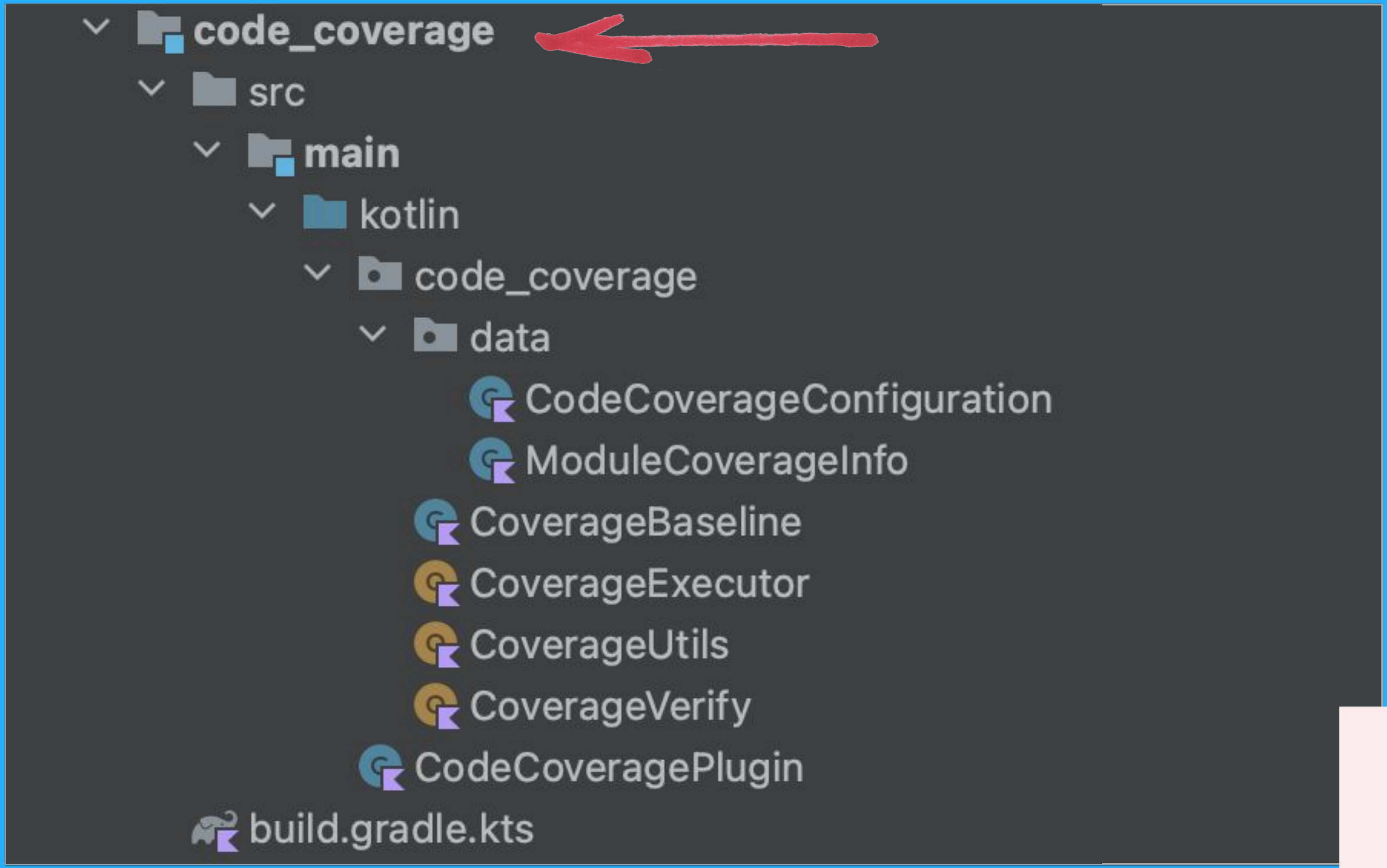


Custom Gradle Plugin Начало



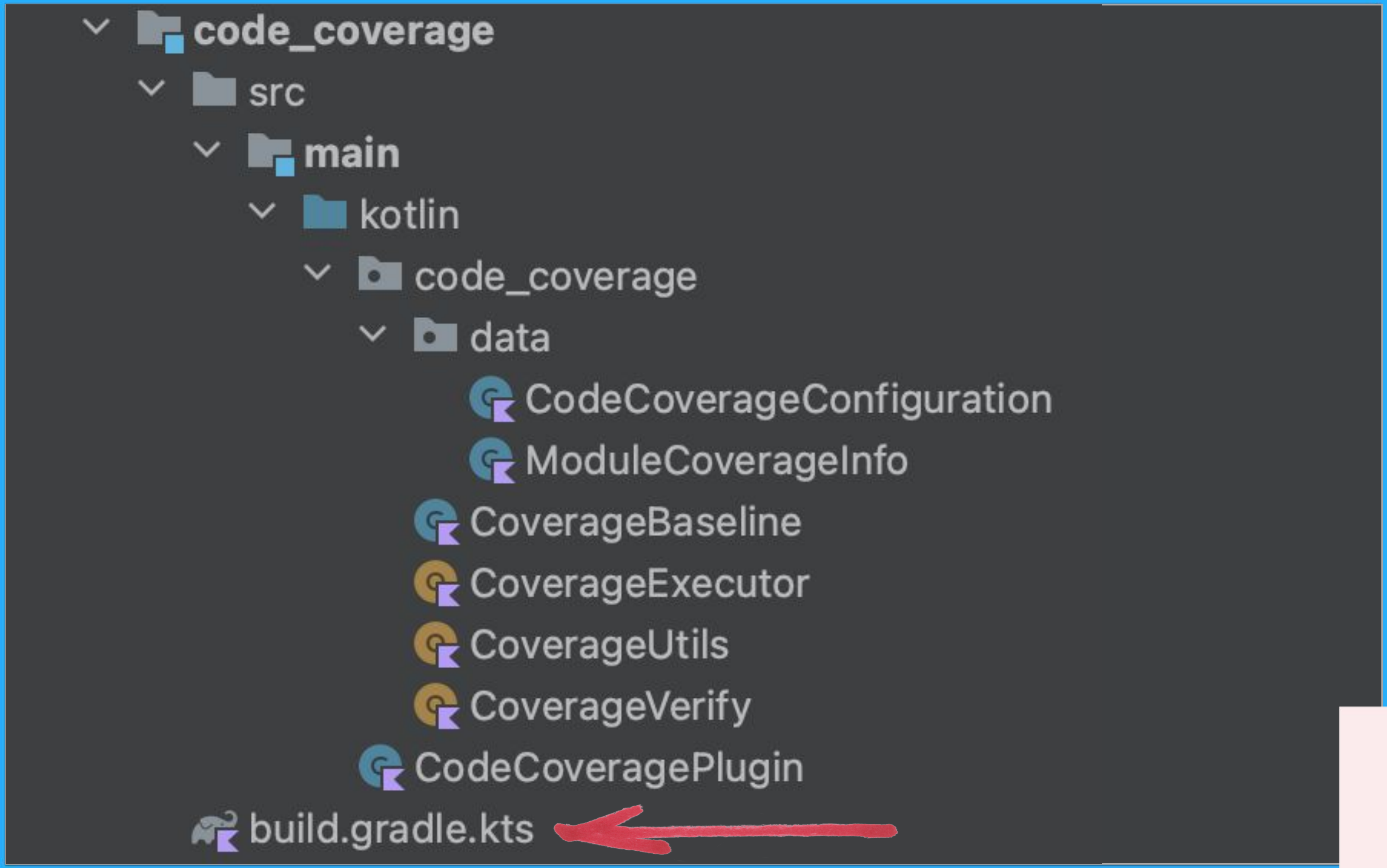
```
code_coverage
  src
    main
      kotlin
        code_coverage
          data
            CodeCoverageConfiguration
            ModuleCoverageInfo
          CoverageBaseline
          CoverageExecutor
          CoverageUtils
          CoverageVerify
        CodeCoveragePlugin
  build.gradle.kts
```

Custom Gradle Plugin Начало



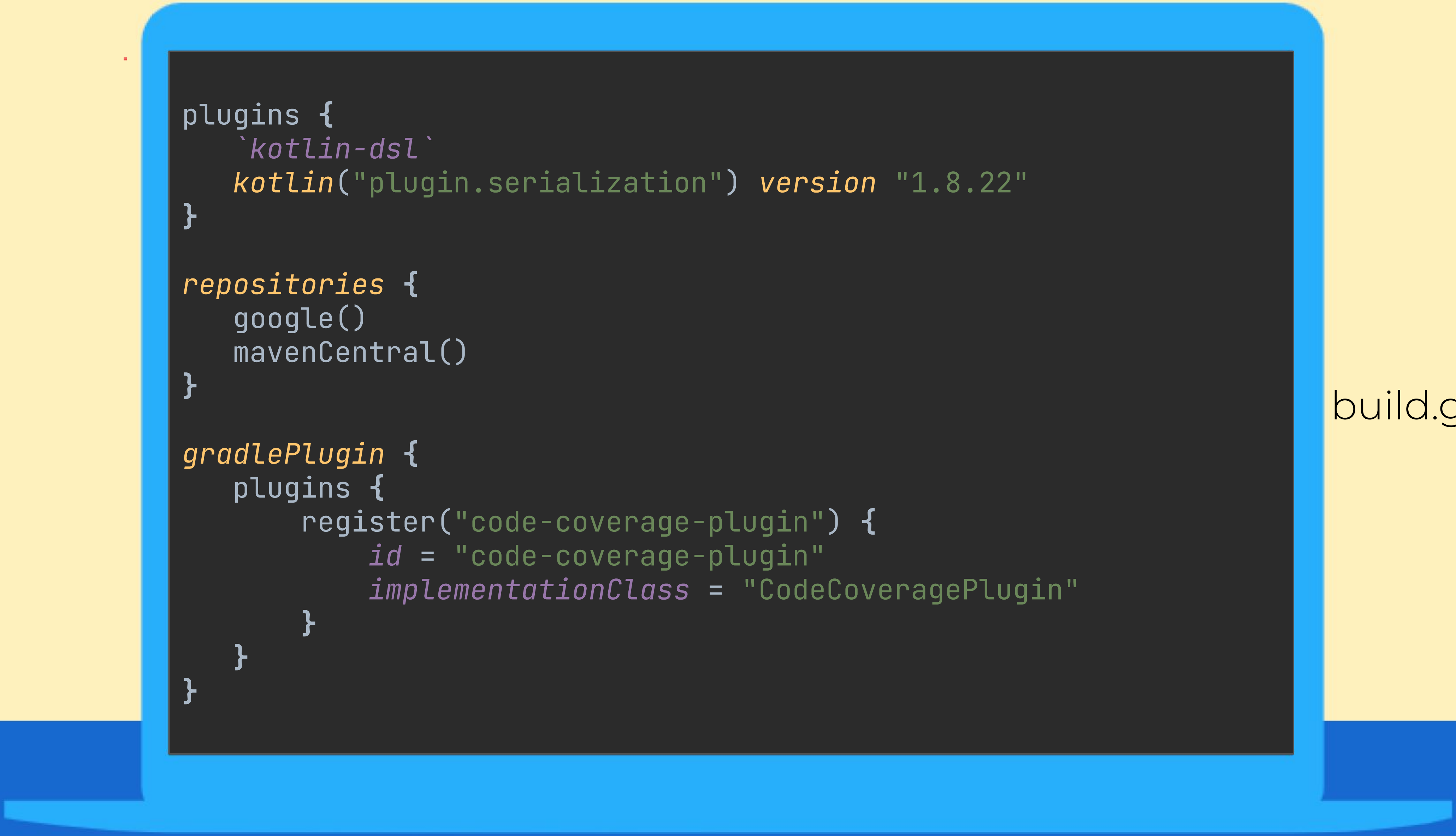
```
code_coverage
  src
    main
      kotlin
        code_coverage
          data
            CodeCoverageConfiguration
            ModuleCoverageInfo
          CoverageBaseline
          CoverageExecutor
          CoverageUtils
          CoverageVerify
        CodeCoveragePlugin
  build.gradle.kts
```

Custom Gradle Plugin Начало



```
code_coverage
  src
    main
      kotlin
        code_coverage
          data
            CodeCoverageConfiguration
            ModuleCoverageInfo
          CoverageBaseline
          CoverageExecutor
          CoverageUtils
          CoverageVerify
        CodeCoveragePlugin
  build.gradle.kts
```

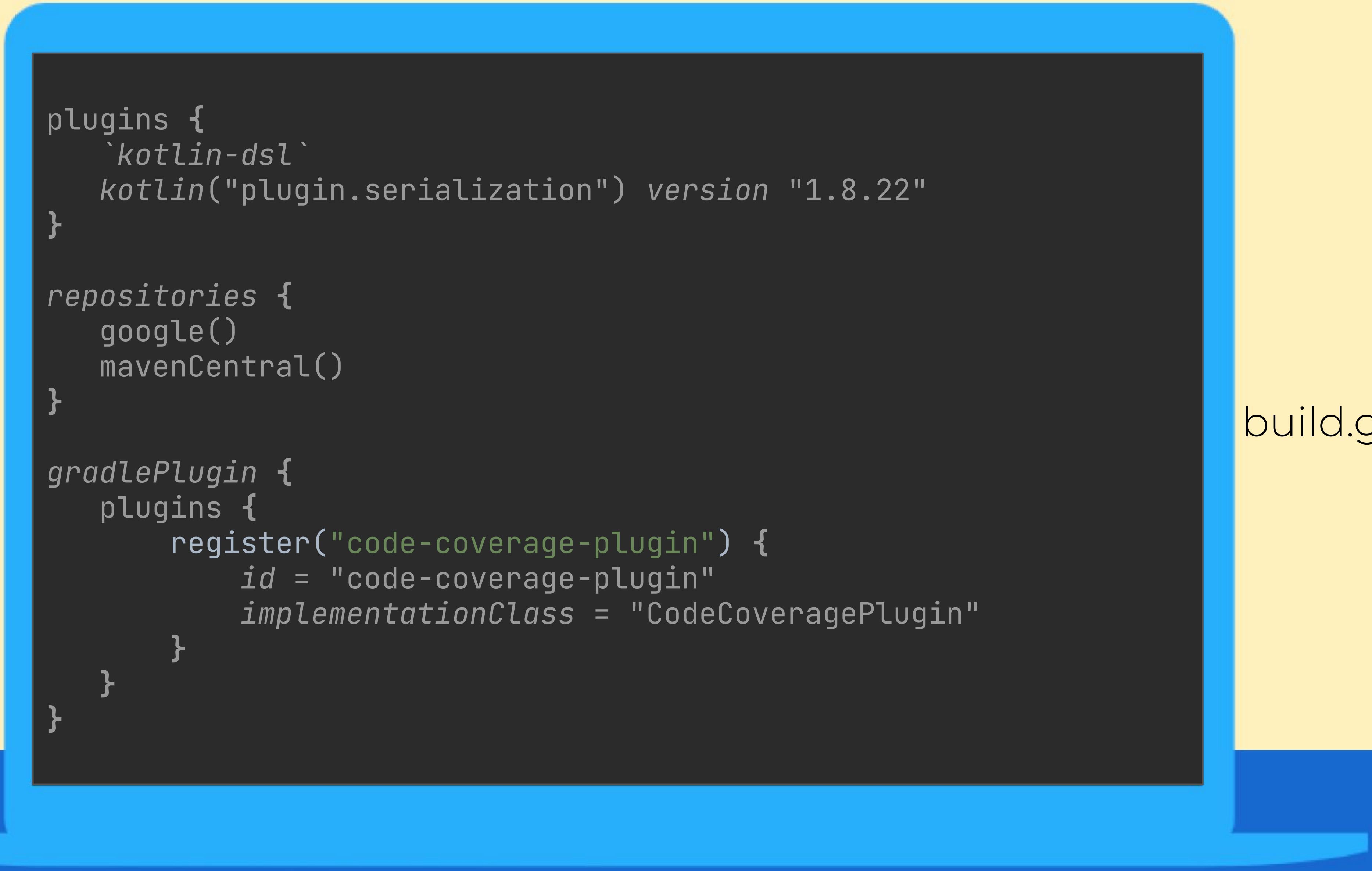
Custom Gradle Plugin Начало



```
plugins {
    `kotlin-dsl`
    kotlin("plugin.serialization") version "1.8.22"
}

repositories {
    google()
    mavenCentral()
}

gradlePlugin {
    plugins {
        register("code-coverage-plugin") {
            id = "code-coverage-plugin"
            implementationClass = "CodeCoveragePlugin"
        }
    }
}
```

build.gradle.kts

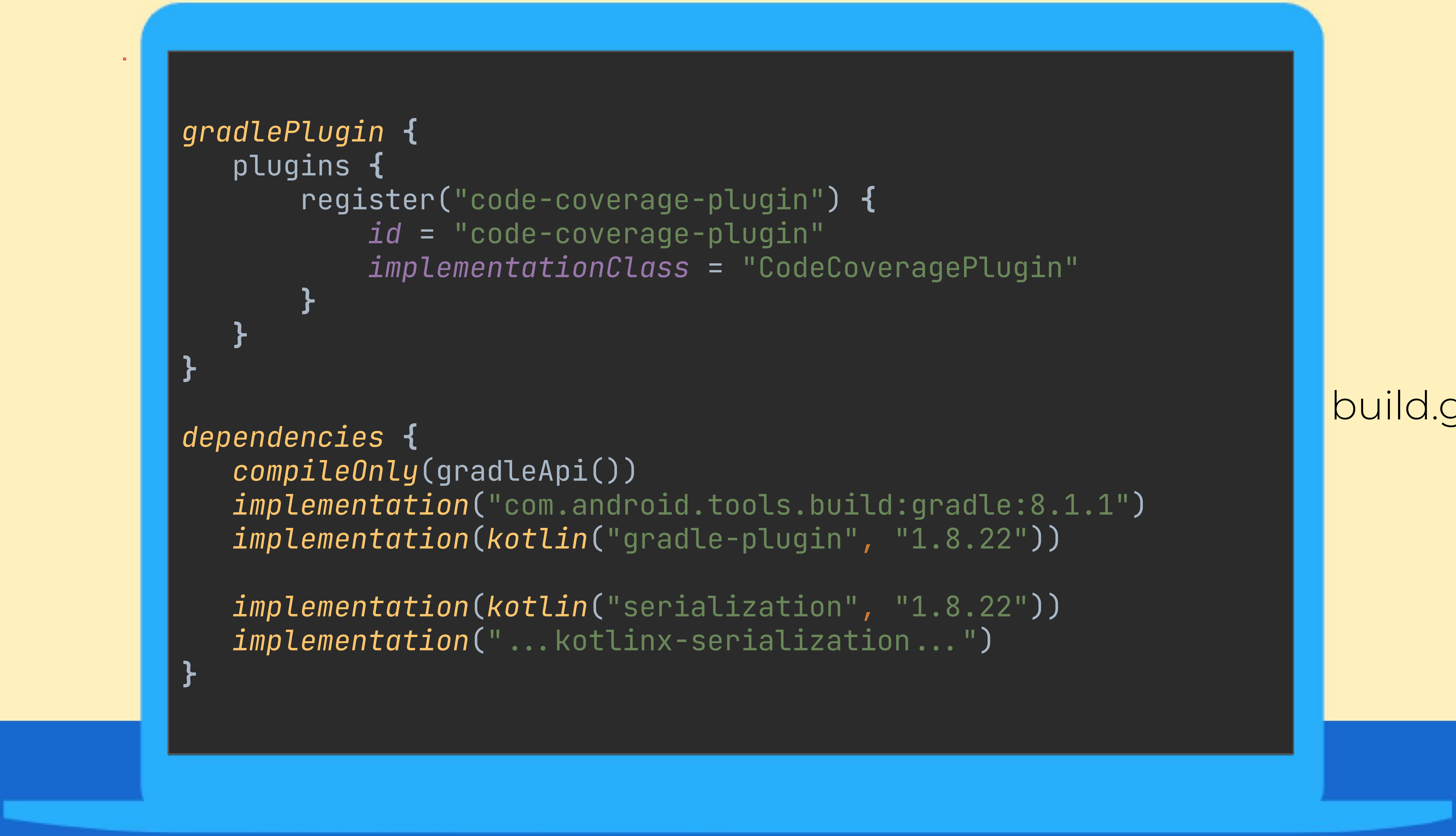
Custom Gradle Plugin Начало



```
plugins {
    `kotlin-dsl`
    kotlin("plugin.serialization") version "1.8.22"
}

repositories {
    google()
    mavenCentral()
}

gradlePlugin {
    plugins {
        register("code-coverage-plugin") {
            id = "code-coverage-plugin"
            implementationClass = "CodeCoveragePlugin"
        }
    }
}
```

build.gradle.kts

Custom Gradle Plugin Начало

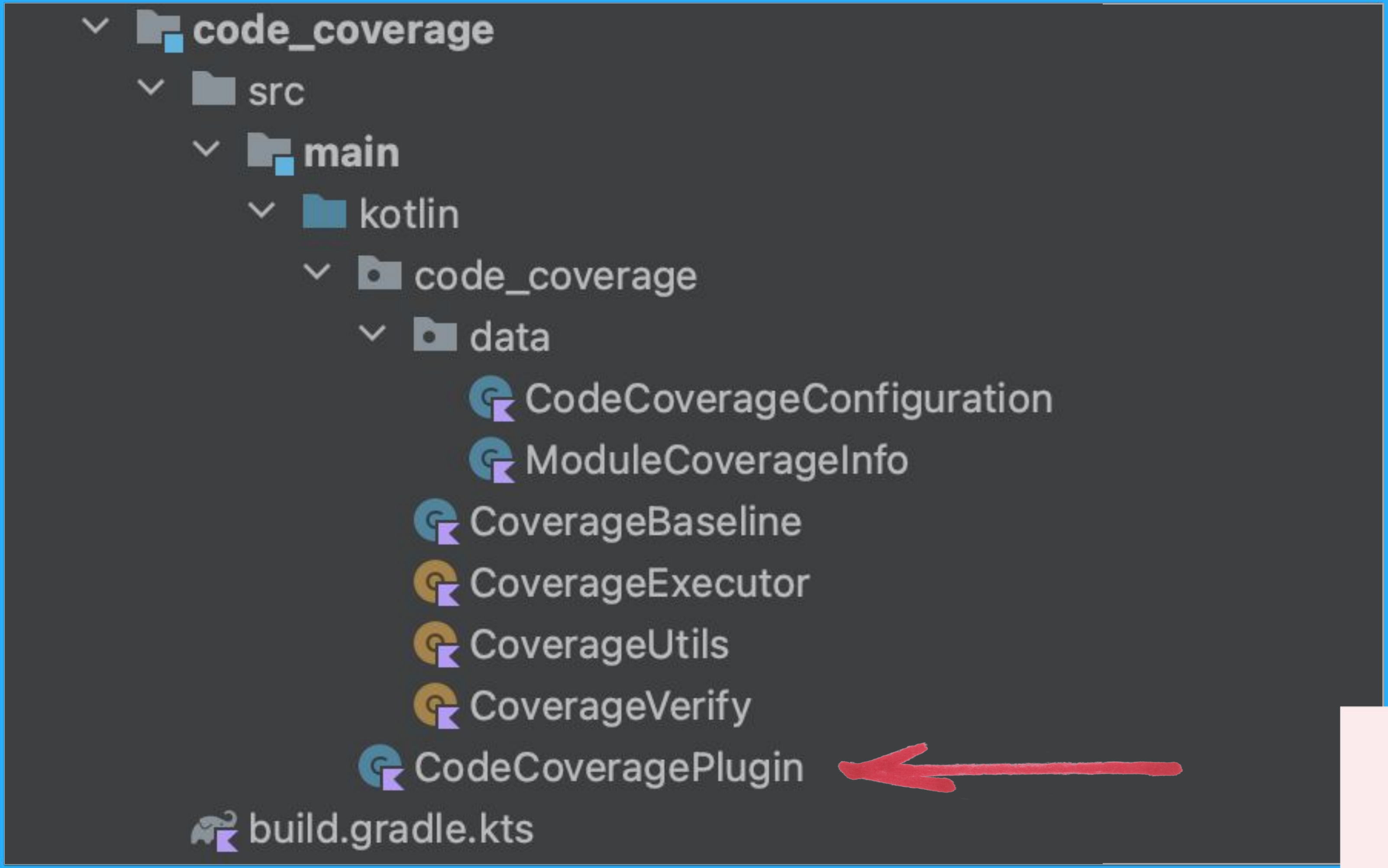


```kotlin
gradlePlugin {
    plugins {
        register("code-coverage-plugin") {
            id = "code-coverage-plugin"
            implementationClass = "CodeCoveragePlugin"
        }
    }
}

dependencies {
    compileOnly(gradleApi())
    implementation("com.android.tools.build:gradle:8.1.1")
    implementation(kotlin("gradle-plugin", "1.8.22"))

    implementation(kotlin("serialization", "1.8.22"))
    implementation("...kotlinx-serialization...")
}
```

build.gradle.kts

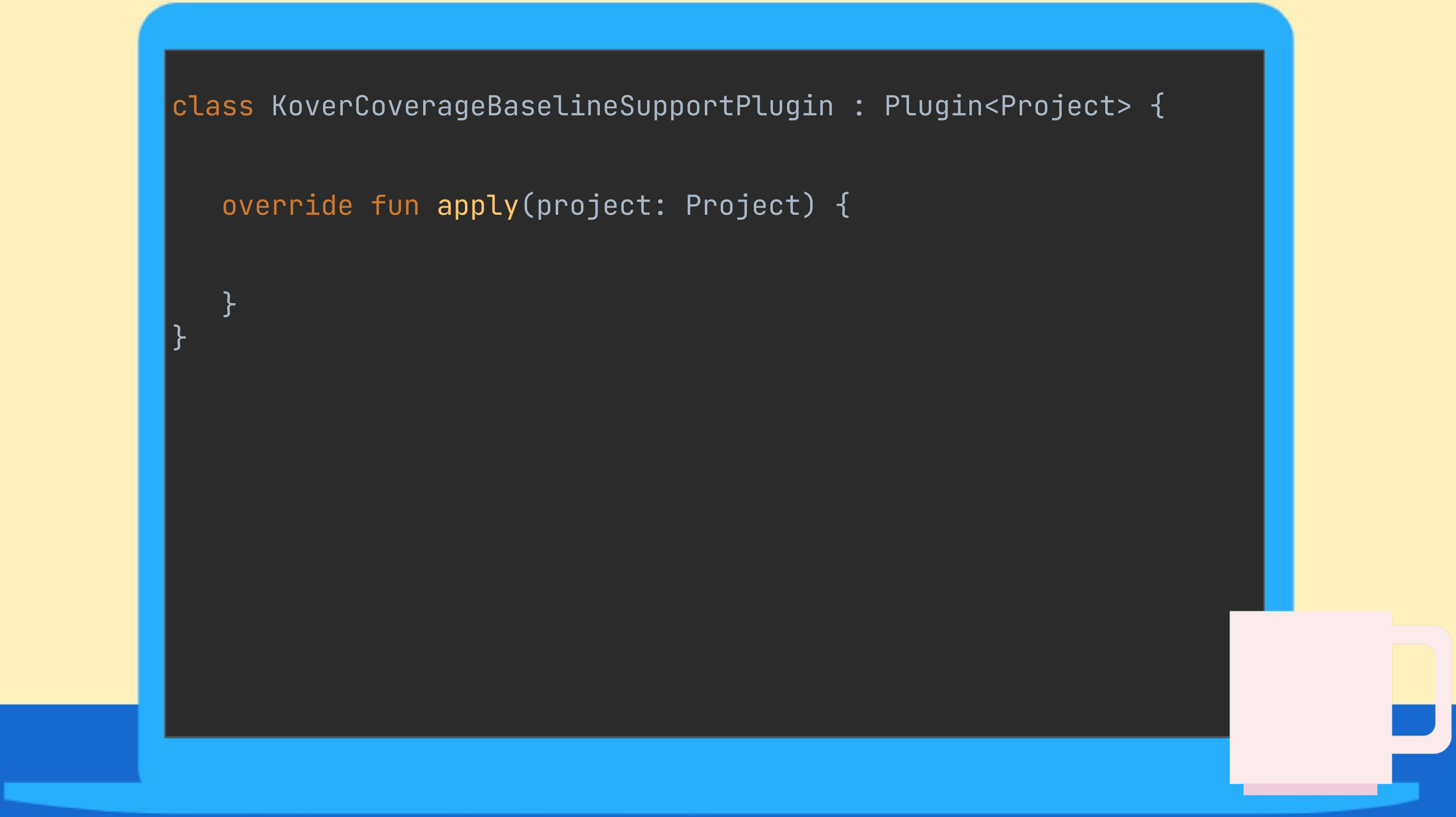
Создадим сам плагин

Custom Gradle Plugin Начало



```
code_coverage
  src
    main
      kotlin
        code_coverage
          data
            CodeCoverageConfiguration
            ModuleCoverageInfo
          CoverageBaseline
          CoverageExecutor
          CoverageUtils
          CoverageVerify
        CodeCoveragePlugin
  build.gradle.kts
```

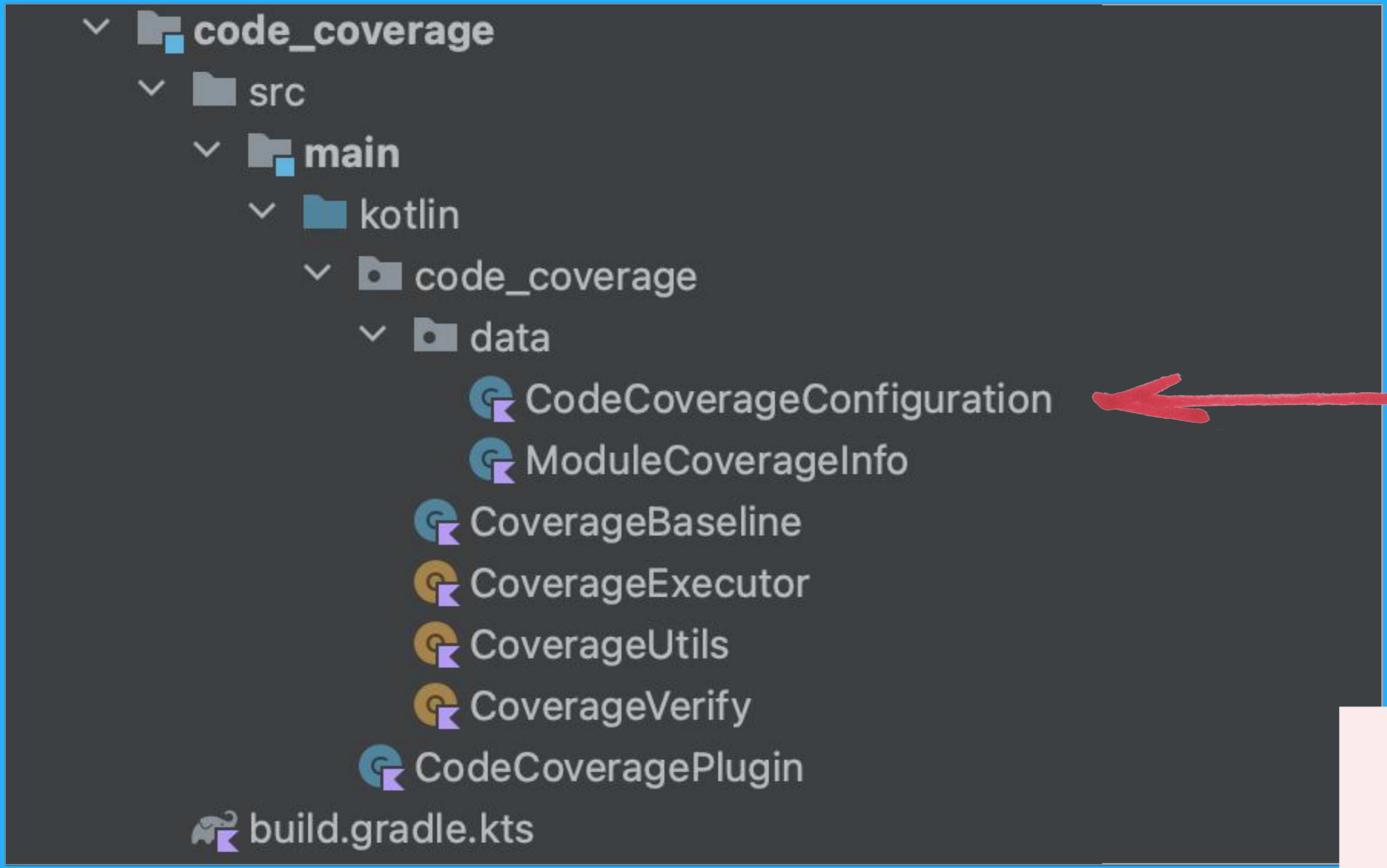
Custom Gradle Plugin



```
class KoverCoverageBaselineSupportPlugin : Plugin<Project> {

    override fun apply(project: Project) {

    }
}
```

Создадим Configuration class

Custom Gradle Plugin Начало

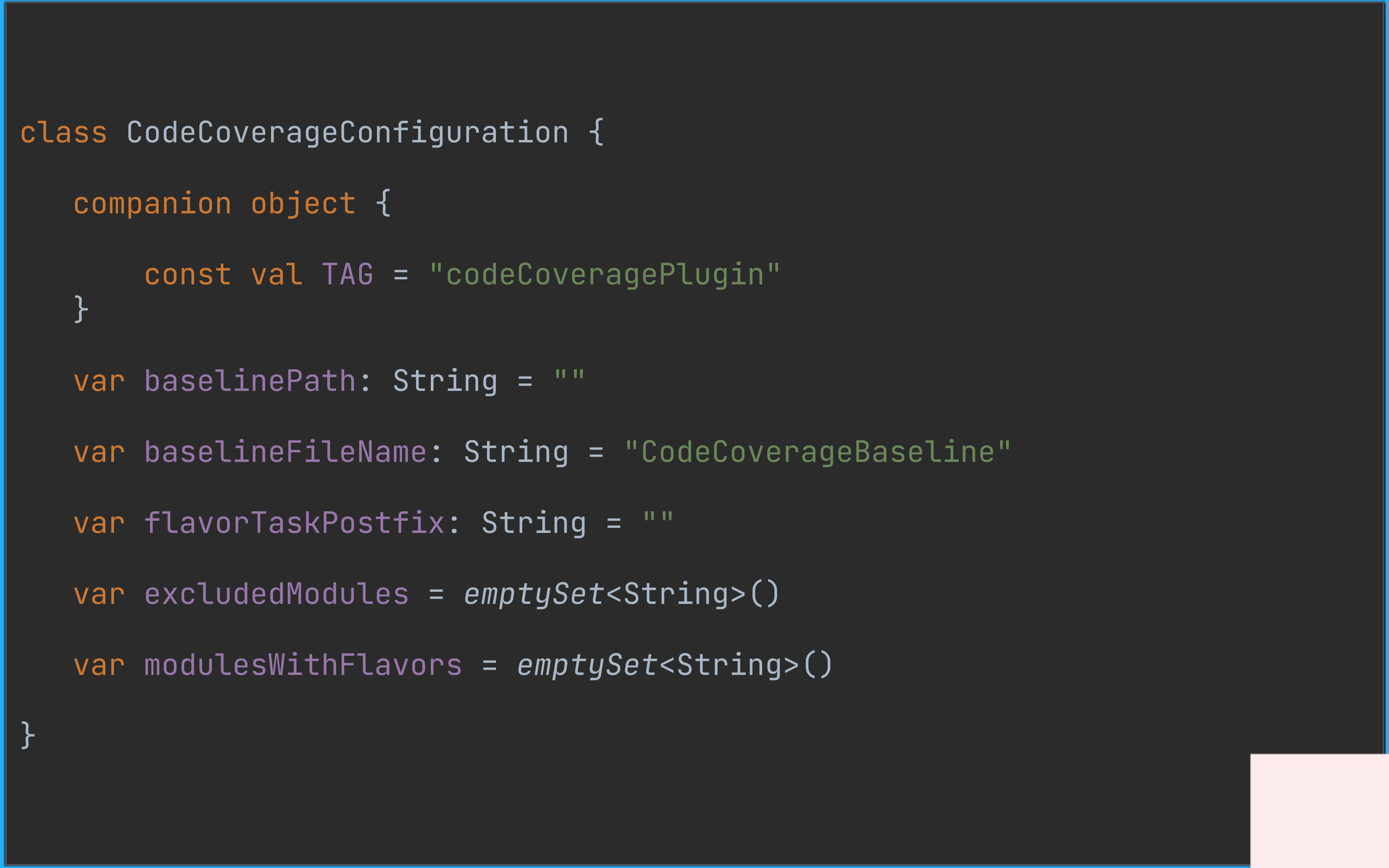


```
code_coverage
  src
    main
      kotlin
        code_coverage
          data
            CodeCoverageConfiguration
            ModuleCoverageInfo
          CoverageBaseline
          CoverageExecutor
          CoverageUtils
          CoverageVerify
        CodeCoveragePlugin
  build.gradle.kts
```

Custom Gradle Plugin - Configuration class

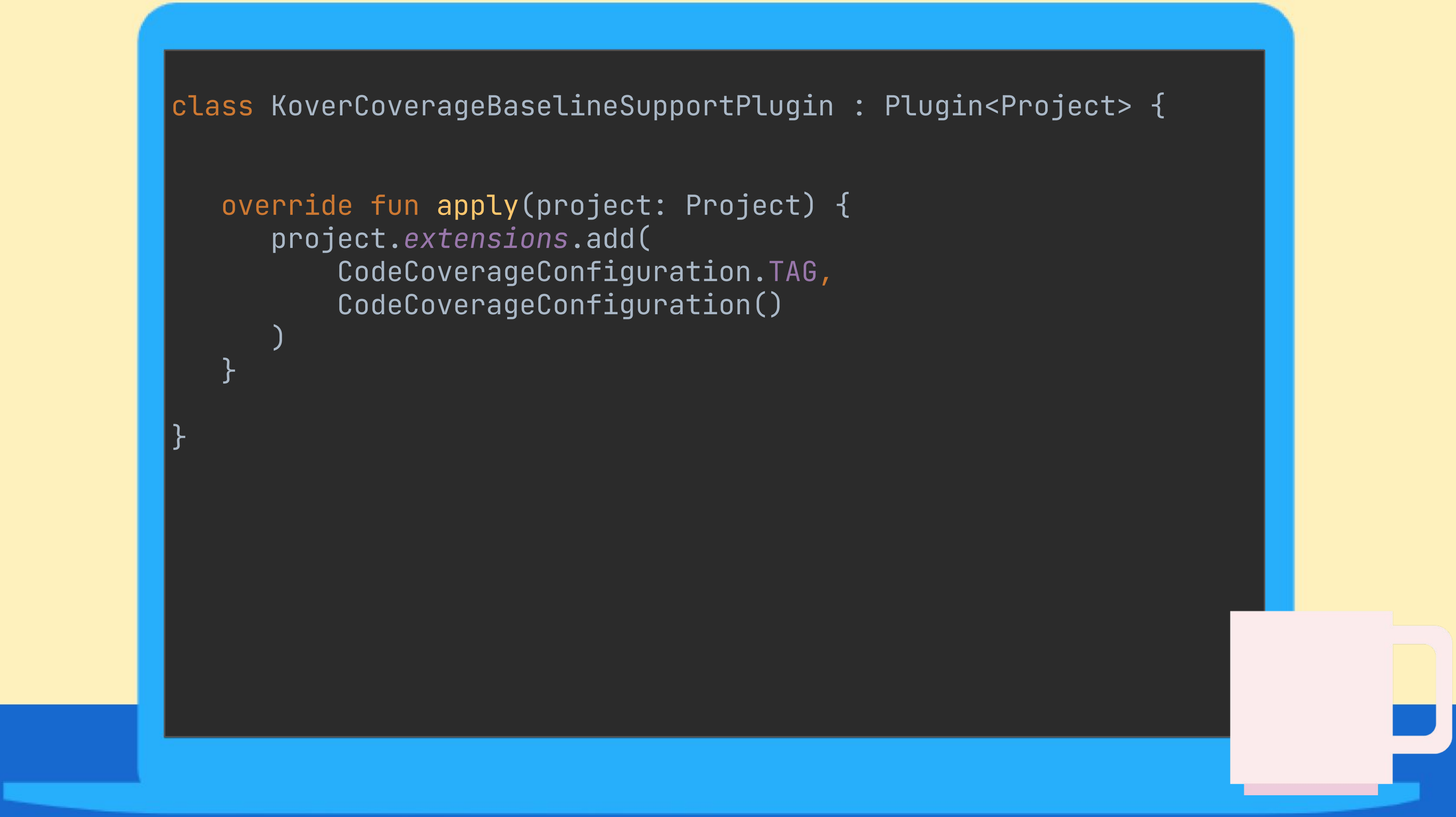
```
codeCoveragePlugin {
   excludedModules = [":design_system"]

   flavorTaskPostfix = "HuaweiDebug"
   modulesWithFlavors = [":app"]

   baselineFileName = "code_coverage_baseline"
   baselinePath = "${project.rootDir}/tools/test-coverage/report/"
}
```

Custom Gradle Plugin - Configuration class



```kotlin
class CodeCoverageConfiguration {

   companion object {

       const val TAG = "codeCoveragePlugin"
   }

   var baselinePath: String = ""

   var baselineFileName: String = "CodeCoverageBaseline"

   var flavorTaskPostfix: String = ""

   var excludedModules = emptySet<String>()

   var modulesWithFlavors = emptySet<String>()

}
```

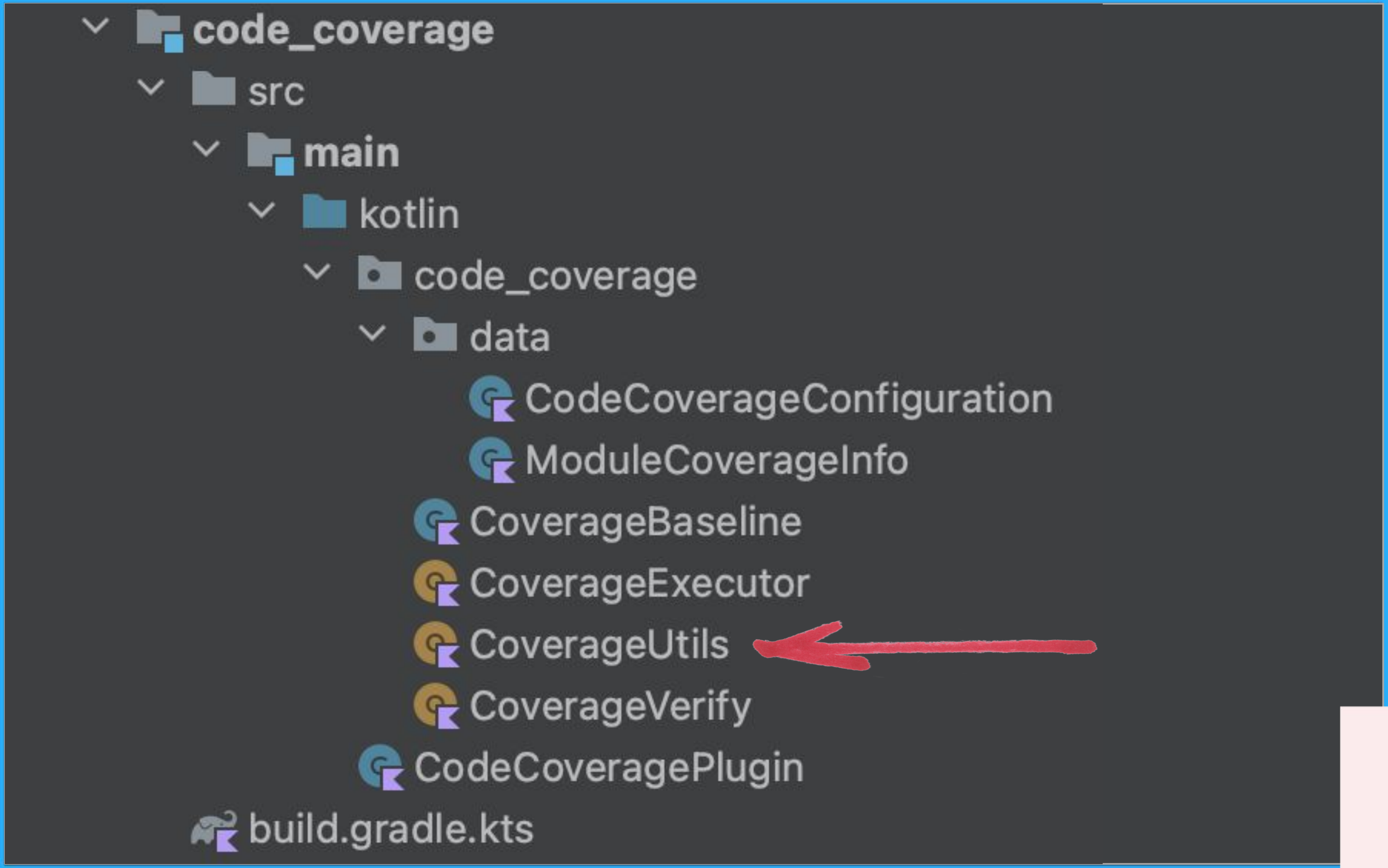
Custom Gradle Plugin



```kotlin
class KoverCoverageBaselineSupportPlugin : Plugin<Project> {

   override fun apply(project: Project) {
      project.extensions.add(
          CodeCoverageConfiguration.TAG,
          CodeCoverageConfiguration()
      )
   }

}
```

Создадим Utils file

Custom Gradle Plugin Начало

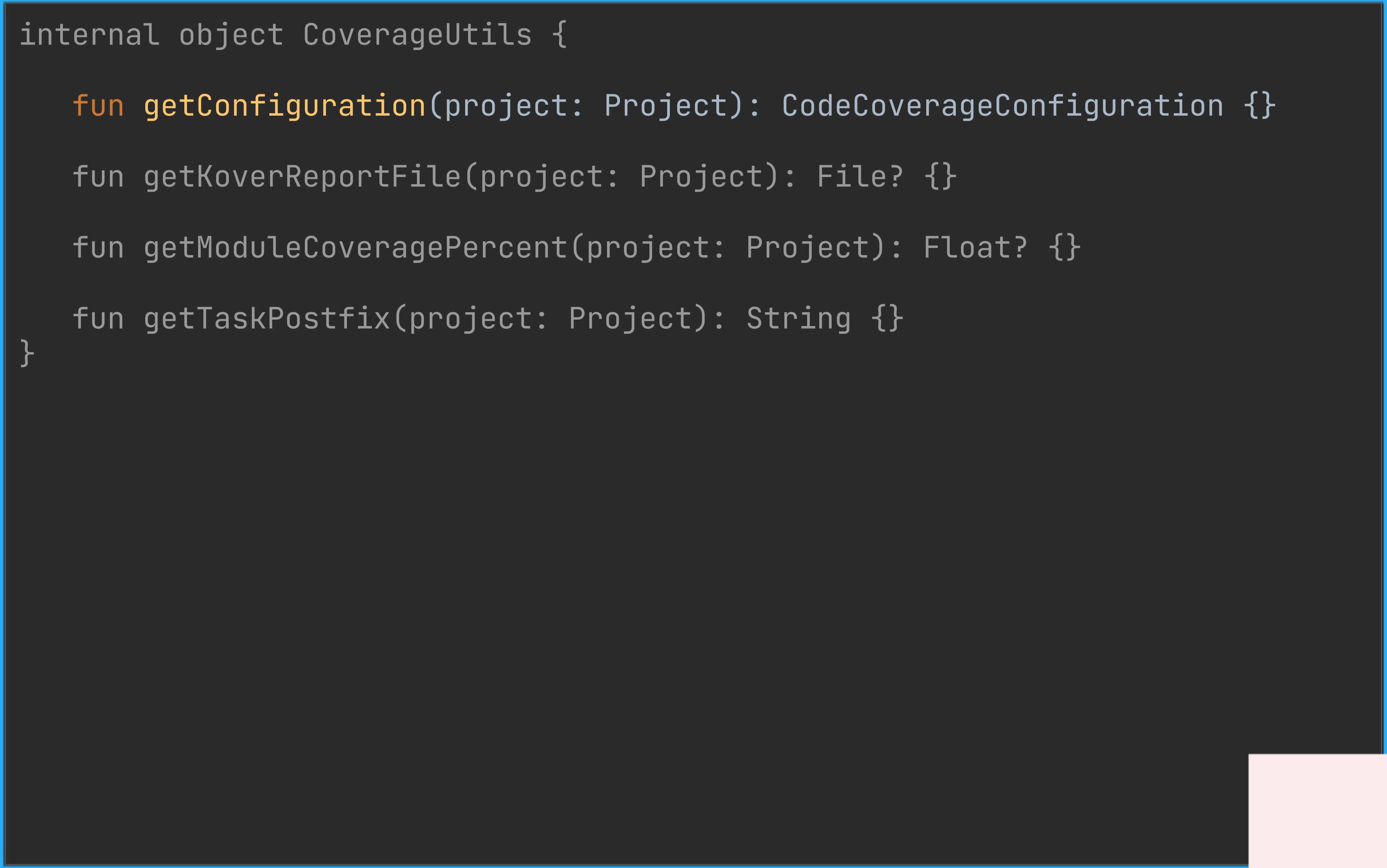


```
code_coverage
  src
    main
      kotlin
        code_coverage
          data
            CodeCoverageConfiguration
            ModuleCoverageInfo
          CoverageBaseline
          CoverageExecutor
          CoverageUtils
          CoverageVerify
        CodeCoveragePlugin
  build.gradle.kts
```

Custom Gradle Plugin - Utils

```kotlin
internal object CoverageUtils {

    fun getConfiguration(project: Project): CodeCoverageConfiguration {}

    fun getKoverReportFile(project: Project): File? {}

    fun getModuleCoveragePercent(project: Project): Float? {}

    fun getTaskPostfix(project: Project): String {}
}
```

Custom Gradle Plugin - Utils



```
internal object CoverageUtils {

    fun getConfiguration(project: Project): CodeCoverageConfiguration {}

    fun getKoverReportFile(project: Project): File? {}

    fun getModuleCoveragePercent(project: Project): Float? {}

    fun getTaskPostfix(project: Project): String {}
}
```

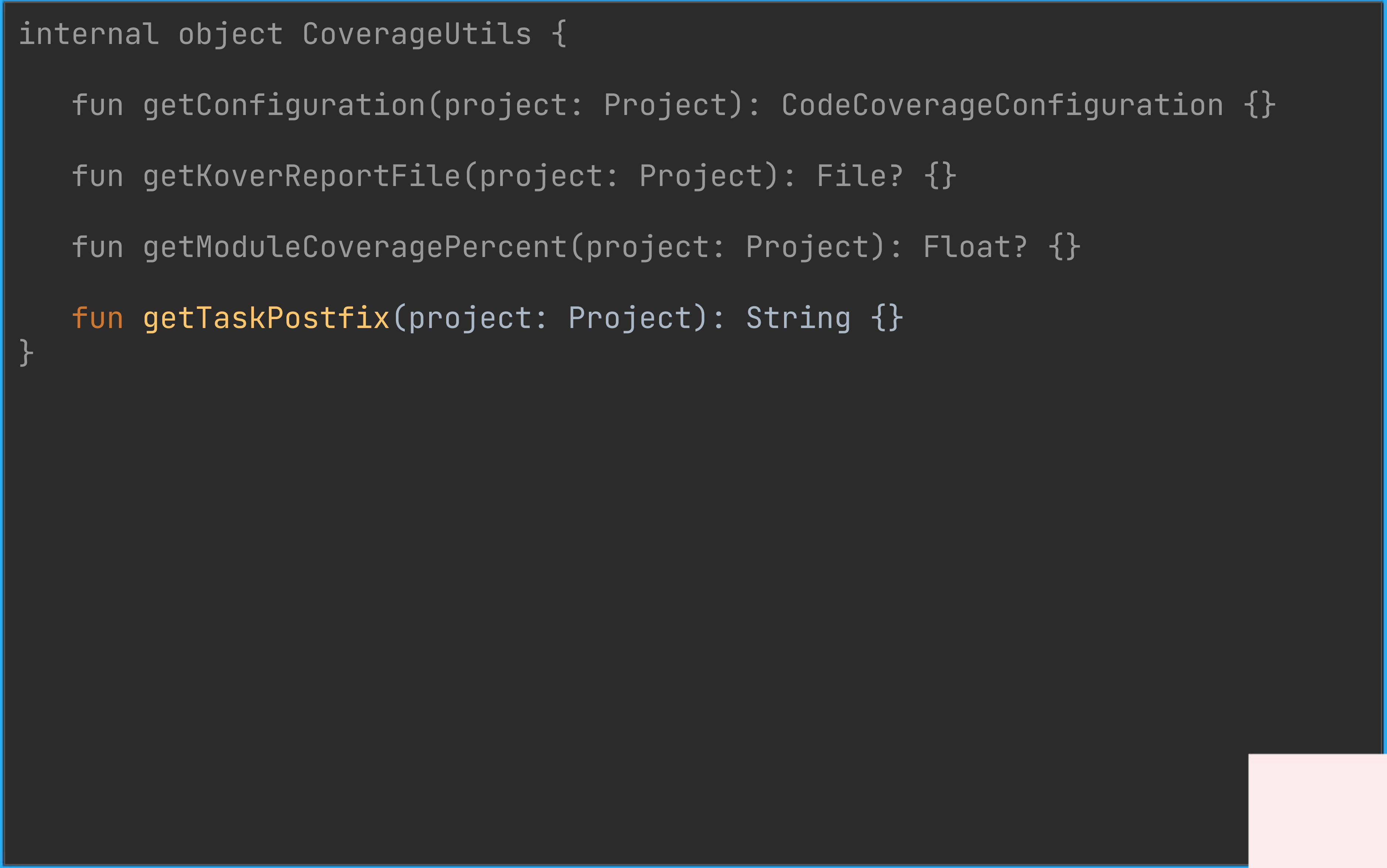
Custom Gradle Plugin - Utils

```
internal object CoverageUtils {

    fun getConfiguration(project: Project): CodeCoverageConfiguration {}

    fun getKoverReportFile(project: Project): File? {}

    fun getModuleCoveragePercent(project: Project): Float? {}

    fun getTaskPostfix(project: Project): String {}
}
```

Custom Gradle Plugin - Utils

```
internal object CoverageUtils {

    fun getConfiguration(project: Project): CodeCoverageConfiguration {}

    fun getKoverReportFile(project: Project): File? {}

    fun getModuleCoveragePercent(project: Project): Float? {}

    fun getTaskPostfix(project: Project): String {}
}
```

Custom Gradle Plugin - Utils

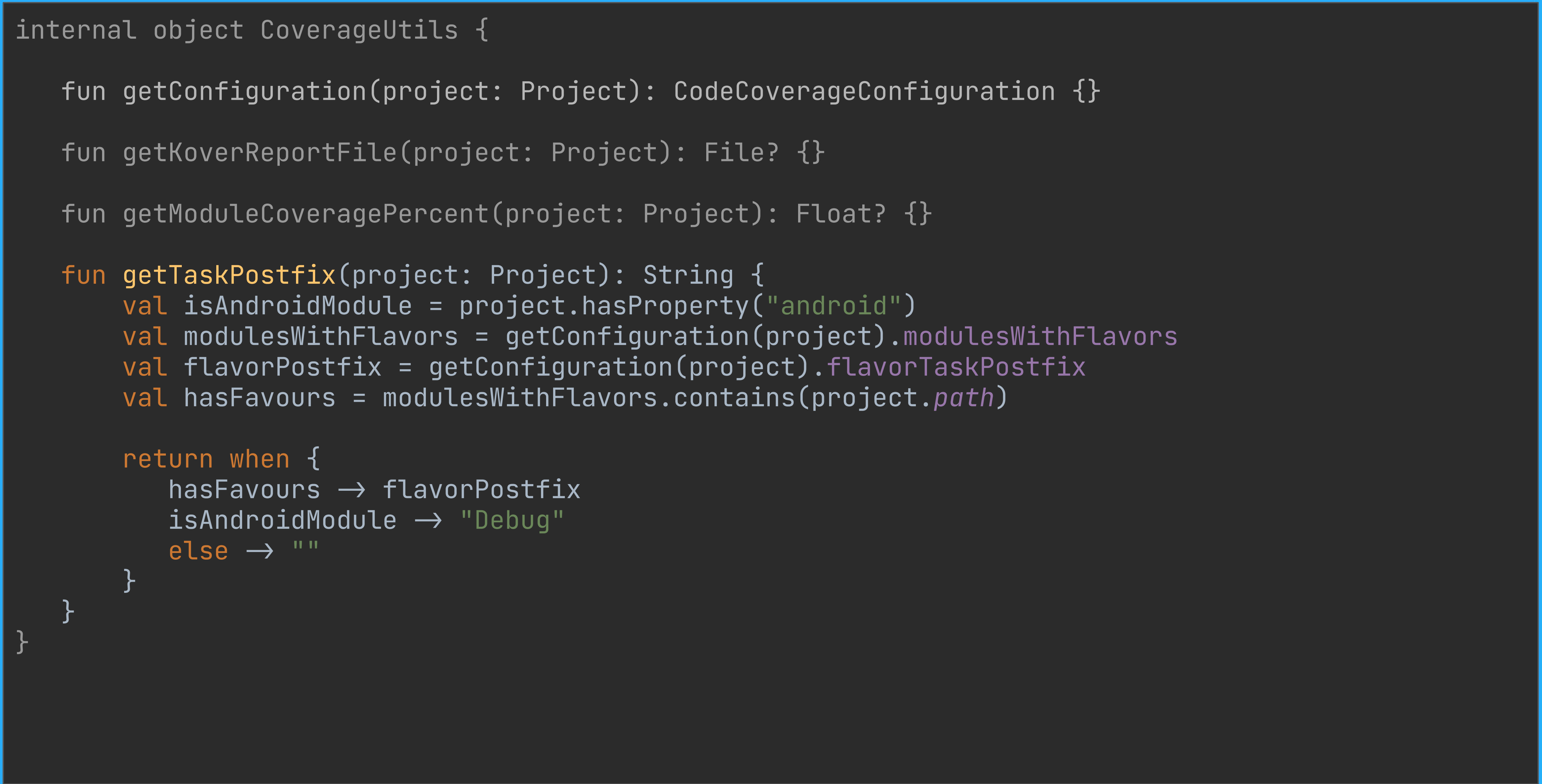


```
internal object CoverageUtils {

    fun getConfiguration(project: Project): CodeCoverageConfiguration {}

    fun getKoverReportFile(project: Project): File? {}

    fun getModuleCoveragePercent(project: Project): Float? {}

    fun getTaskPostfix(project: Project): String {}
}
```

Custom Gradle Plugin - verify task

```kotlin
internal object CoverageUtils {

   fun getConfiguration(project: Project): CodeCoverageConfiguration {
       return requireNotNull(
           value = project.rootProject.extensions.findByName(CodeCoverageConfiguration.TAG),
           lazyMessage = {
               "Unable to find ${CodeCoverageConfiguration.TAG} in ${project.rootProject}"
           }
       ) as CodeCoverageConfiguration
   }


   fun getKoverReportFile(project: Project): File? {}

   fun getModuleCoveragePercent(project: Project): Float? {}

   fun getTaskPostfix(project: Project): String {}
}
```

Custom Gradle Plugin - verify task

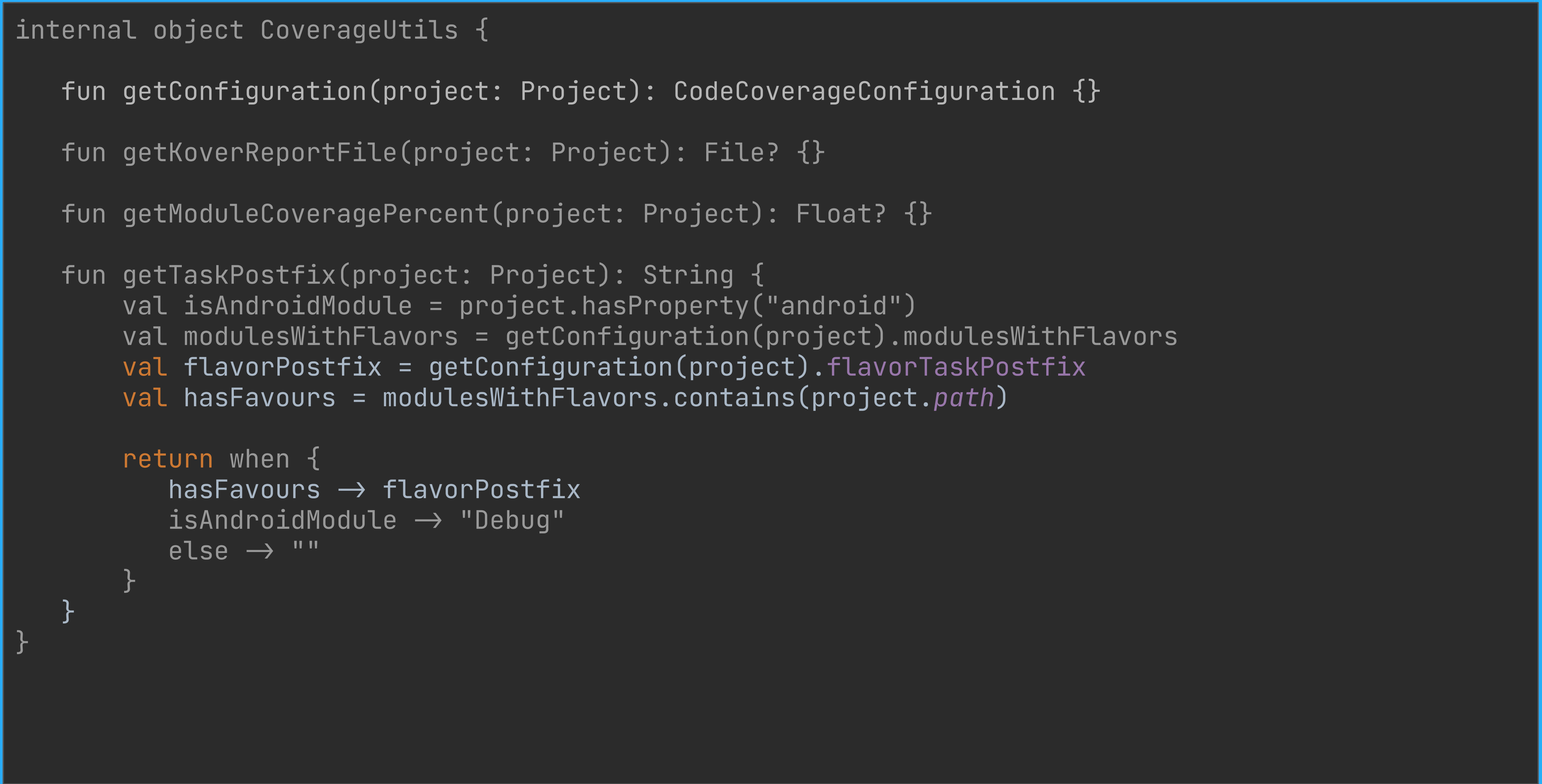


```kotlin
internal object CoverageUtils {

   fun getConfiguration(project: Project): CodeCoverageConfiguration {}

   fun getKoverReportFile(project: Project): File? {}

   fun getModuleCoveragePercent(project: Project): Float? {}

   fun getTaskPostfix(project: Project): String {
       val isAndroidModule = project.hasProperty("android")
       val modulesWithFlavors = getConfiguration(project).modulesWithFlavors
       val flavorPostfix = getConfiguration(project).flavorTaskPostfix
       val hasFavours = modulesWithFlavors.contains(project.path)

       return when {
          hasFavours -> flavorPostfix
          isAndroidModule -> "Debug"
          else -> ""
       }
   }
}
```

Custom Gradle Plugin - verify task

```kotlin
internal object CoverageUtils {

   fun getConfiguration(project: Project): CodeCoverageConfiguration {}

   fun getKoverReportFile(project: Project): File? {}

   fun getModuleCoveragePercent(project: Project): Float? {}

   fun getTaskPostfix(project: Project): String {
       val isAndroidModule = project.hasProperty("android")
       val modulesWithFlavors = getConfiguration(project).modulesWithFlavors
       val flavorPostfix = getConfiguration(project).flavorTaskPostfix
       val hasFavours = modulesWithFlavors.contains(project.path)

       return when {
          hasFavours -> flavorPostfix
          isAndroidModule -> "Debug"
          else -> ""
       }
   }
}
```

Custom Gradle Plugin - verify task



```kotlin
internal object CoverageUtils {

   fun getConfiguration(project: Project): CodeCoverageConfiguration {}

   fun getKoverReportFile(project: Project): File? {}

   fun getModuleCoveragePercent(project: Project): Float? {}

   fun getTaskPostfix(project: Project): String {
       val isAndroidModule = project.hasProperty("android")
       val modulesWithFlavors = getConfiguration(project).modulesWithFlavors
       val flavorPostfix = getConfiguration(project).flavorTaskPostfix
       val hasFavours = modulesWithFlavors.contains(project.path)

       return when {
          hasFavours → flavorPostfix
          isAndroidModule → "Debug"
          else → ""
       }
   }
}
```

Custom Gradle Plugin - verify task

```kotlin
internal object CoverageUtils {

   fun getConfiguration(project: Project): CodeCoverageConfiguration {}

   fun getKoverReportFile(project: Project): File? {}

   fun getModuleCoveragePercent(project: Project): Float? {}

   fun getTaskPostfix(project: Project): String {
       val isAndroidModule = project.hasProperty("android")
       val modulesWithFlavors = getConfiguration(project).modulesWithFlavors
       val flavorPostfix = getConfiguration(project).flavorTaskPostfix
       val hasFavours = modulesWithFlavors.contains(project.path)

       return when {
          hasFavours -> flavorPostfix
          isAndroidModule -> "Debug"
          else -> ""
       }
   }
}
```

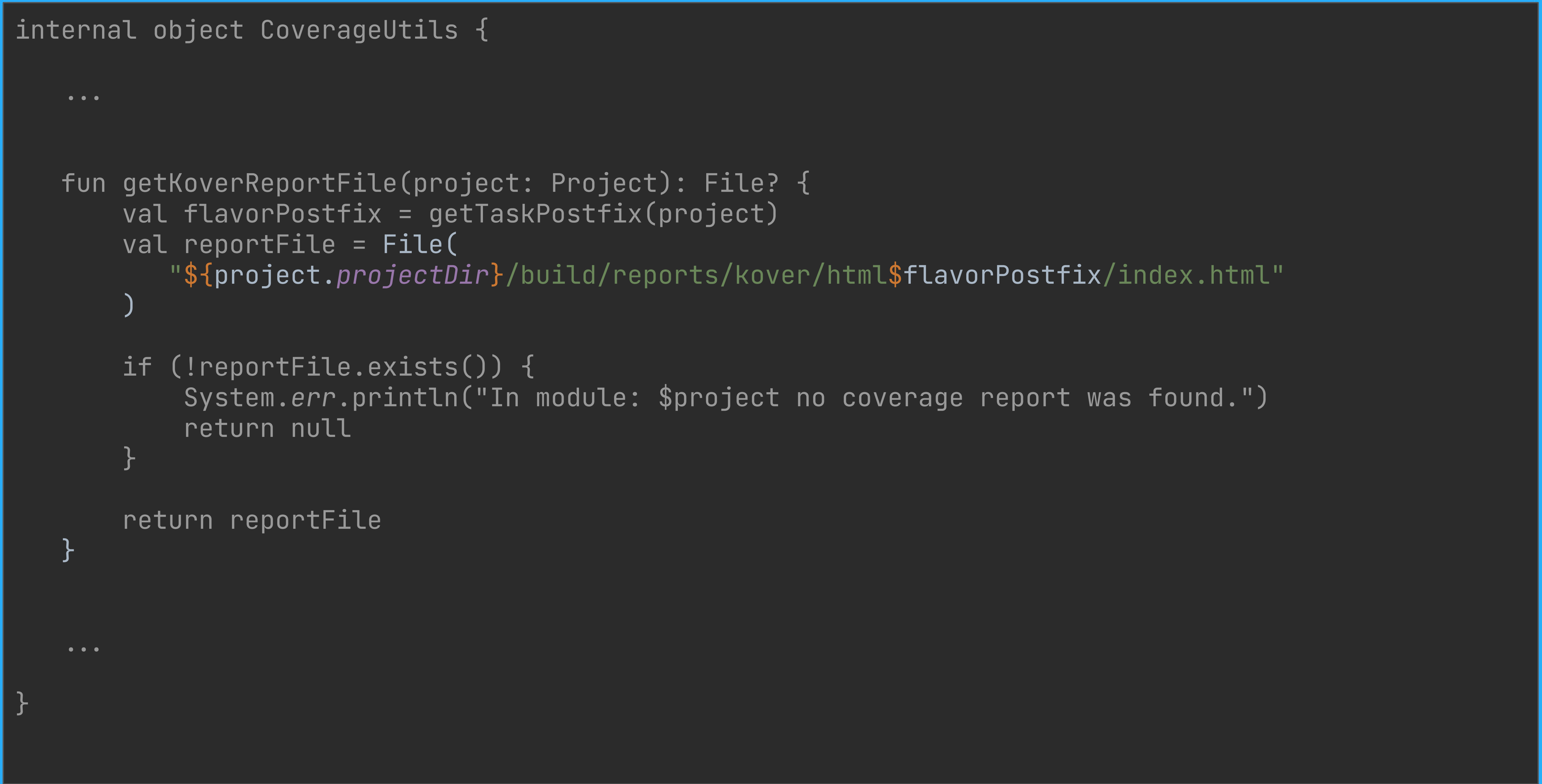
Custom Gradle Plugin - verify task

```kotlin
internal object CoverageUtils {

    ...

    fun getModuleCoveragePercent(project: Project): Float? {
        val reportFile = getKoverReportFile(project) ?: return null

        return reportFile
            .readText()
            .substringAfter("<span class=\"percent\">")
            .substringBefore("%")
            .trim()
            .toFloatOrNull() ?: 100f
    }

    ...

}
```

Custom Gradle Plugin - verify task

```kotlin
internal object CoverageUtils {

   ...


   fun getKoverReportFile(project: Project): File? {
       val flavorPostfix = getTaskPostfix(project)
       val reportFile = File(
          "${project.projectDir}/build/reports/kover/html$flavorPostfix/index.html"
       )

       if (!reportFile.exists()) {
           System.err.println("In module: $project no coverage report was found.")
           return null
       }

       return reportFile
   }


   ...

}
```

Custom Gradle Plugin - verify task



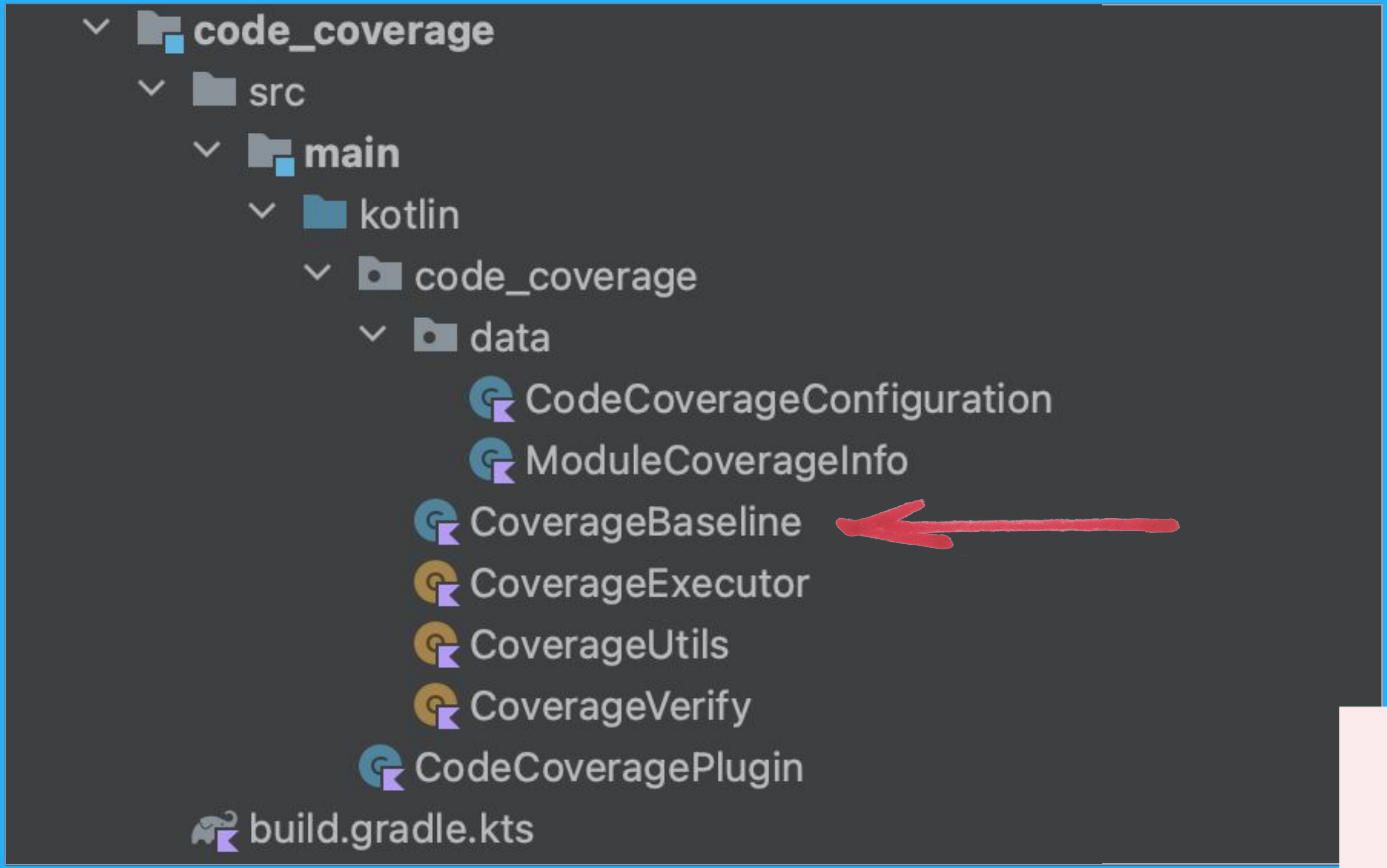
```kotlin
internal object CoverageUtils {

   ...


   fun getKoverReportFile(project: Project): File? {
       val flavorPostfix = getTaskPostfix(project)
       val reportFile = File(
          "${project.projectDir}/build/reports/kover/html$flavorPostfix/index.html"
       )

       if (!reportFile.exists()) {
           System.err.println("In module: $project no coverage report was found.")
           return null
       }

       return reportFile
   }


   ...

}
```

Custom Gradle Plugin - verify task

```kotlin
internal object CoverageUtils {

    ...


    fun getKoverReportFile(project: Project): File? {
        val flavorPostfix = getTaskPostfix(project)
        val reportFile = File(
           "${project.projectDir}/build/reports/kover/html$flavorPostfix/index.html"
        )

        if (!reportFile.exists()) {
            System.err.println("In module: $project no coverage report was found.")
            return null
        }

        return reportFile
    }



    ...

}
```

Создадим Baseline task

Custom Gradle Plugin Начало



```
code_coverage
  src
    main
      kotlin
        code_coverage
          data
            CodeCoverageConfiguration
            ModuleCoverageInfo
          CoverageBaseline
          CoverageExecutor
          CoverageUtils
          CoverageVerify
        CodeCoveragePlugin
  build.gradle.kts
```

Custom Gradle Plugin - Baseline task

```kotlin
class CoverageBaseline {

    fun clearBaseline(project: Project) {}

    fun update(project: Project) {}

    private fun getOrCreateBaselineFile(project: Project): File {}
}
```

Custom Gradle Plugin - Baseline task

```kotlin
class CoverageBaseline {

   private val coverageInfoList = mutableListOf<ModuleCoverageInfo>()

   fun clearBaseline(project: Project) {
       val baselineFile = getOrCreateBaselineFile(project)
       baselineFile.writeText("")
   }

 ...
```

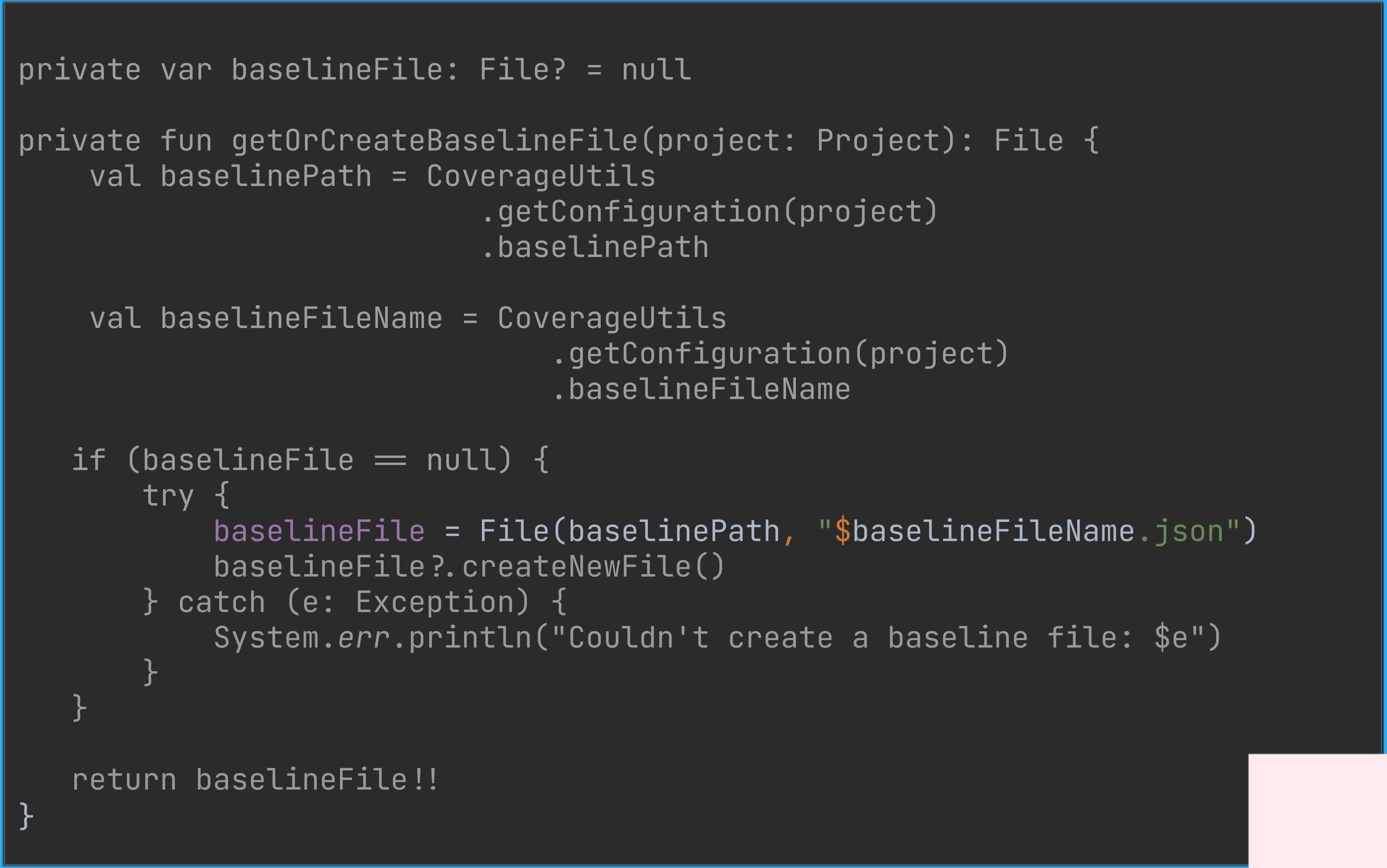
Custom Gradle Plugin - Baseline task

```kotlin
private var baselineFile: File? = null

private fun getOrCreateBaselineFile(project: Project): File {
    val baselinePath = CoverageUtils
                            .getConfiguration(project)
                            .baselinePath

    val baselineFileName = CoverageUtils
                                .getConfiguration(project)
                                .baselineFileName

   if (baselineFile == null) {
       try {
           baselineFile = File(baselinePath, "$baselineFileName.json")
           baselineFile?.createNewFile()
       } catch (e: Exception) {
           System.err.println("Couldn't create a baseline file: $e")
       }
   }

   return baselineFile!!
}
```

Custom Gradle Plugin - Baseline task

```kotlin
private var baselineFile: File? = null

private fun getOrCreateBaselineFile(project: Project): File {
    val baselinePath = CoverageUtils
                           .getConfiguration(project)
                           .baselinePath

    val baselineFileName = CoverageUtils
                               .getConfiguration(project)
                               .baselineFileName

   if (baselineFile == null) {
       try {
           baselineFile = File(baselinePath, "$baselineFileName.json")
           baselineFile?.createNewFile()
       } catch (e: Exception) {
           System.err.println("Couldn't create a baseline file: $e")
       }
   }

   return baselineFile!!
}
```

Custom Gradle Plugin - Baseline task



```
private var baselineFile: File? = null

private fun getOrCreateBaselineFile(project: Project): File {
    val baselinePath = CoverageUtils
                          .getConfiguration(project)
                          .baselinePath

    val baselineFileName = CoverageUtils
                              .getConfiguration(project)
                              .baselineFileName

   if (baselineFile == null) {
       try {
           baselineFile = File(baselinePath, "$baselineFileName.json")
           baselineFile?.createNewFile()
       } catch (e: Exception) {
           System.err.println("Couldn't create a baseline file: $e")
       }
   }

   return baselineFile!!
}
```

Custom Gradle Plugin - verify task

```kotlin
fun update(project: Project) {
   val isRoot = project.path.isBlank()
   if (isRoot) return

   // prepare baseline lines
   project.subprojects {
       val excludedModules = CoverageUtils.getConfiguration(project).excludedModules

       if (!excludedModules.contains(this.path)) {
           val coverageInfo = ModuleCoverageInfo(
               moduleName = this.path,
               percentage = CoverageUtils.getModuleCoveragePercent(this) ?: 0f
           )
           coverageInfoList.add(coverageInfo)
       }
   }

   val baselineFile = getOrCreateBaselineFile(project)
   val json = Json {
       prettyPrint = true
   }

   baselineFile.writeText(json.encodeToString(coverageInfoList))
}
```

Custom Gradle Plugin - verify task

```kotlin
fun update(project: Project) {
   val isRoot = project.path.isBlank()
   if (isRoot) return

   // prepare baseline lines
   project.subprojects {
       val excludedModules = CoverageUtils.getConfiguration(project).excludedModules

       if (!excludedModules.contains(this.path)) {
           val coverageInfo = ModuleCoverageInfo(
               moduleName = this.path,
               percentage = CoverageUtils.getModuleCoveragePercent(this) ?: 0f
           )
           coverageInfoList.add(coverageInfo)
       }
   }

   val baselineFile = getOrCreateBaselineFile(project)
   val json = Json {
       prettyPrint = true
   }

   baselineFile.writeText(json.encodeToString(coverageInfoList))
}
```

Custom Gradle Plugin - verify task

```
fun update(project: Project) {
   val isRoot = project.path.isBlank()
   if (isRoot) return

   // prepare baseline lines
   project.subprojects {
       val excludedModules = CoverageUtils.getConfiguration(project).excludedModules

       if (!excludedModules.contains(this.path)) {
           val coverageInfo = ModuleCoverageInfo(
               moduleName = this.path,
               percentage = CoverageUtils.getModuleCoveragePercent(this) ?: 0f
           )
           coverageInfoList.add(coverageInfo)
       }
   }

   val baselineFile = getOrCreateBaselineFile(project)
   val json = Json {
       prettyPrint = true
   }

   baselineFile.writeText(json.encodeToString(coverageInfoList))
}
```

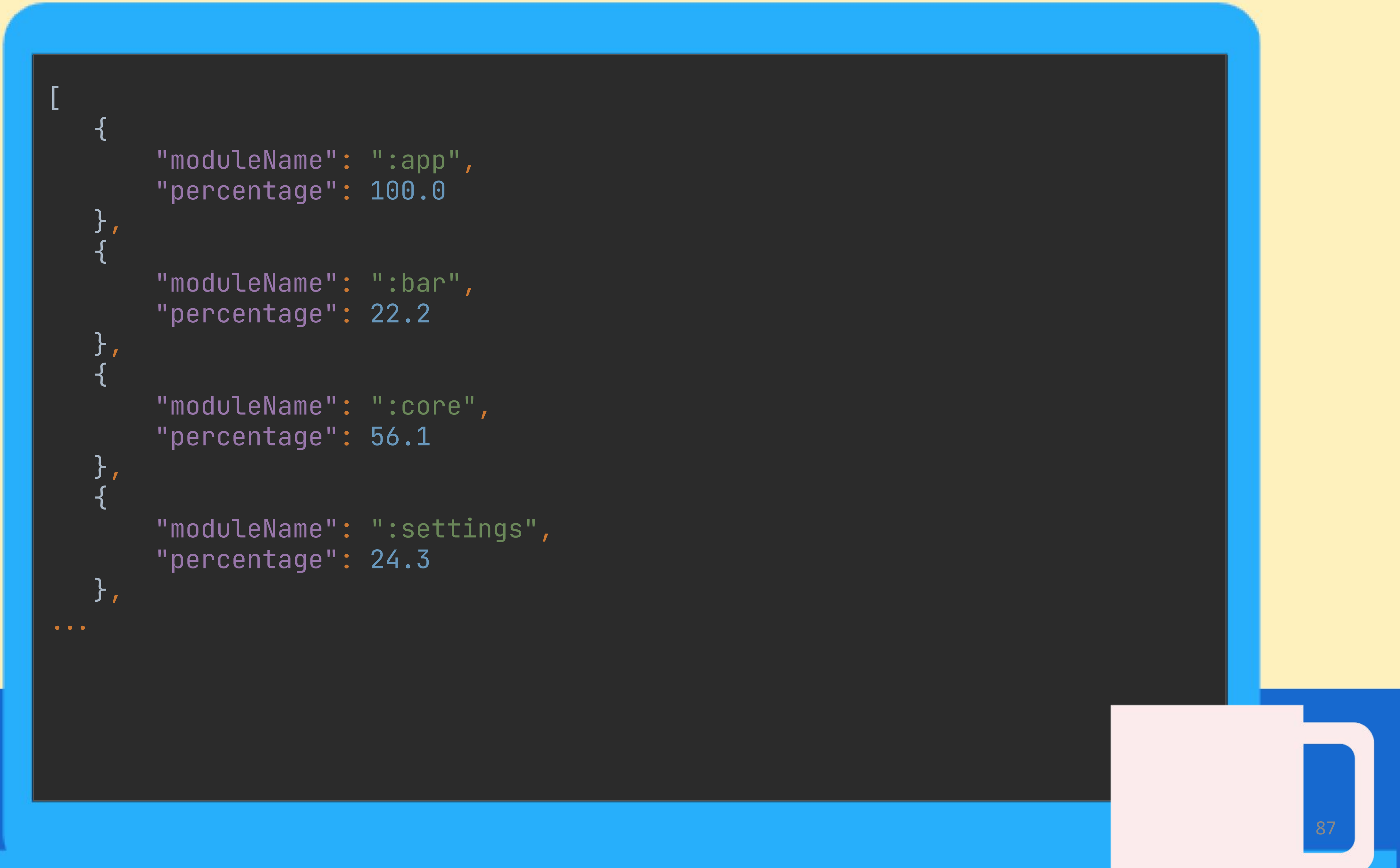
Custom Gradle Plugin - verify task

```
fun update(project: Project) {
   val isRoot = project.path.isBlank()
   if (isRoot) return

   // prepare baseline lines
   project.subprojects {
       val excludedModules = CoverageUtils.getConfiguration(project).excludedModules

       if (!excludedModules.contains(this.path)) {
           val coverageInfo = ModuleCoverageInfo(
               moduleName = this.path,
               percentage = CoverageUtils.getModuleCoveragePercent(this) ?: 0f
           )
           coverageInfoList.add(coverageInfo)
       }
   }

   val baselineFile = getOrCreateBaselineFile(project)
   val json = Json {
       prettyPrint = true
   }

   baselineFile.writeText(json.encodeToString(coverageInfoList))
}
```

Custom Gradle Plugin - verify task

```kotlin
fun update(project: Project) {
   val isRoot = project.path.isBlank()
   if (isRoot) return

   // prepare baseline lines
   project.subprojects {
       val excludedModules = CoverageUtils.getConfiguration(project).excludedModules

       if (!excludedModules.contains(this.path)) {
           val coverageInfo = ModuleCoverageInfo(
               moduleName = this.path,
               percentage = CoverageUtils.getModuleCoveragePercent(this) ?: 0f
           )
           coverageInfoList.add(coverageInfo)
       }
   }

   val baselineFile = getOrCreateBaselineFile(project)
   val json = Json {
       prettyPrint = true
   }

   baselineFile.writeText(json.encodeToString(coverageInfoList))
}
```

Custom Gradle Plugin - Baseline task

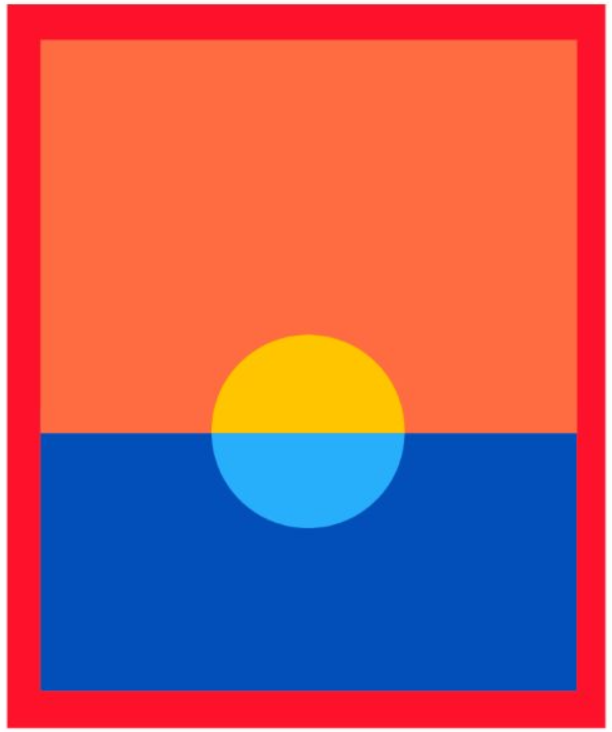


```
[
    {
        "moduleName": ":app",
        "percentage": 100.0
    },
    {
        "moduleName": ":bar",
        "percentage": 22.2
    },
    {
        "moduleName": ":core",
        "percentage": 56.1
    },
    {
        "moduleName": ":settings",
        "percentage": 24.3
    },
...
```

Требования к custom gradle plugin

Какие команды нам нужны?

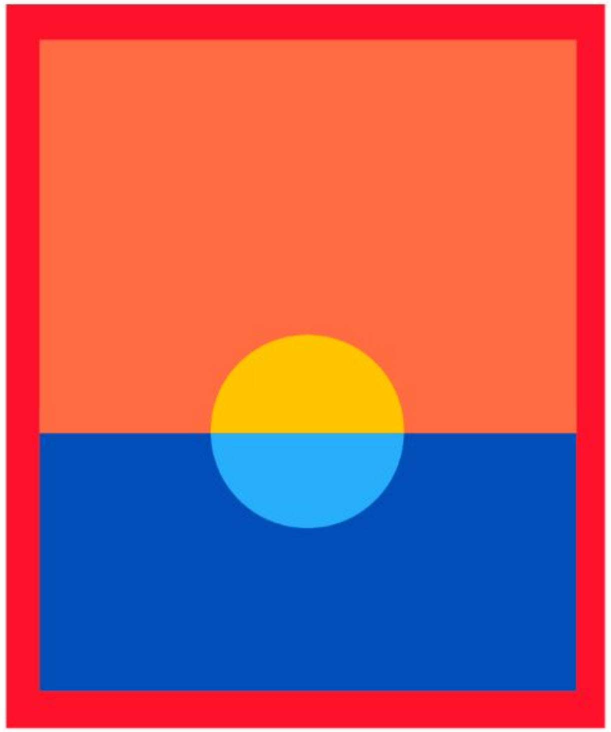
- 🔵 Команда создания baseline ✅
- 🔵 Команда создания репорта
- 🔵 Команда проверки покрытия



Требования к custom gradle plugin

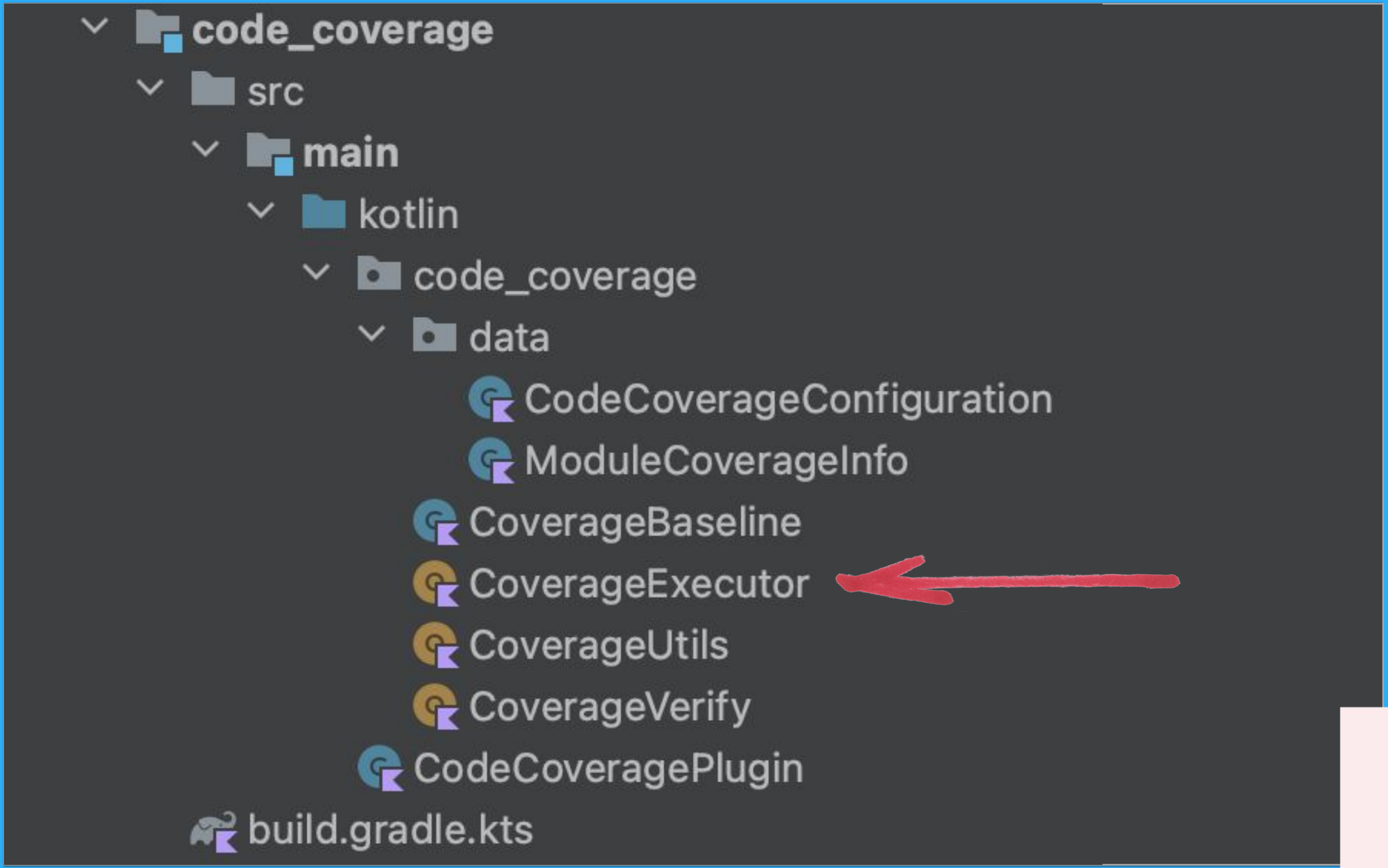
Специфичные требования?

- Baseline должен быть удобочитаемым ✅
- Плагин выкидывает исключение если покрытие уменьшилось
- Команды запускают kover вне зависимости от типа модуля и flavor



Создадим Generate kover report task

Custom Gradle Plugin Начало



```
code_coverage
  src
    main
      kotlin
        code_coverage
          data
            CodeCoverageConfiguration
            ModuleCoverageInfo
          CoverageBaseline
          CoverageExecutor
          CoverageUtils
          CoverageVerify
        CodeCoveragePlugin
  build.gradle.kts
```

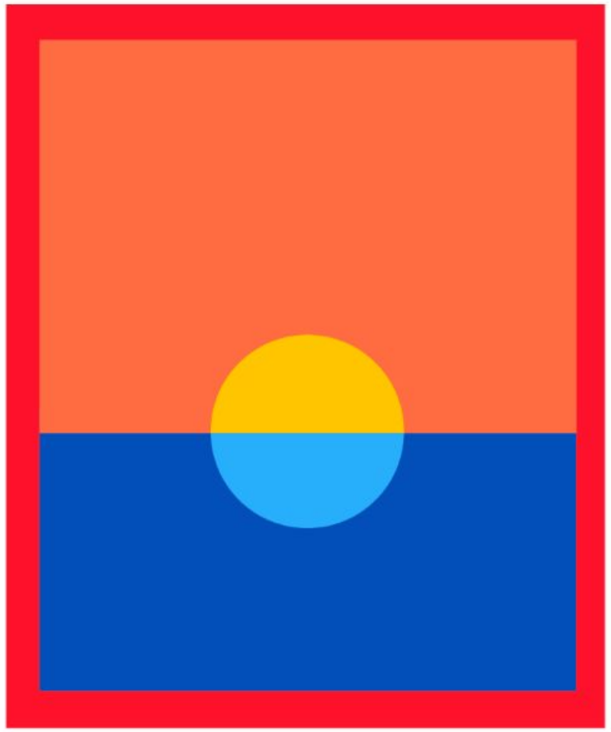
Custom Gradle Plugin - Generate kover report task

```
object KoverExecutor {

   fun executeWithHtmlReport(project: Project, task: Task) {
       val flavourPostfix = getFlavor(project)
       task.dependsOn("koverHtmlReport$flavorPostfix")
   }
}
```

Требования к custom gradle plugin

Какие команды нам нужны?

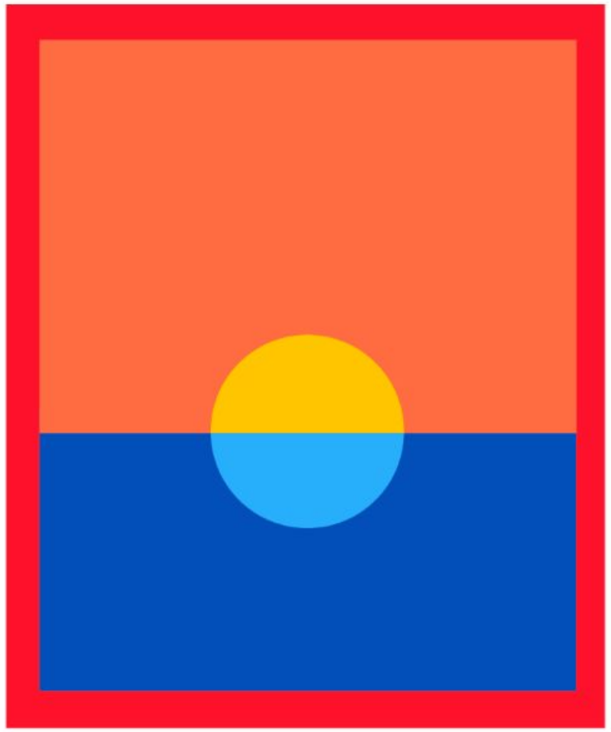
- Команда создания baseline ✅
- Команда создания репорта ✅
- Команда проверки покрытия



Требования к custom gradle plugin

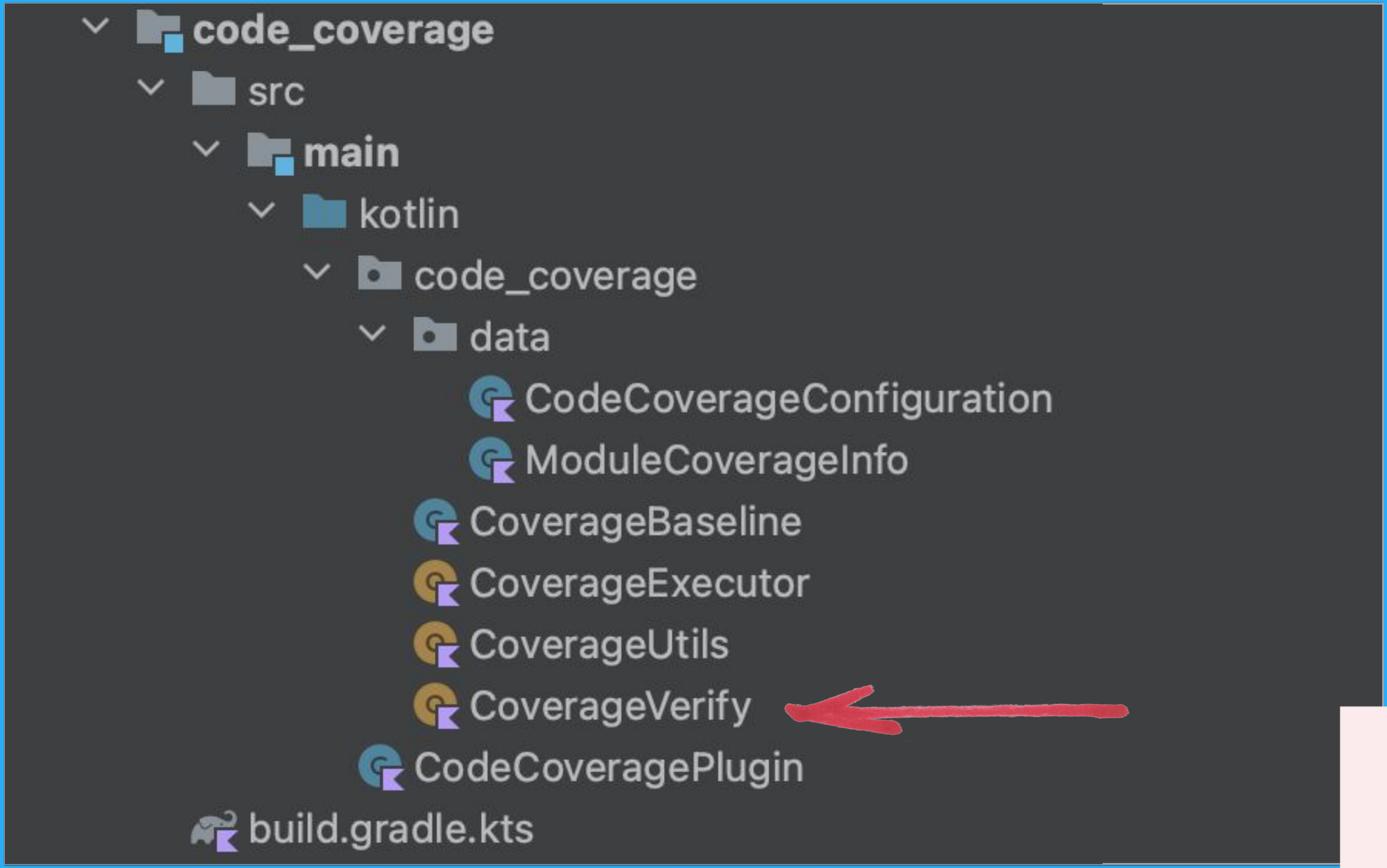
Специфичные требования?

- 🔵 Baseline должен быть удобочитаемым ✅
- 🔵 Плагин выкидывает исключение если покрытие уменьшилось
- 🔵 Команды запускают kover вне зависимости от типа модуля и flavor ✅



Создадим Generate kover verify

Custom Gradle Plugin Начало



```
code_coverage
  src
    main
      kotlin
        code_coverage
          data
            CodeCoverageConfiguration
            ModuleCoverageInfo
          CoverageBaseline
          CoverageExecutor
          CoverageUtils
          CoverageVerify
        CodeCoveragePlugin
  build.gradle.kts
```

Custom Gradle Plugin - verify task

```kotlin
object CoverageVerify {

    fun verify(project: Project)

    private fun loadBaselineFile(project: Project): List<ModuleCoverageInfo>

    private fun assertPercentage(
        project: Project,
        newPercent: Float,
        baselinePercent: Float
    )
}
```

Custom Gradle Plugin - verify task

```kotlin
object CoverageVerify {

   private var baselineFile: File? = null
   private var baseline: List<ModuleCoverageInfo> = emptyList()

   private fun loadBaselineFile(project: Project): List<ModuleCoverageInfo> {
       if (baseline.isNotEmpty()) return baseline
       val baselinePath = CoverageUtils.getConfiguration(project).baselinePath
       val baselineFileName = CoverageUtils.getConfiguration(project).baselineFileName

       baselineFile = File(baselinePath, "$baselineFileName.json")
       if (baselineFile?.exists() ≠ true) {
           System.err.println("Code coverage baseline file wasn't found.")
           baselineFile = null
       }

       baseline = Json.decodeFromString<List<ModuleCoverageInfo>>(
           baselineFile!!.readText(Charsets.UTF_8)
       )

       return baseline
   }
```

Custom Gradle Plugin - verify task

```kotlin
object CoverageVerify {

   private var baselineFile: File? = null
   private var baseline: List<ModuleCoverageInfo> = emptyList()

   private fun loadBaselineFile(project: Project): List<ModuleCoverageInfo> {
       if (baseline.isNotEmpty()) return baseline
       val baselinePath = CoverageUtils.getConfiguration(project).baselinePath
       val baselineFileName = CoverageUtils.getConfiguration(project).baselineFileName

       baselineFile = File(baselinePath, "$baselineFileName.json")
       if (baselineFile?.exists() ≠ true) {
           System.err.println("Code coverage baseline file wasn't found.")
           baselineFile = null
       }

       baseline = Json.decodeFromString<List<ModuleCoverageInfo>>(
           baselineFile!!.readText(Charsets.UTF_8)
       )

       return baseline
   }
```

Custom Gradle Plugin - verify task

```kotlin
object CoverageVerify {

   private var baselineFile: File? = null
   private var baseline: List<ModuleCoverageInfo> = emptyList()

   private fun loadBaselineFile(project: Project): List<ModuleCoverageInfo> {
       if (baseline.isNotEmpty()) return baseline
       val baselinePath = CoverageUtils.getConfiguration(project).baselinePath
       val baselineFileName = CoverageUtils.getConfiguration(project).baselineFileName

       baselineFile = File(baselinePath, "$baselineFileName.json")
       if (baselineFile?.exists() ≠ true) {
           System.err.println("Code coverage baseline file wasn't found.")
           baselineFile = null
       }

       baseline = Json.decodeFromString<List<ModuleCoverageInfo>>(
           baselineFile!!.readText(Charsets.UTF_8)
       )

       return baseline
   }
```

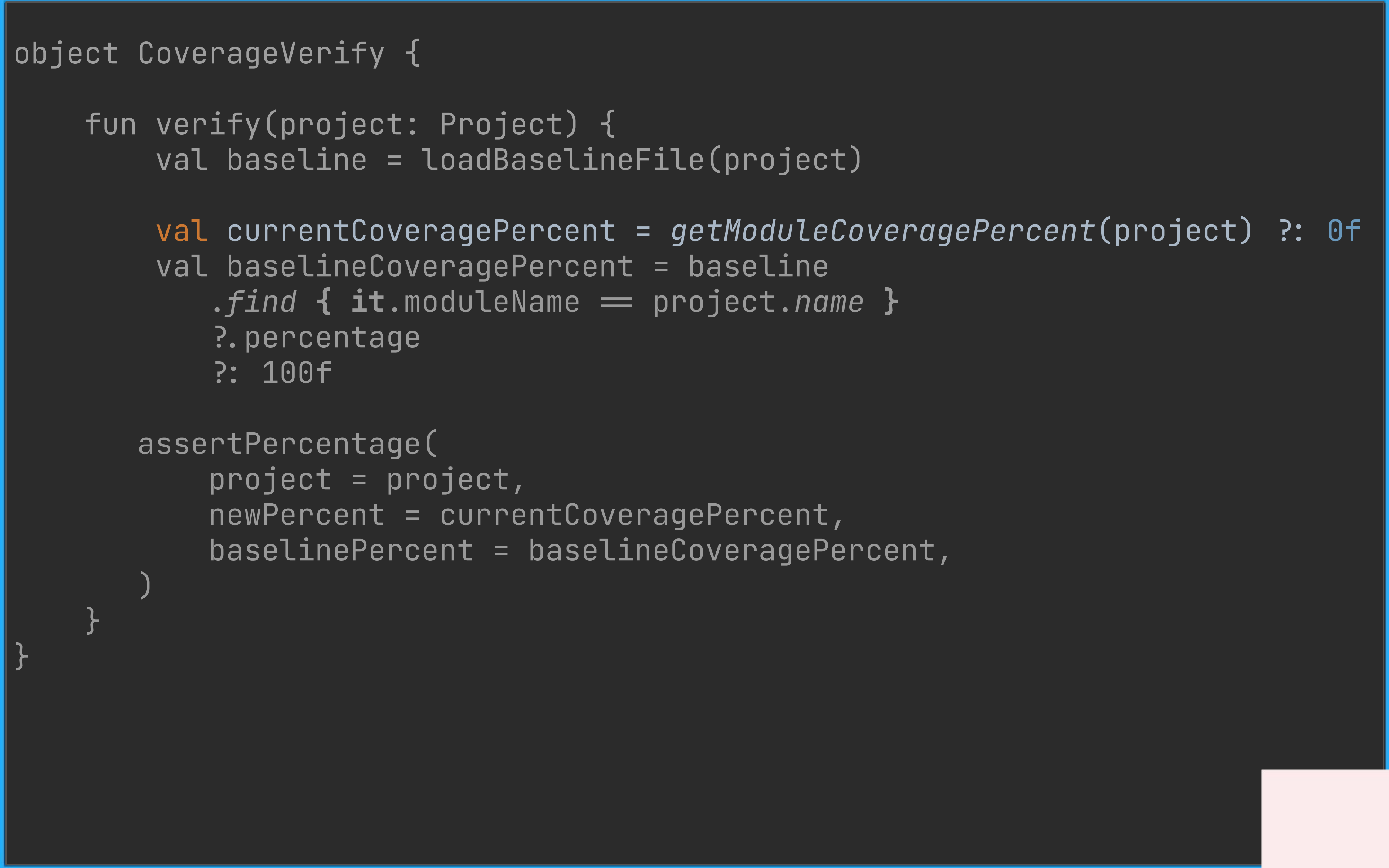
Custom Gradle Plugin - verify task

```kotlin
object CoverageVerify {

    fun verify(project: Project) {
        val baseline = loadBaselineFile(project)

        val currentCoveragePercent = getModuleCoveragePercent(project) ?: 0f
        val baselineCoveragePercent = baseline
            .find { it.moduleName == project.name }
            ?.percentage
            ?: 100f

       assertPercentage(
           project = project,
           newPercent = currentCoveragePercent,
           baselinePercent = baselineCoveragePercent,
       )
    }
}
```

Custom Gradle Plugin - verify task

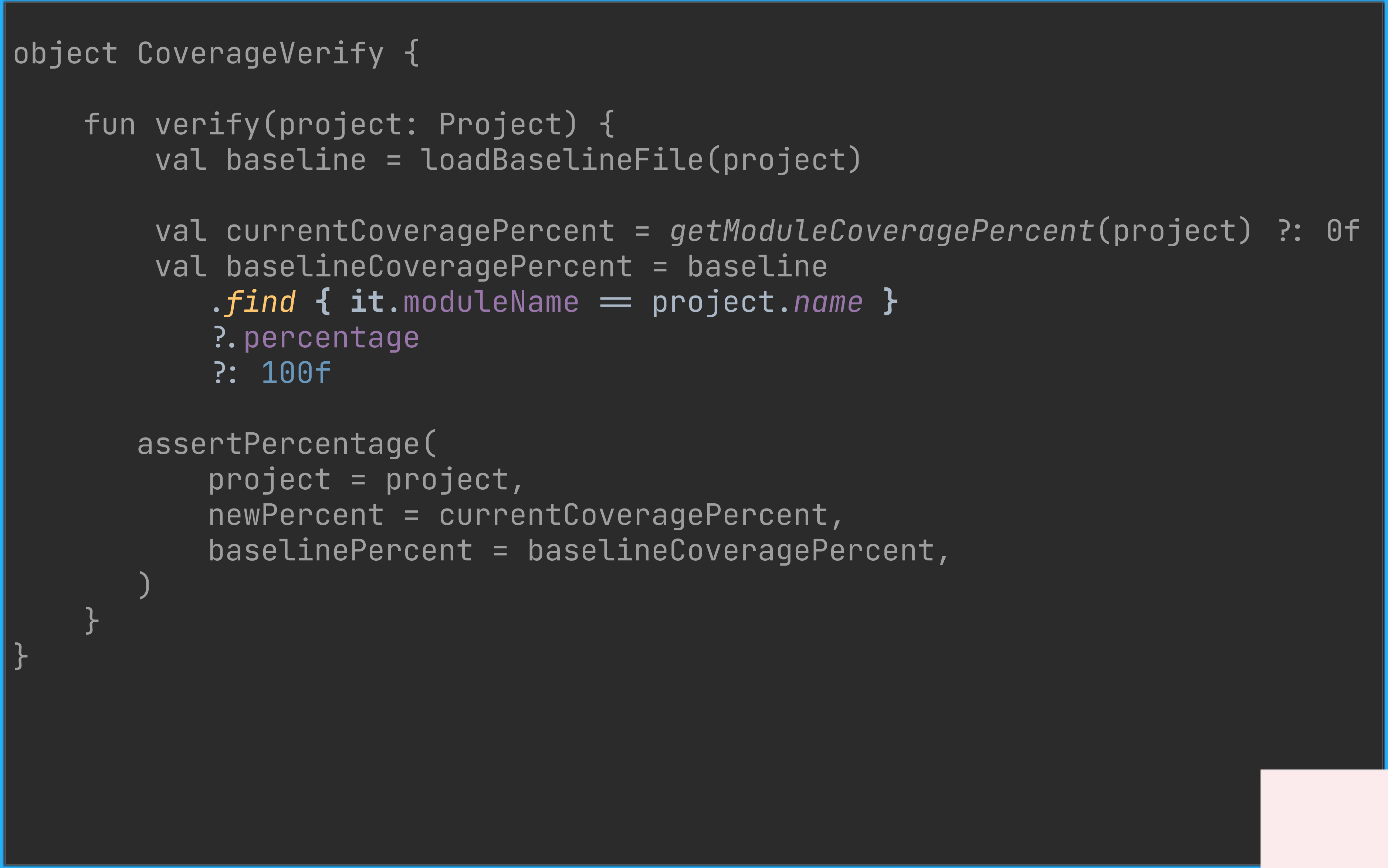
```kotlin
object CoverageVerify {

    fun verify(project: Project) {
        val baseline = loadBaselineFile(project)

        val currentCoveragePercent = getModuleCoveragePercent(project) ?: 0f
        val baselineCoveragePercent = baseline
           .find { it.moduleName == project.name }
           ?.percentage
           ?: 100f

       assertPercentage(
           project = project,
           newPercent = currentCoveragePercent,
           baselinePercent = baselineCoveragePercent,
       )
    }
}
```

Custom Gradle Plugin - verify task



```
object CoverageVerify {

    fun verify(project: Project) {
        val baseline = loadBaselineFile(project)

        val currentCoveragePercent = getModuleCoveragePercent(project) ?: 0f
        val baselineCoveragePercent = baseline
           .find { it.moduleName == project.name }
           ?.percentage
           ?: 100f

       assertPercentage(
           project = project,
           newPercent = currentCoveragePercent,
           baselinePercent = baselineCoveragePercent,
       )
    }
}
```

Custom Gradle Plugin - verify task

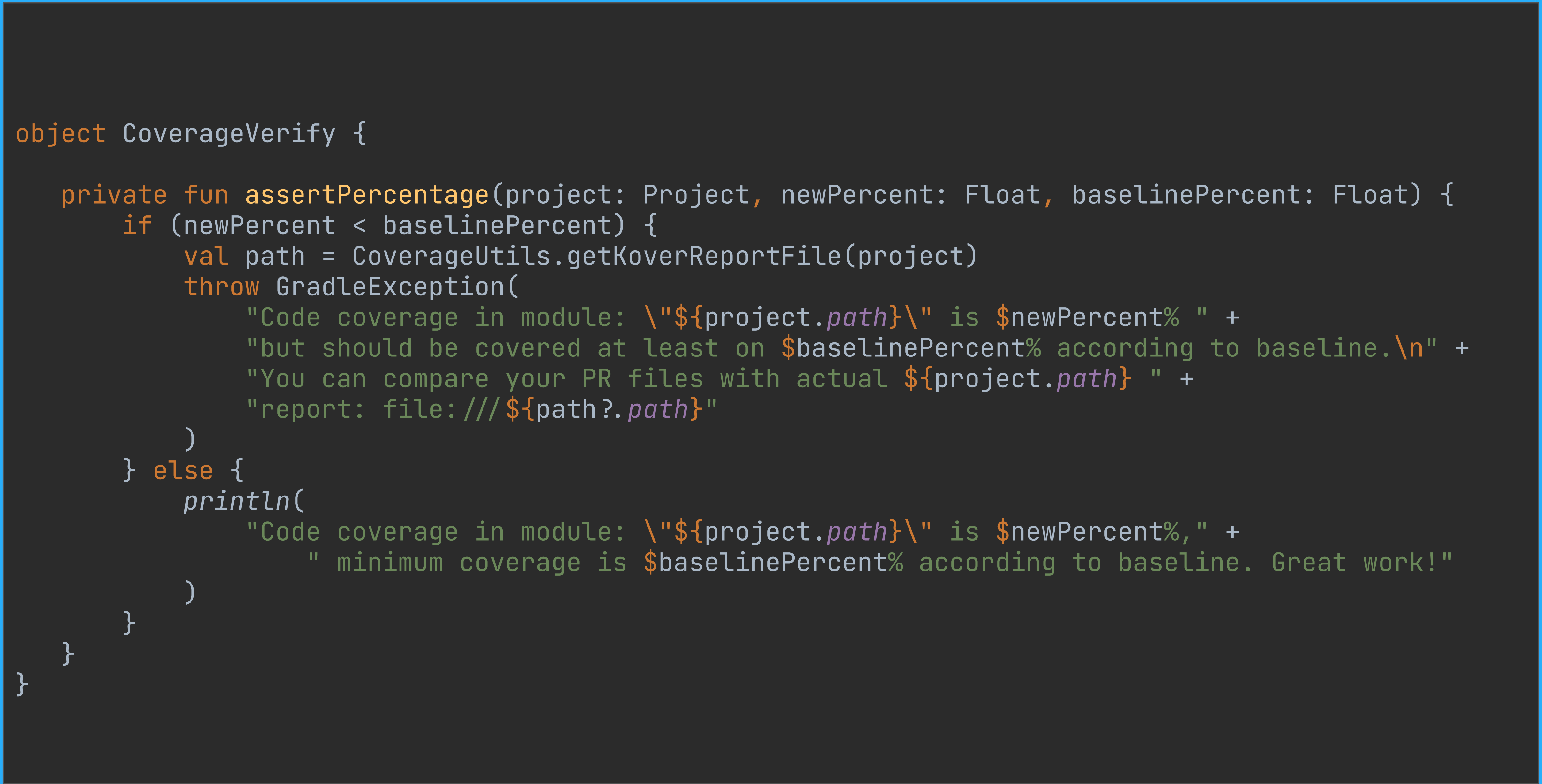


```
object CoverageVerify {

    fun verify(project: Project) {
        val baseline = loadBaselineFile(project)

        val currentCoveragePercent = getModuleCoveragePercent(project) ?: 0f
        val baselineCoveragePercent = baseline
            .find { it.moduleName == project.name }
            ?.percentage
            ?: 100f

       assertPercentage(
           project = project,
           newPercent = currentCoveragePercent,
           baselinePercent = baselineCoveragePercent,
       )
    }
}
```

Custom Gradle Plugin - verify task

```kotlin
object CoverageVerify {

    fun verify(project: Project) {
        val baseline = loadBaselineFile(project)

        val currentCoveragePercent = getModuleCoveragePercent(project) ?: 0f
        val baselineCoveragePercent = baseline
           .find { it.moduleName == project.name }
           ?.percentage
           ?: 100f

       assertPercentage(
           project = project,
           newPercent = currentCoveragePercent,
           baselinePercent = baselineCoveragePercent,
       )
    }
}
```

Custom Gradle Plugin - verify task



```
object CoverageVerify {

   private fun assertPercentage(project: Project, newPercent: Float, baselinePercent: Float) {
       if (newPercent < baselinePercent) {
           val path = CoverageUtils.getKoverReportFile(project)
           throw GradleException(
               "Code coverage in module: \"${project.path}\" is $newPercent% " +
               "but should be covered at least on $baselinePercent% according to baseline.\n" +
               "You can compare your PR files with actual ${project.path} " +
               "report: file:///${path?.path}"
           )
       } else {
           println(
               "Code coverage in module: \"${project.path}\" is $newPercent%," +
                   " minimum coverage is $baselinePercent% according to baseline. Great work!"
           )
       }
   }
}
```

Custom Gradle Plugin - verify task

```
object CoverageVerify {

   private fun assertPercentage(project: Project, newPercent: Float, baselinePercent: Float) {
       if (newPercent < baselinePercent) {
           val path = CoverageUtils.getKoverReportFile(project)
           throw GradleException(
               "Code coverage in module: \"${project.path}\" is $newPercent% " +
               "but should be covered at least on $baselinePercent% according to baseline.\n" +
               "You can compare your PR files with actual ${project.path} " +
               "report: file:///${path?.path}"
           )
       } else {
           println(
               "Code coverage in module: \"${project.path}\" is $newPercent%," +
                   " minimum coverage is $baselinePercent% according to baseline. Great work!"
           )
       }
   }
}
```

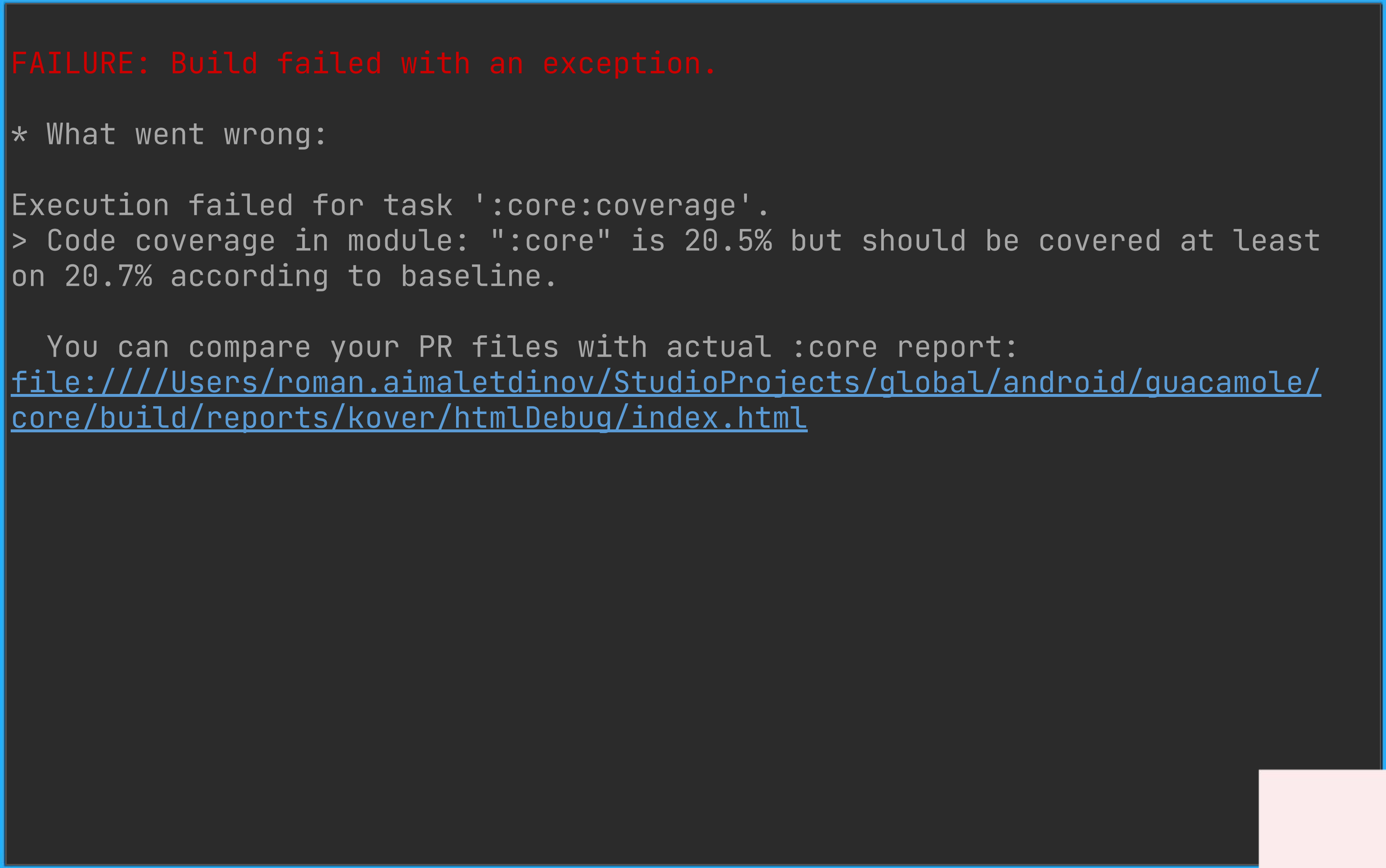
Custom Gradle Plugin - verify task

```kotlin
object CoverageVerify {

   private fun assertPercentage(project: Project, newPercent: Float, baselinePercent: Float) {
       if (newPercent < baselinePercent) {
           val path = CoverageUtils.getKoverReportFile(project)
           throw GradleException(
               "Code coverage in module: \"${project.path}\" is $newPercent% " +
               "but should be covered at least on $baselinePercent% according to baseline.\n" +
               "You can compare your PR files with actual ${project.path} " +
               "report: file:///${path?.path}"
           )
       } else {
           println(
               "Code coverage in module: \"${project.path}\" is $newPercent%," +
                   " minimum coverage is $baselinePercent% according to baseline. Great work!"
           )
       }
   }
}
```

Custom Gradle Plugin - verify task

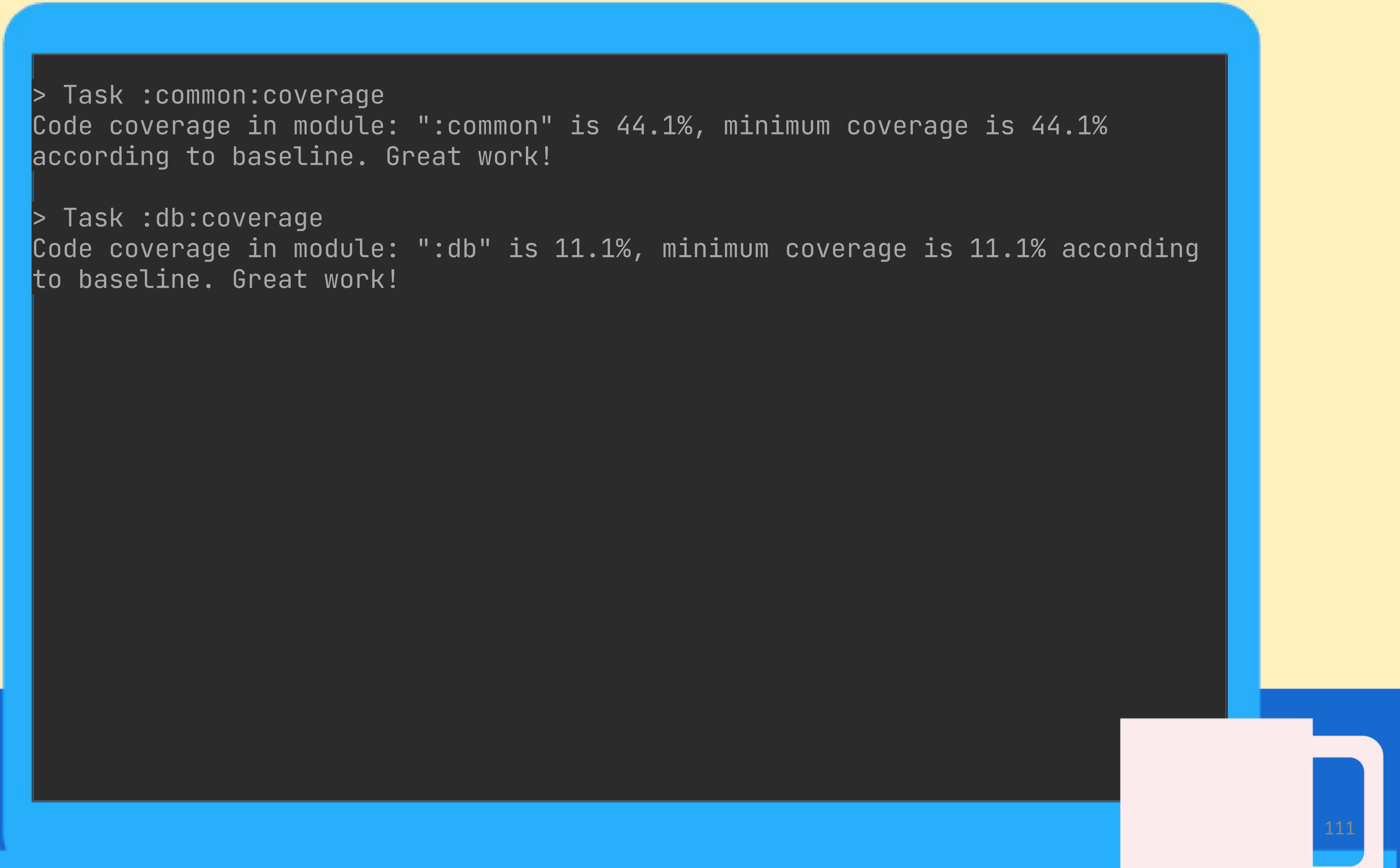
```kotlin
object CoverageVerify {

   private fun assertPercentage(project: Project, newPercent: Float, baselinePercent: Float) {
       if (newPercent < baselinePercent) {
           val path = CoverageUtils.getKoverReportFile(project)
           throw GradleException(
               "Code coverage in module: \"${project.path}\" is $newPercent% " +
               "but should be covered at least on $baselinePercent% according to baseline.\n" +
               "You can compare your PR files with actual ${project.path} " +
               "report: file:///${path?.path}"
           )
       } else {
           println(
               "Code coverage in module: \"${project.path}\" is $newPercent%," +
                   " minimum coverage is $baselinePercent% according to baseline. Great work!"
           )
       }
   }
}
```

Custom Gradle Plugin - verify task



```
FAILURE: Build failed with an exception.

* What went wrong:

Execution failed for task ':core:coverage'.
> Code coverage in module: ":core" is 20.5% but should be covered at least
on 20.7% according to baseline.

  You can compare your PR files with actual :core report:
file:////Users/roman.aimaletdinov/StudioProjects/global/android/guacamole/
core/build/reports/kover/htmlDebug/index.html
```

Custom Gradle Plugin - verify task

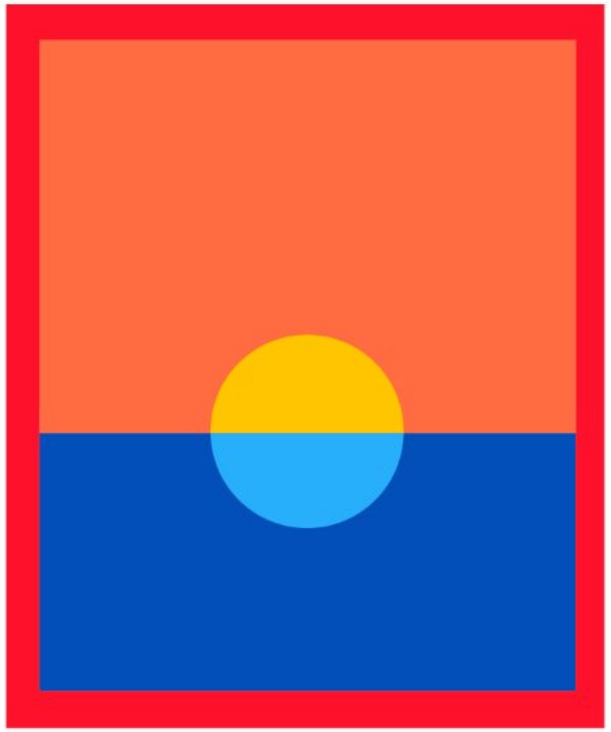


```
> Task :common:coverage
Code coverage in module: ":common" is 44.1%, minimum coverage is 44.1%
according to baseline. Great work!

> Task :db:coverage
Code coverage in module: ":db" is 11.1%, minimum coverage is 11.1% according
to baseline. Great work!
```

Требования к custom gradle plugin

Какие команды нам нужны?

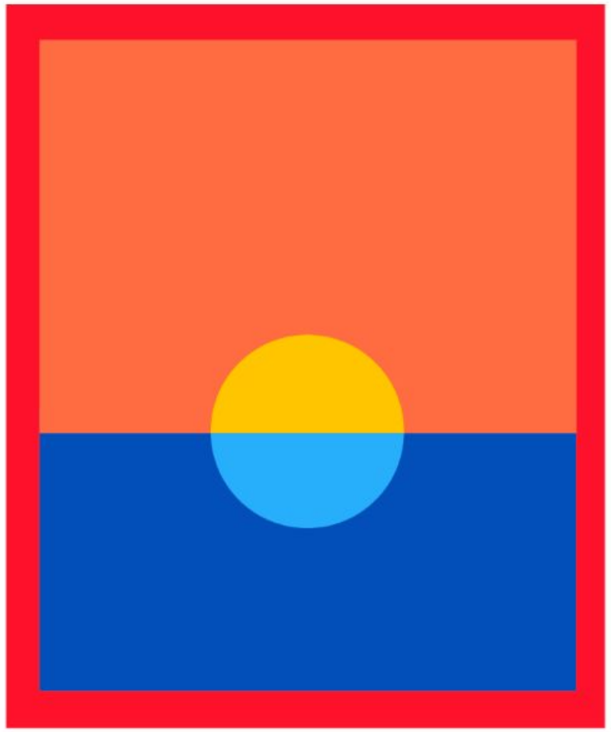
- Команда создания baseline ✅
- Команда создания репорта ✅
- Команда проверки покрытия ✅



Требования к custom gradle plugin

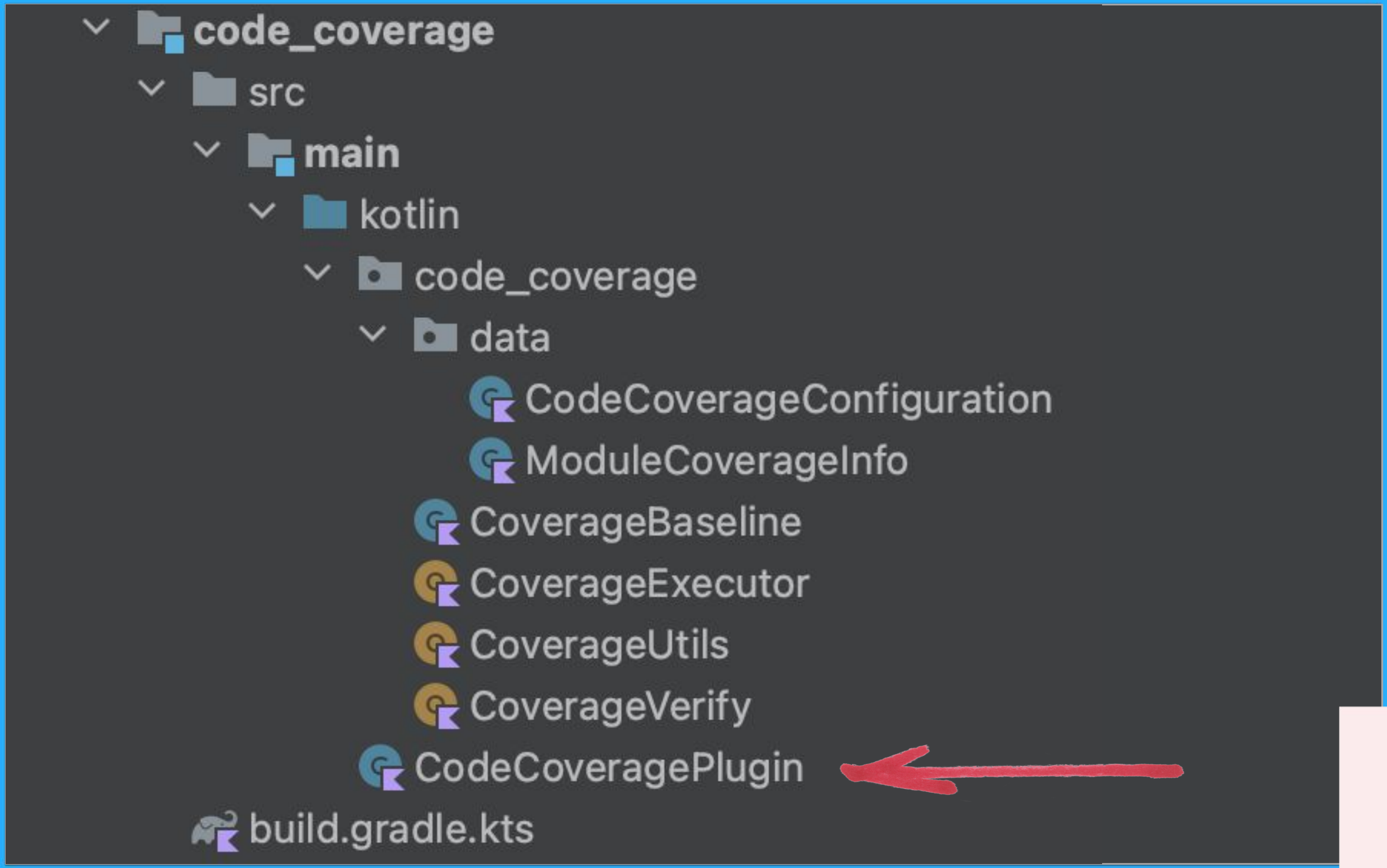
Специфичные требования?

- 🔵 Baseline должен быть удобочитаемым ✅
- 🔵 Плагин выкидывает исключение если покрытие уменьшилось ✅
- 🔵 Команды запускают kover вне зависимости от типа модуля и flavor ✅



Зарегистрируем наши таски

Custom Gradle Plugin Начало



```
code_coverage
  src
    main
      kotlin
        code_coverage
          data
            CodeCoverageConfiguration
            ModuleCoverageInfo
          CoverageBaseline
          CoverageExecutor
          CoverageUtils
          CoverageVerify
        CodeCoveragePlugin
  build.gradle.kts
```

Custom Gradle Plugin

```kotlin
class CodeCoveragePlugin : Plugin<Project> {

    // ...

    override fun apply(project: Project) {
        project.tasks.register(BASELINE_TASK_NAME) {
            doFirst {
                val coverageBaselineTask = CoverageBaseline()
                coverageBaselineTask.clearBaseline(project)
                coverageBaselineTask.update(project)
            }
        }
}
```

Custom Gradle Plugin

```kotlin
class CodeCoveragePlugin : Plugin<Project> {

    // ...

    override fun apply(project: Project) {
        // ...

        project
            .subprojects
            .forEach { module ->
                module.tasks.register(VERIFY_COVERAGE_TASK_NAME) {
                this.dependsOn(GENERATE_REPORT_TASK_NAME)
                doLast {
                    CoverageVerify.verify(module)
                }
            }

        module.tasks.register(GENERATE_REPORT_TASK_NAME) {
            CoverageExecutor.executeWithHtmlReport(module, this)
        }
    }
}
```

Требования к фиче

Процент покрытия проекта улучшается каждый PR

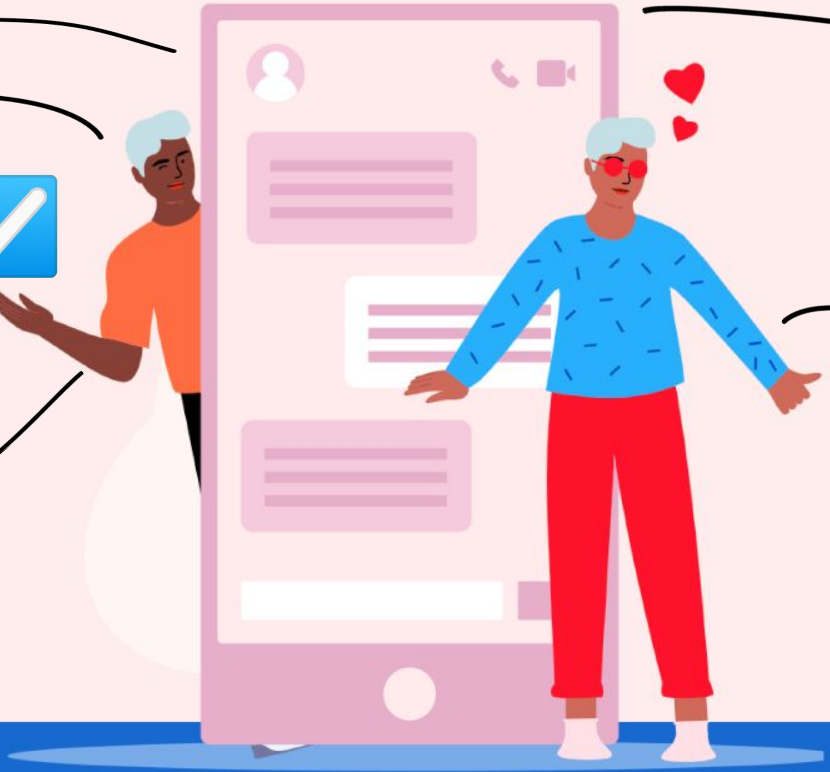
Соблюдение правил контролирует CI, а не человек

Если модуль новый, вся логика должна быть покрыта тестами ✅

Хотелось бы что бы все это было под impact analysis

Мы можем посмотреть покрытие ✅

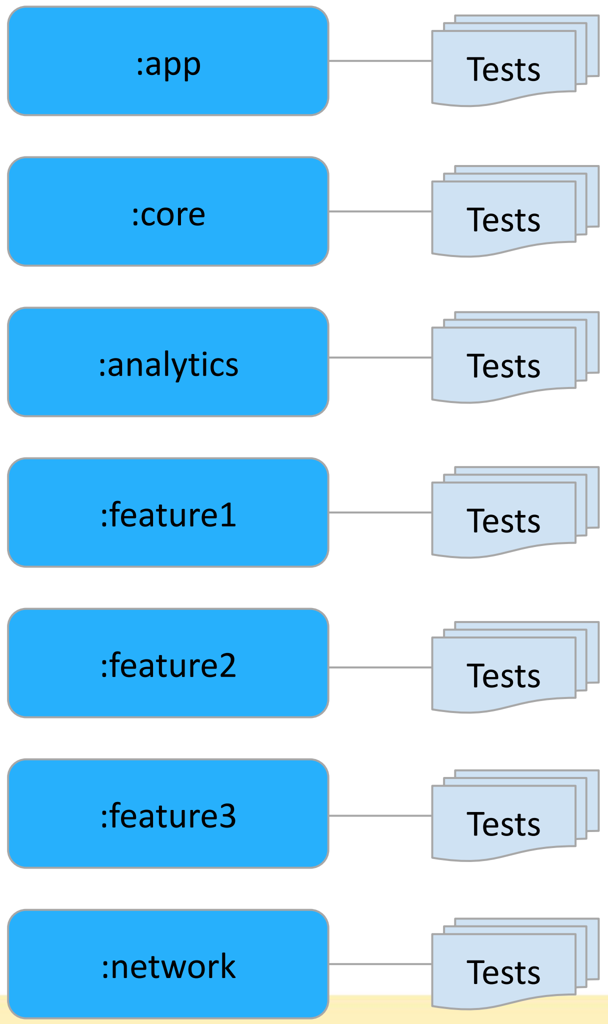
Если модуль старый, то вся новая логика должна быть покрыта ✅



Настроим impact analysis

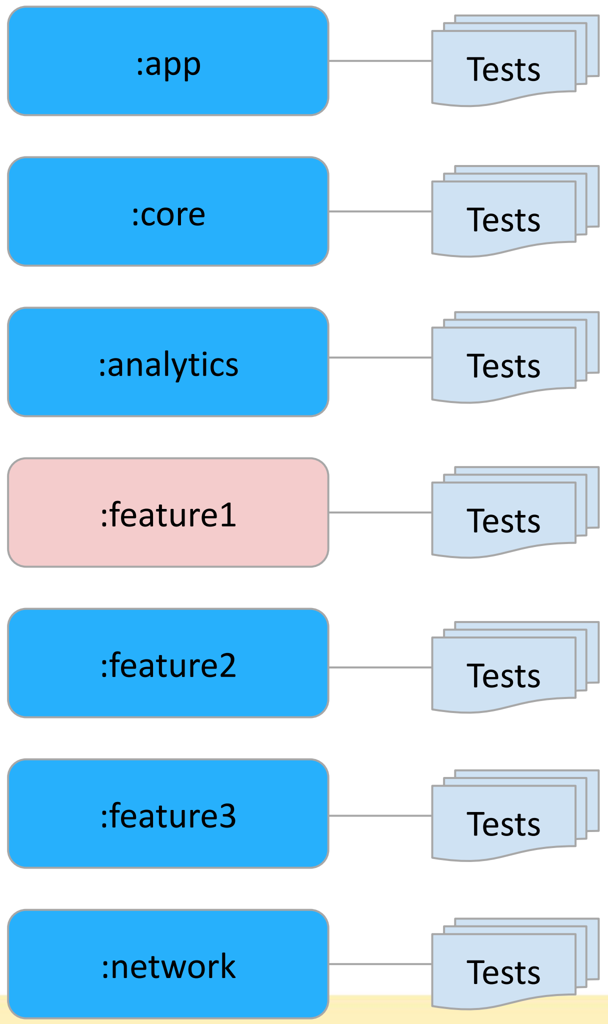
Что такое impact analysis?

Стандартный подход



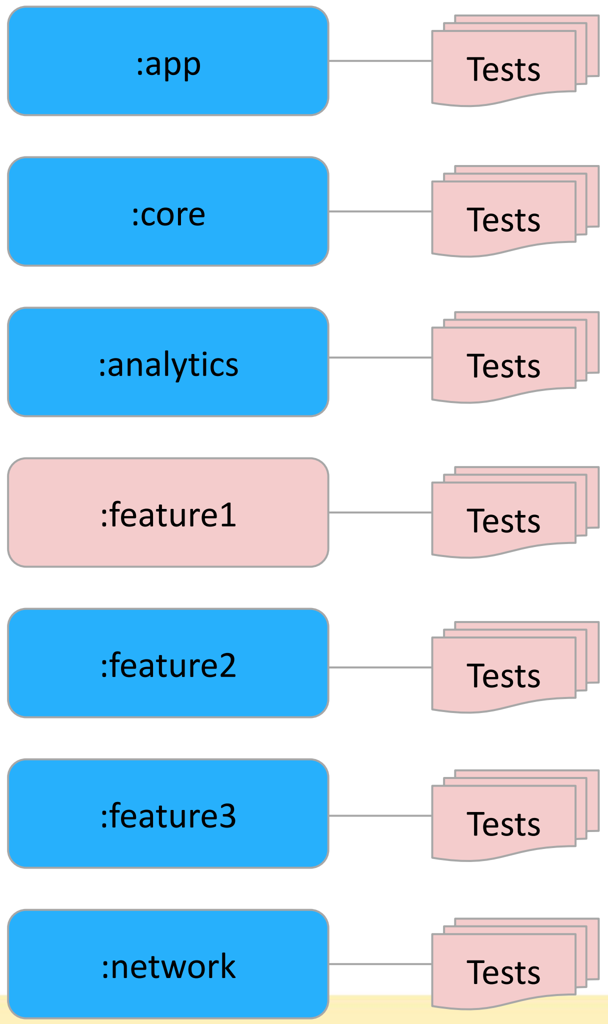
Что такое impact analysis?

Стандартный подход



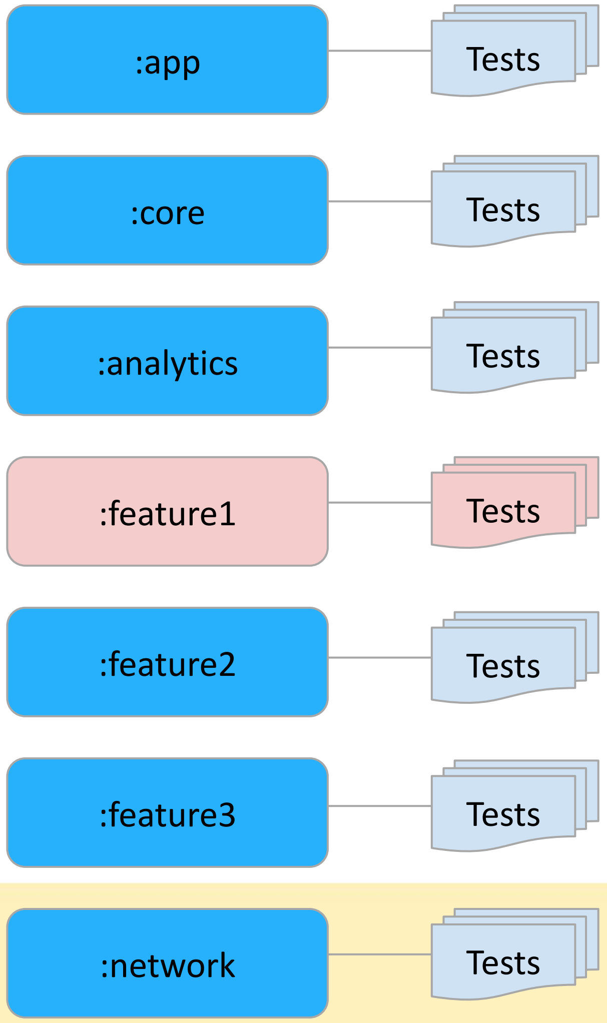
Что такое impact analysis?

Стандартный подход



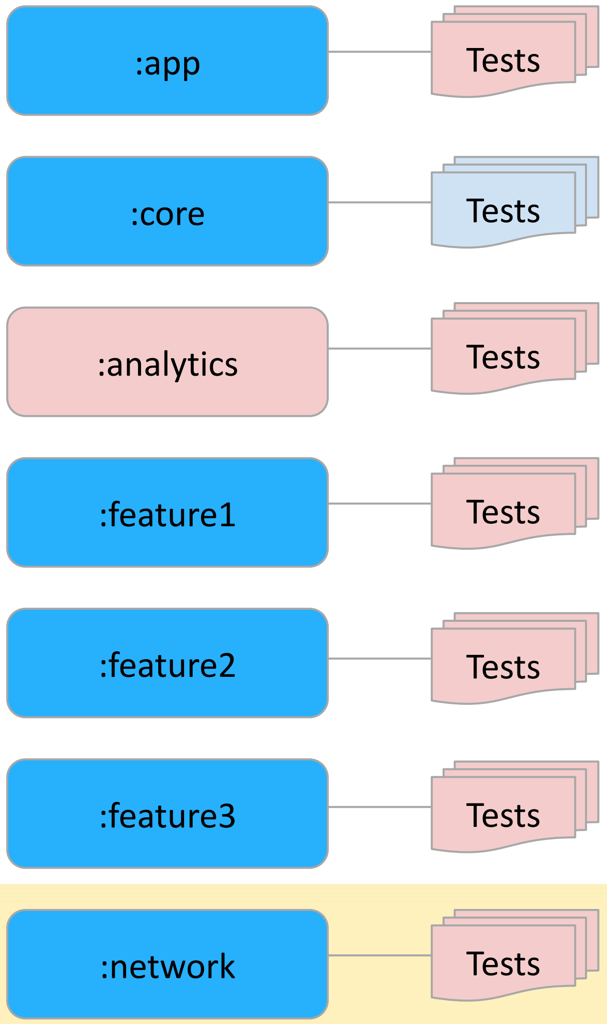
Что такое impact analysis?

Impact analysis подход



Что такое impact analysis?

Impact analysis подход

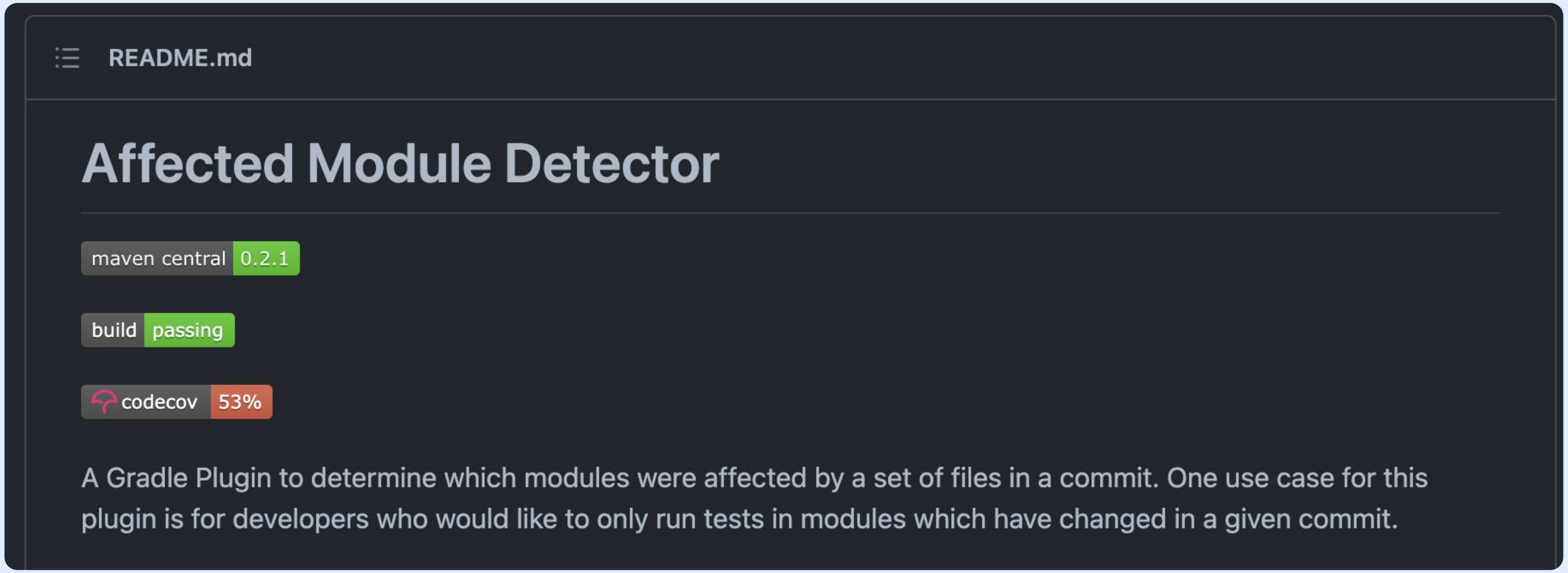


Как засетапить impact analysis?

Доклад Алены
Половковой
из Сбера

Доклад от Максима
Щепалина
из Тинькофф

Как засетапить impact analysis?


README.md

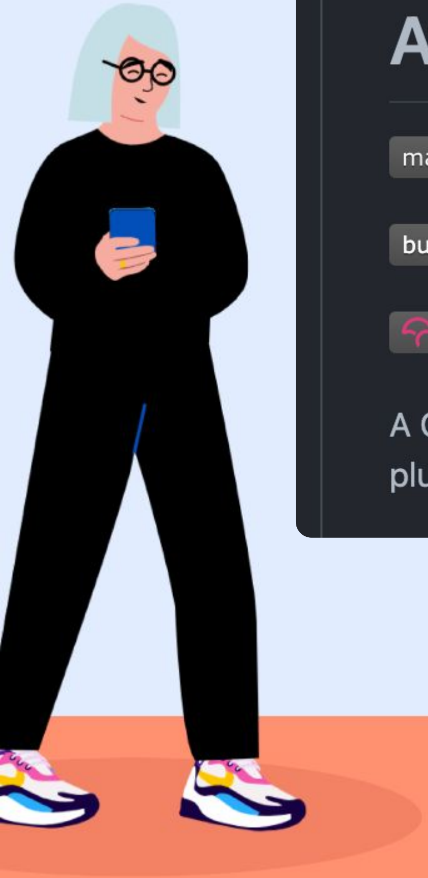
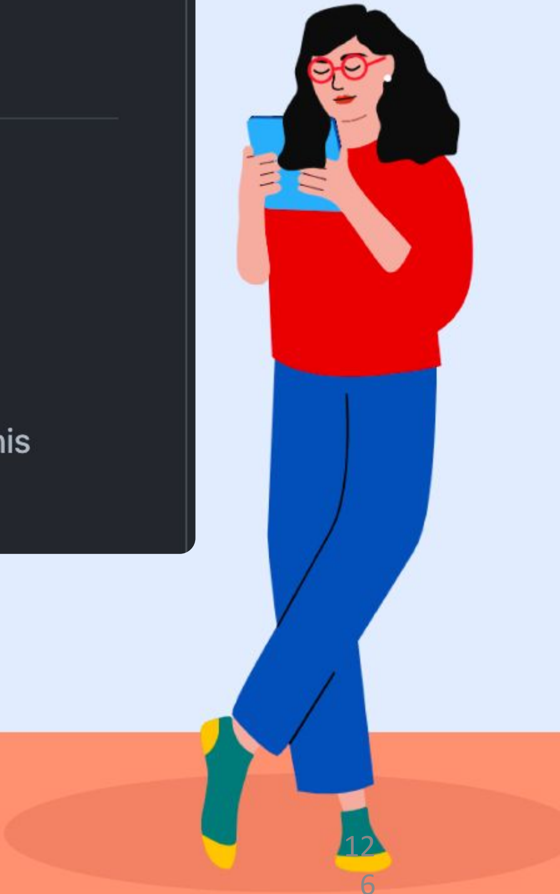
Affected Module Detector

maven central 0.2.1

build passing

codecov 53%

A Gradle Plugin to determine which modules were affected by a set of files in a commit. One use case for this plugin is for developers who would like to only run tests in modules which have changed in a given commit.

Настроим impact analysis

```
affectedModuleDetector {
    baseDir = "${project.rootDir}"
    logFilename = "output.log"
    logFolder = "${project.rootDir}/tools/impact-analysis/output"
    specifiedBranch = "origin/main"
    compareFrom = "SpecifiedBranchCommitMergeBase"
    includeUncommitted = false
    top = "HEAD"

    customTasks = [
        new AffectedModuleConfiguration.CustomTask(
            "runUnitTestByImpactAndVerifyCoverage",
            "coverage",
            "Run unit tests by impact analysis with code coverage"
        )
    ]
}
```

Настроим impact analysis

```
affectedModuleDetector {
    baseDir = "${project.rootDir}"
    logFilename = "output.log"
    logFolder = "${project.rootDir}/tools/impact-analysis/output"
    specifiedBranch = "origin/main"
    compareFrom = "SpecifiedBranchCommitMergeBase"
    includeUncommitted = false
    top = "HEAD"

    customTasks = [
        new AffectedModuleConfiguration.CustomTask(
            "runUnitTestByImpactAndVerifyCoverage",
            "coverage",
            "Run unit tests by impact analysis with code coverage"
        )
    ]
}
```

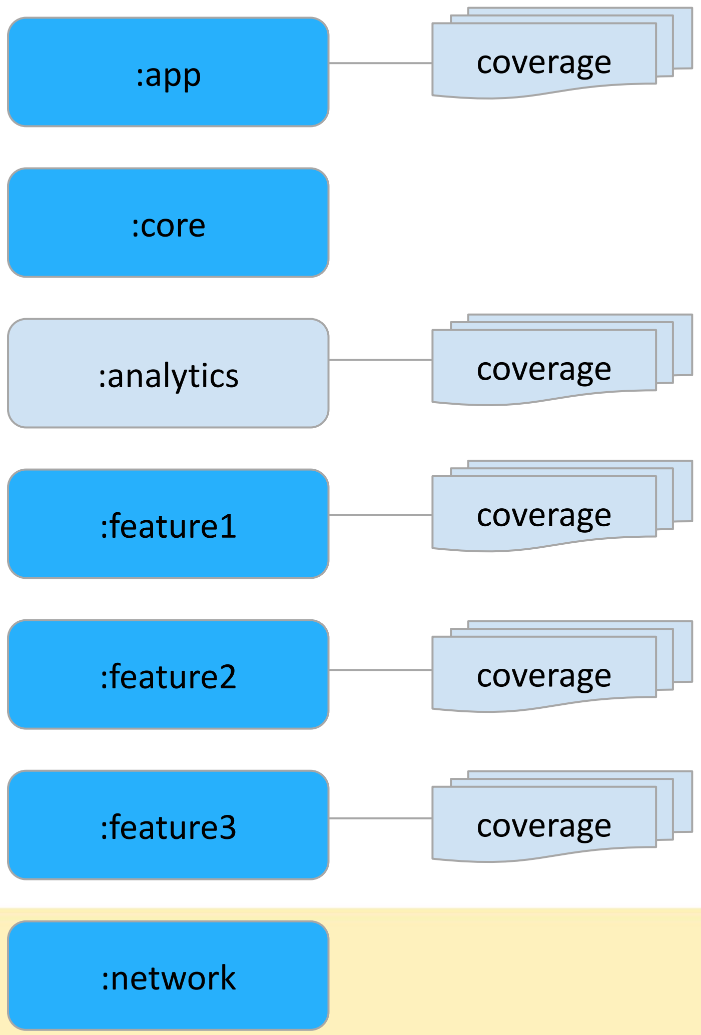
Настроим impact analysis

```
affectedModuleDetector {
    baseDir = "${project.rootDir}"
    logFilename = "output.log"
    logFolder = "${project.rootDir}/tools/impact-analysis/output"
    specifiedBranch = "origin/main"
    compareFrom = "SpecifiedBranchCommitMergeBase"
    includeUncommitted = false
    top = "HEAD"

    customTasks = [
        new AffectedModuleConfiguration.CustomTask(
            "runUnitTestByImpactAndVerifyCoverage",
            "coverage",
            "Run unit tests by impact analysis with code coverage"
        )
    ]
}
```

Настроим impact analysis

```
./gradlew runUnitTestByImpactAndVerifyCoverage
```

Настроим impact analysis



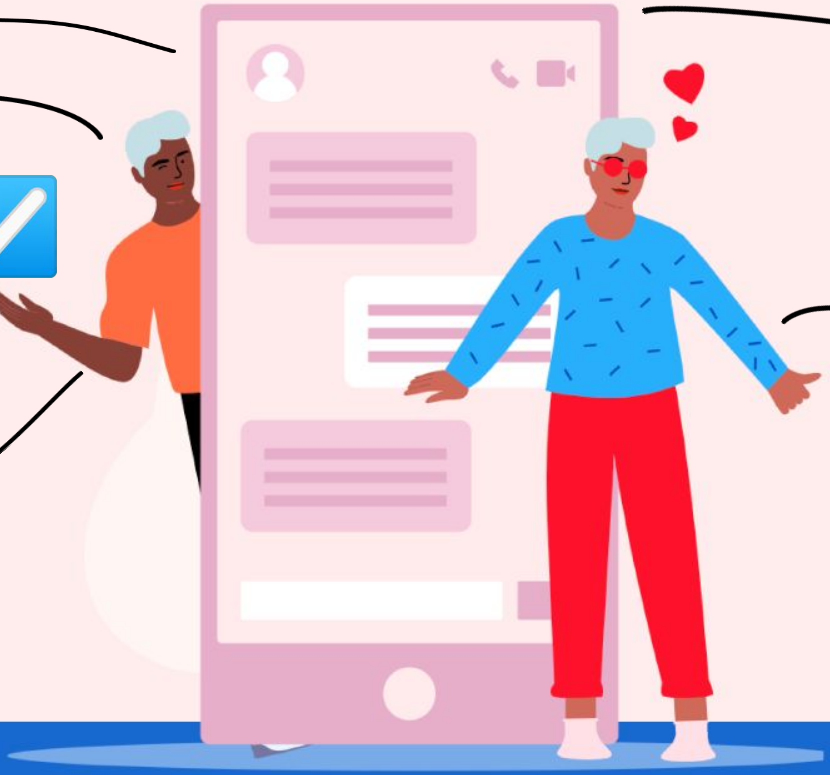
Требования к фиче

Процент покрытия проекта улучшается каждый PR

Соблюдение правил контролирует CI, а не человек

Если модуль новый, вся логика должна быть покрыта тестами ✅

Хотелось бы что бы все это было под impact analysis ✅

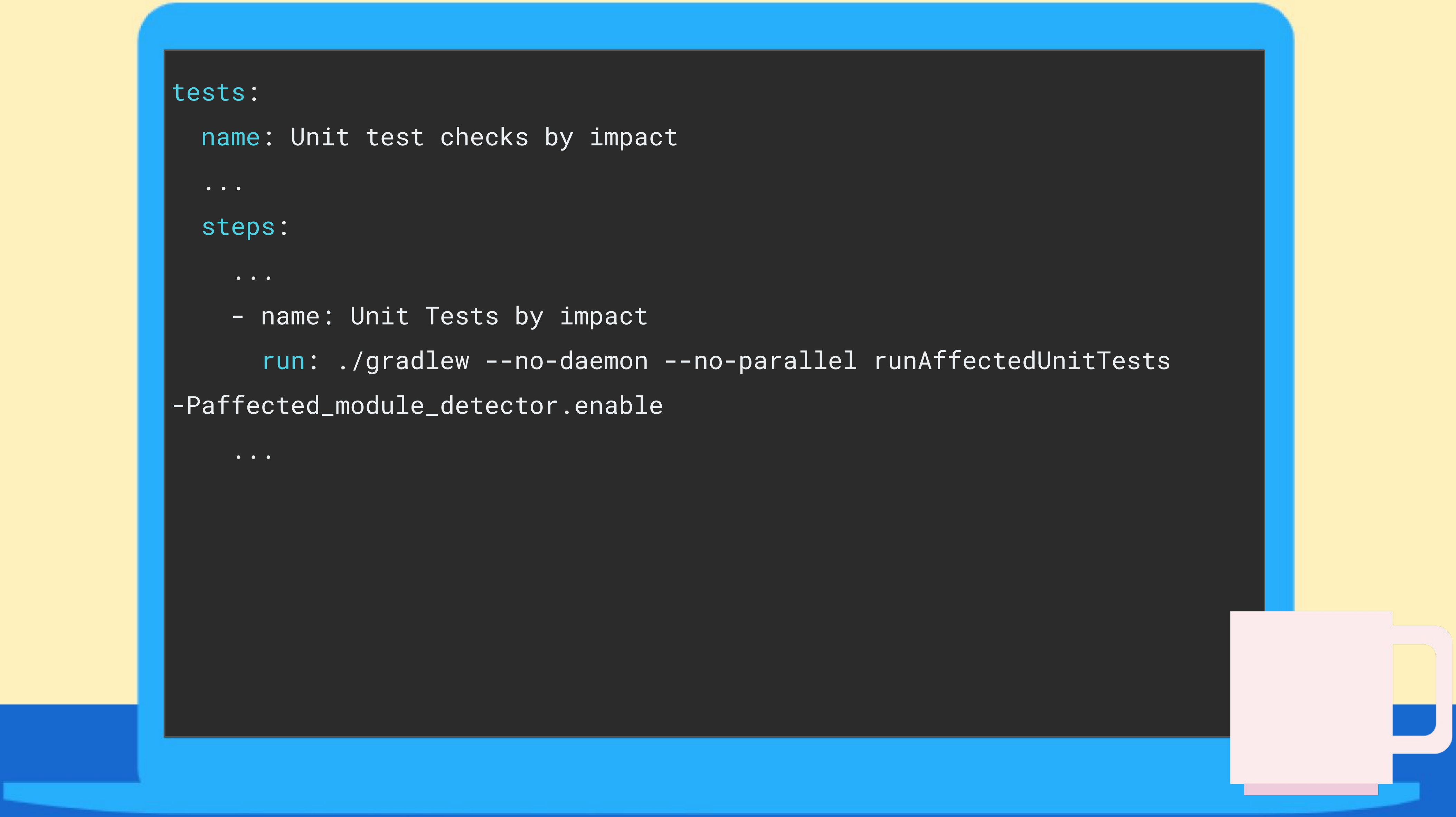


Мы можем посмотреть покрытие ✅

Если модуль старый, то вся новая логика должна быть покрыта ✅

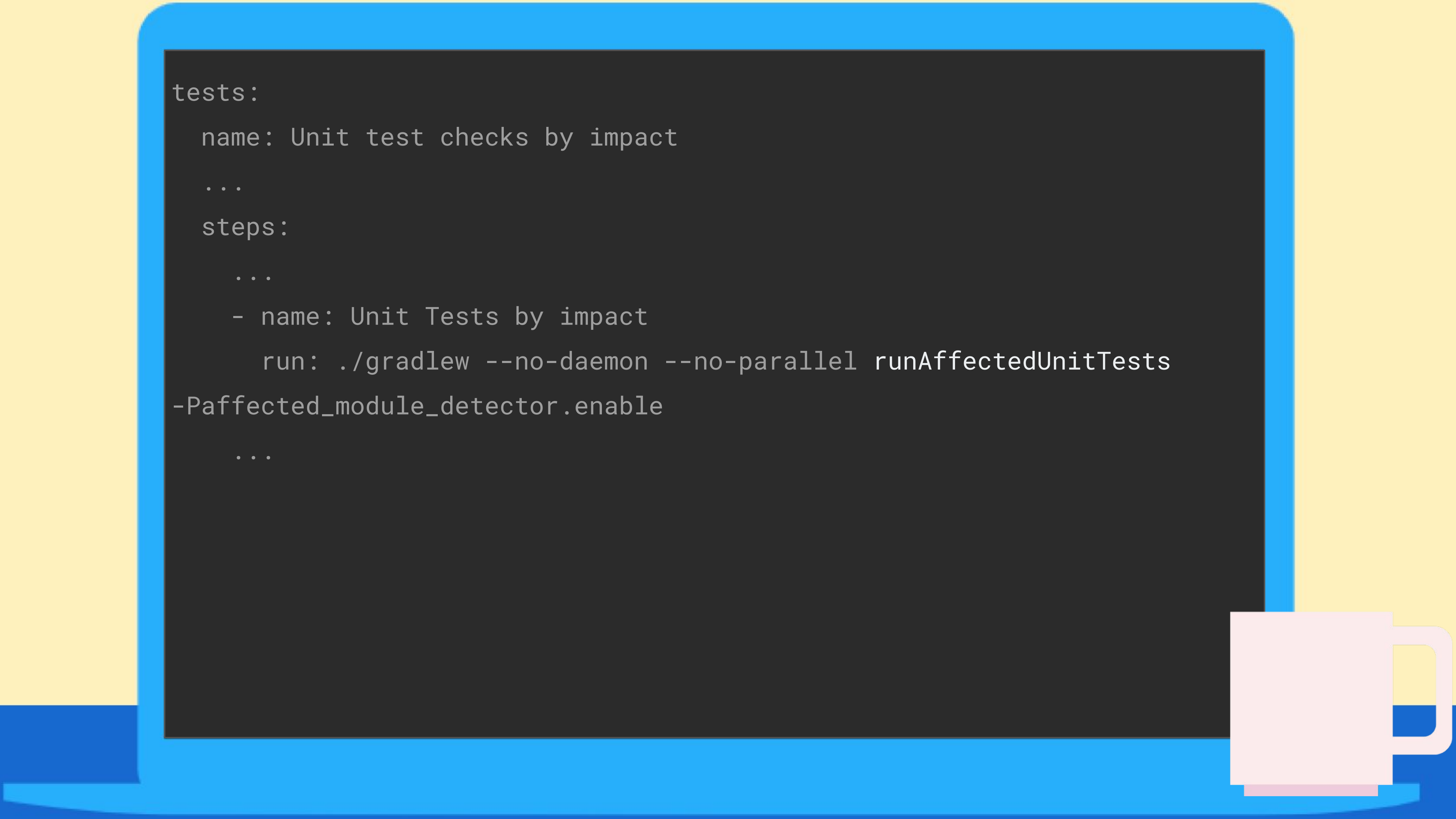
Настроим CI

Настроим CI | как было раньше



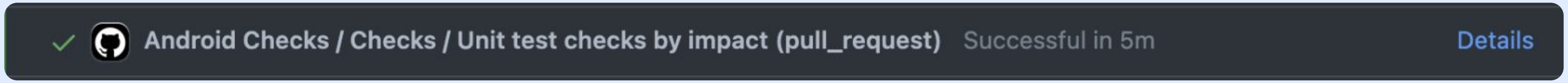
```
tests:
  name: Unit test checks by impact
  ...
  steps:
    ...
    - name: Unit Tests by impact
      run: ./gradlew --no-daemon --no-parallel runAffectedUnitTests
-Paffected_module_detector.enable
    ...
```

Настроим CI | как было раньше



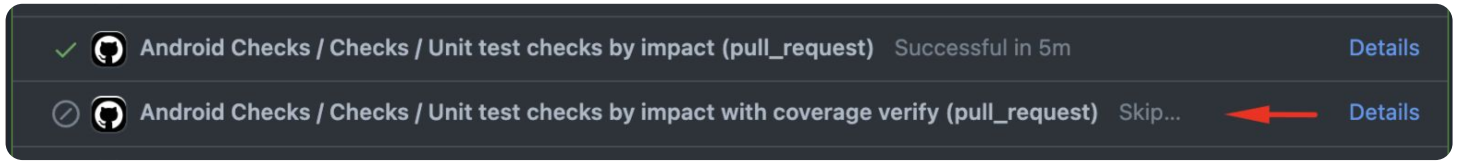
```
tests:
  name: Unit test checks by impact
  ...
  steps:
    ...
    - name: Unit Tests by impact
      run: ./gradlew --no-daemon --no-parallel runAffectedUnitTests
-Paffected_module_detector.enable
    ...
```

Настроим CI | как было раньше


✓ Android Checks / Checks / Unit test checks by impact (pull_request) Successful in 5m Details

Настроим CI | как стало

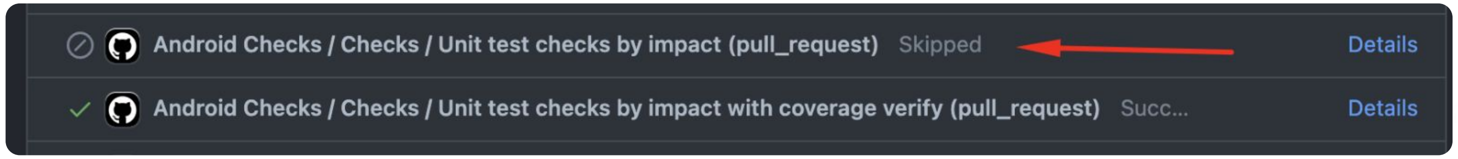
Когда мержим в feature-branch



✓ Android Checks / Checks / Unit test checks by impact (pull_request) Successful in 5m Details

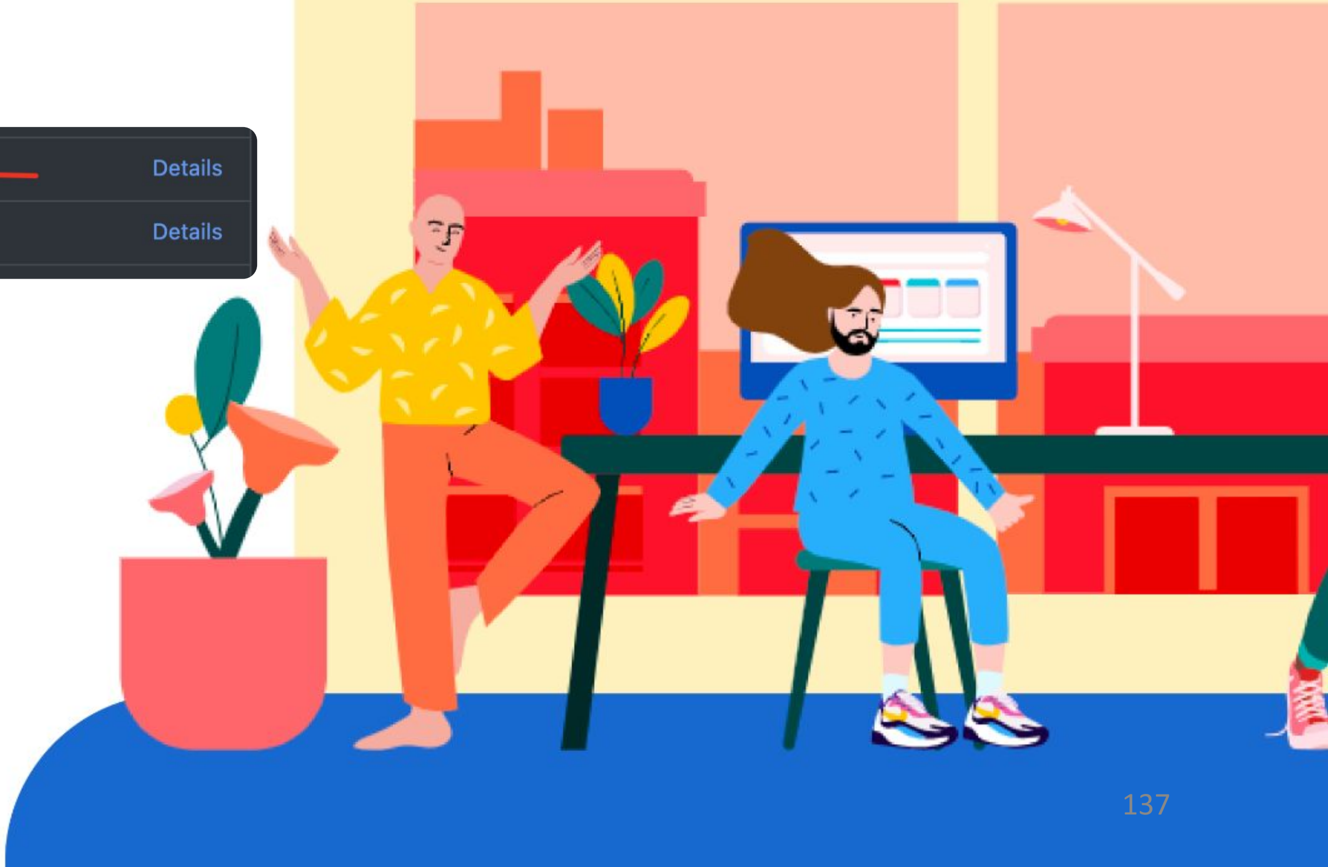
⊘ Android Checks / Checks / Unit test checks by impact with coverage verify (pull_request) Skip... Details

Когда мержим в main



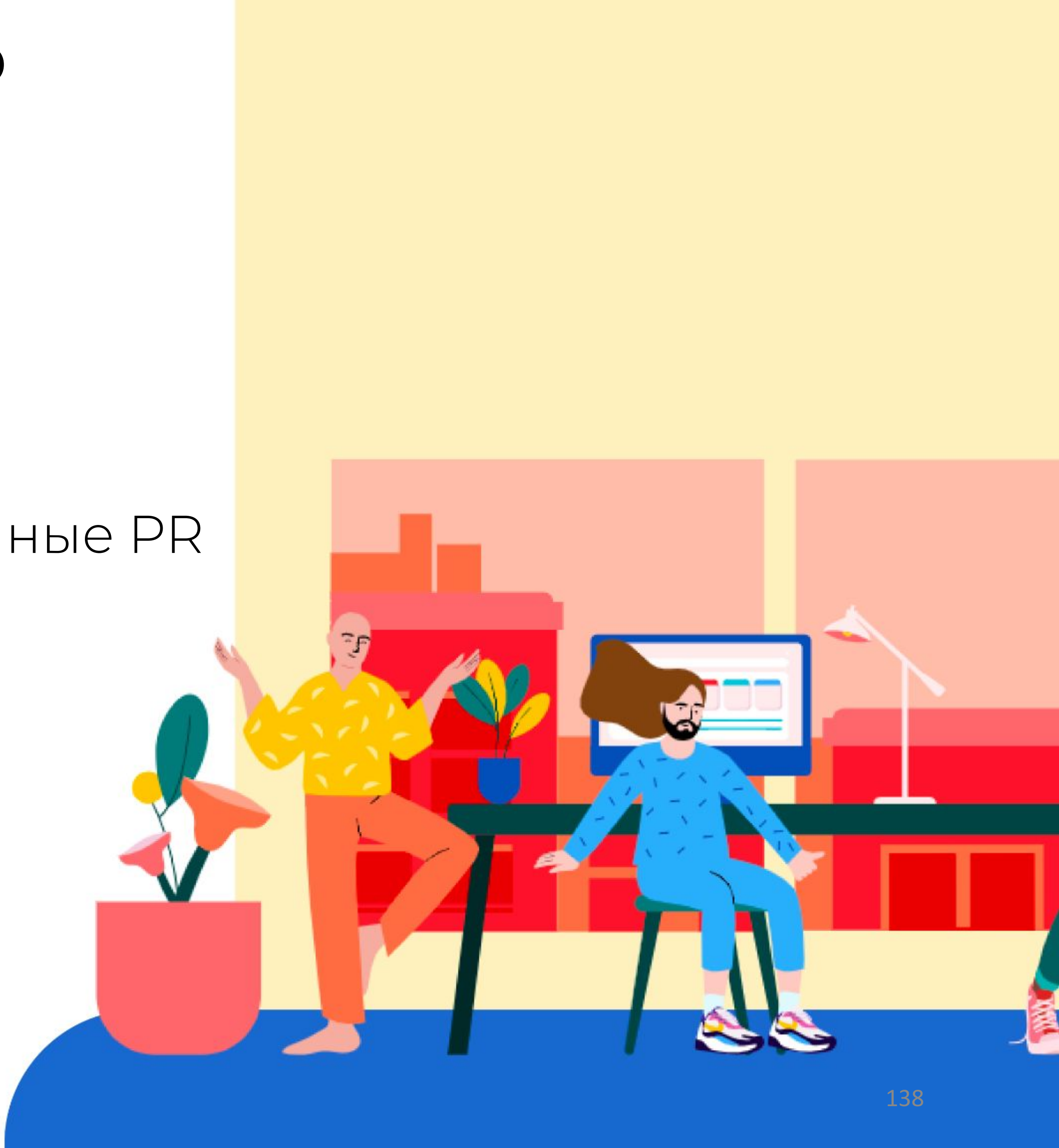
⊘ Android Checks / Checks / Unit test checks by impact (pull_request) Skipped Details

✓ Android Checks / Checks / Unit test checks by impact with coverage verify (pull_request) Succ... Details

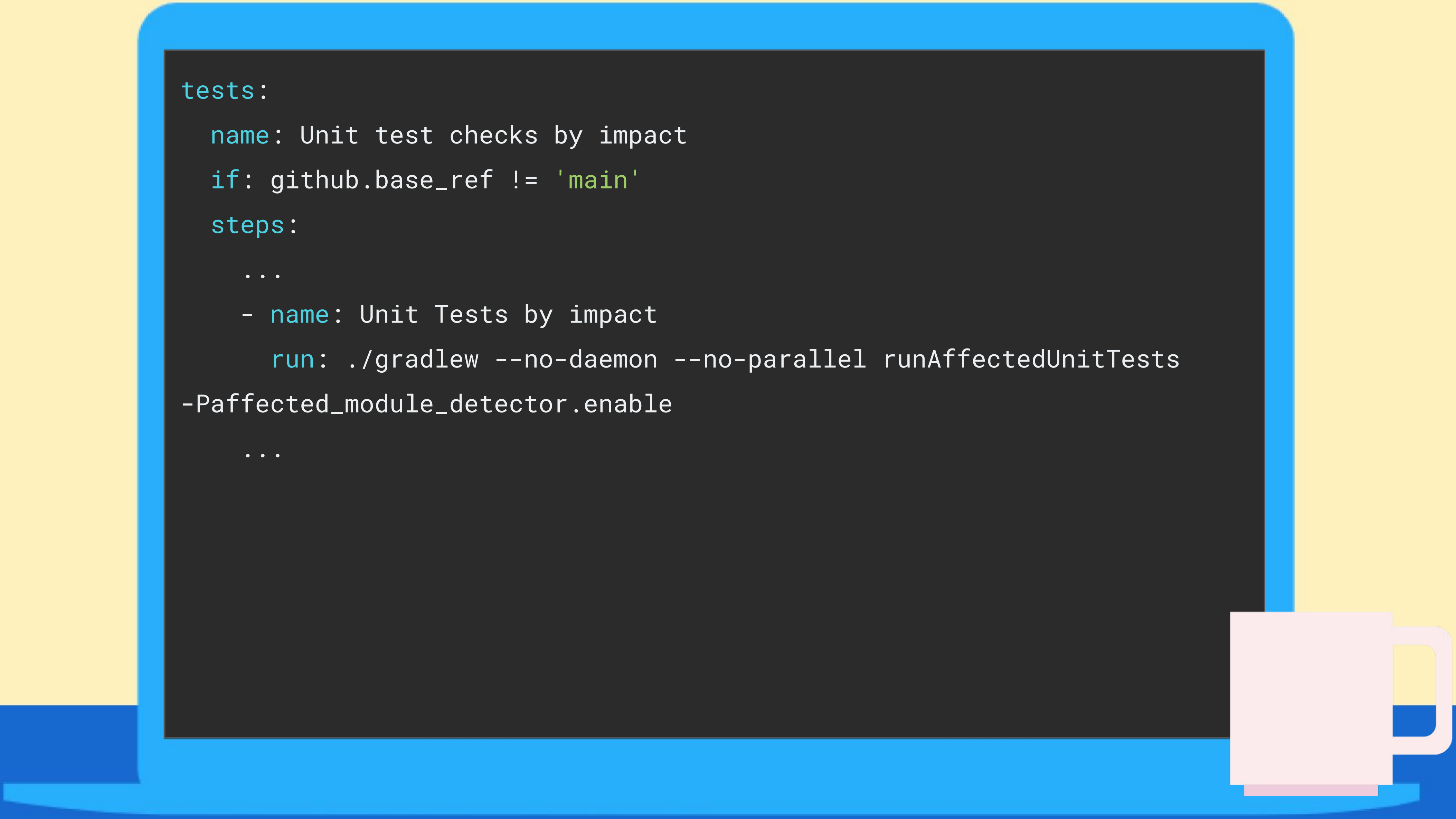


Настроим CI | как стало

Не хотим проверять промежуточные PR в feature-branch

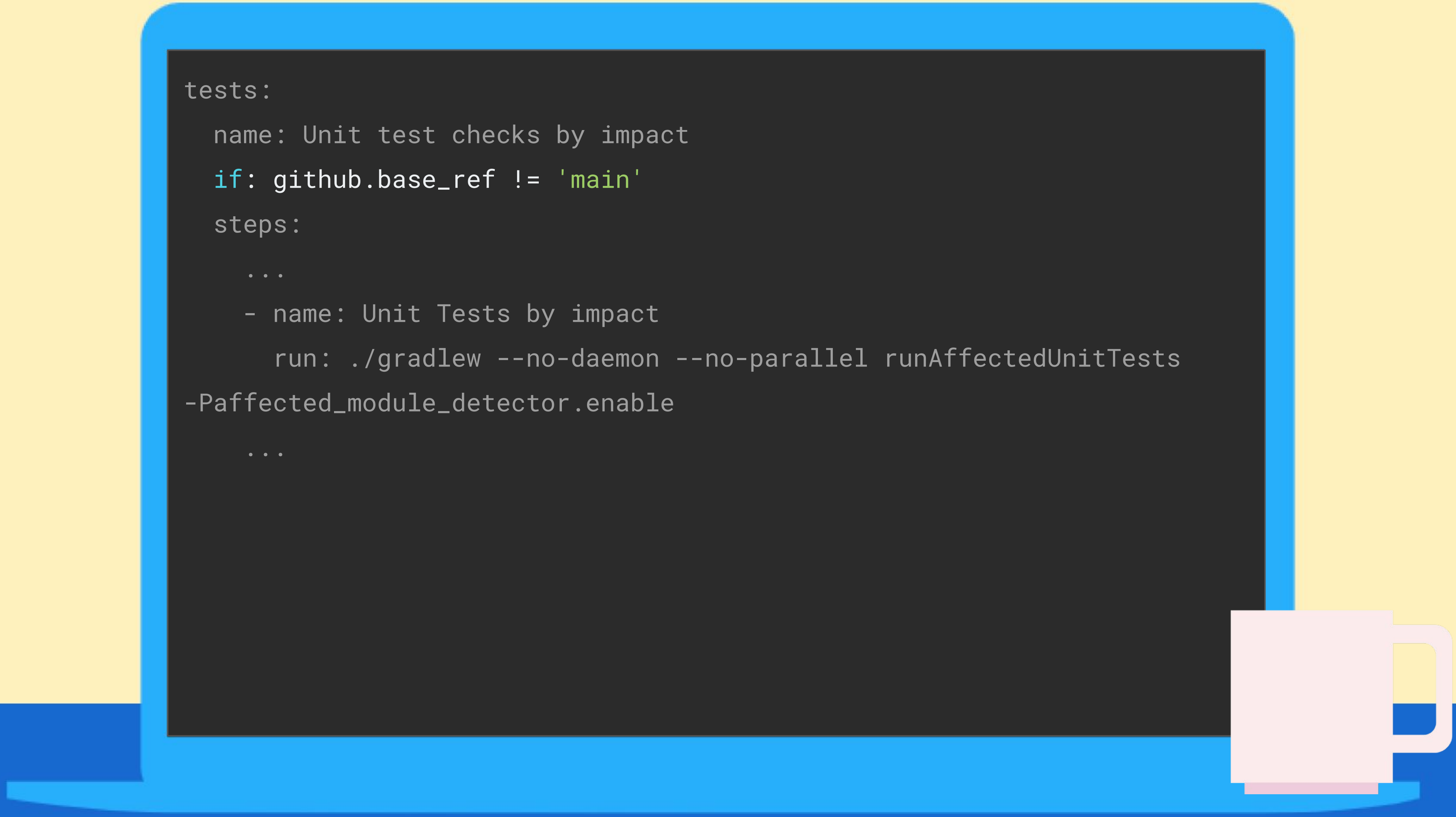


Настроим CI | как стало



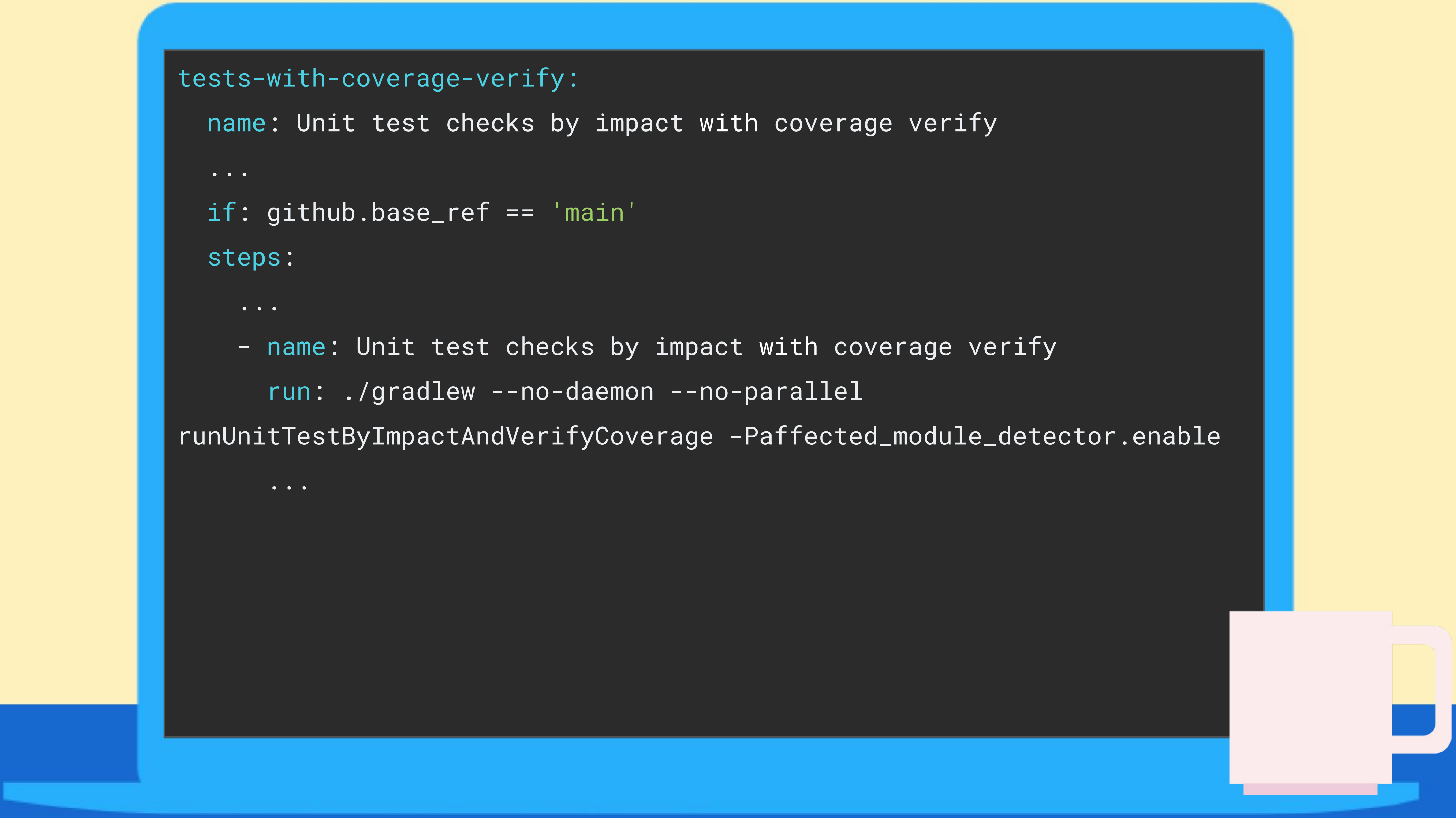
```
tests:
  name: Unit test checks by impact
  if: github.base_ref != 'main'
  steps:
    ...
    - name: Unit Tests by impact
      run: ./gradlew --no-daemon --no-parallel runAffectedUnitTests
-Paffected_module_detector.enable
    ...
```

Настроим CI | как стало



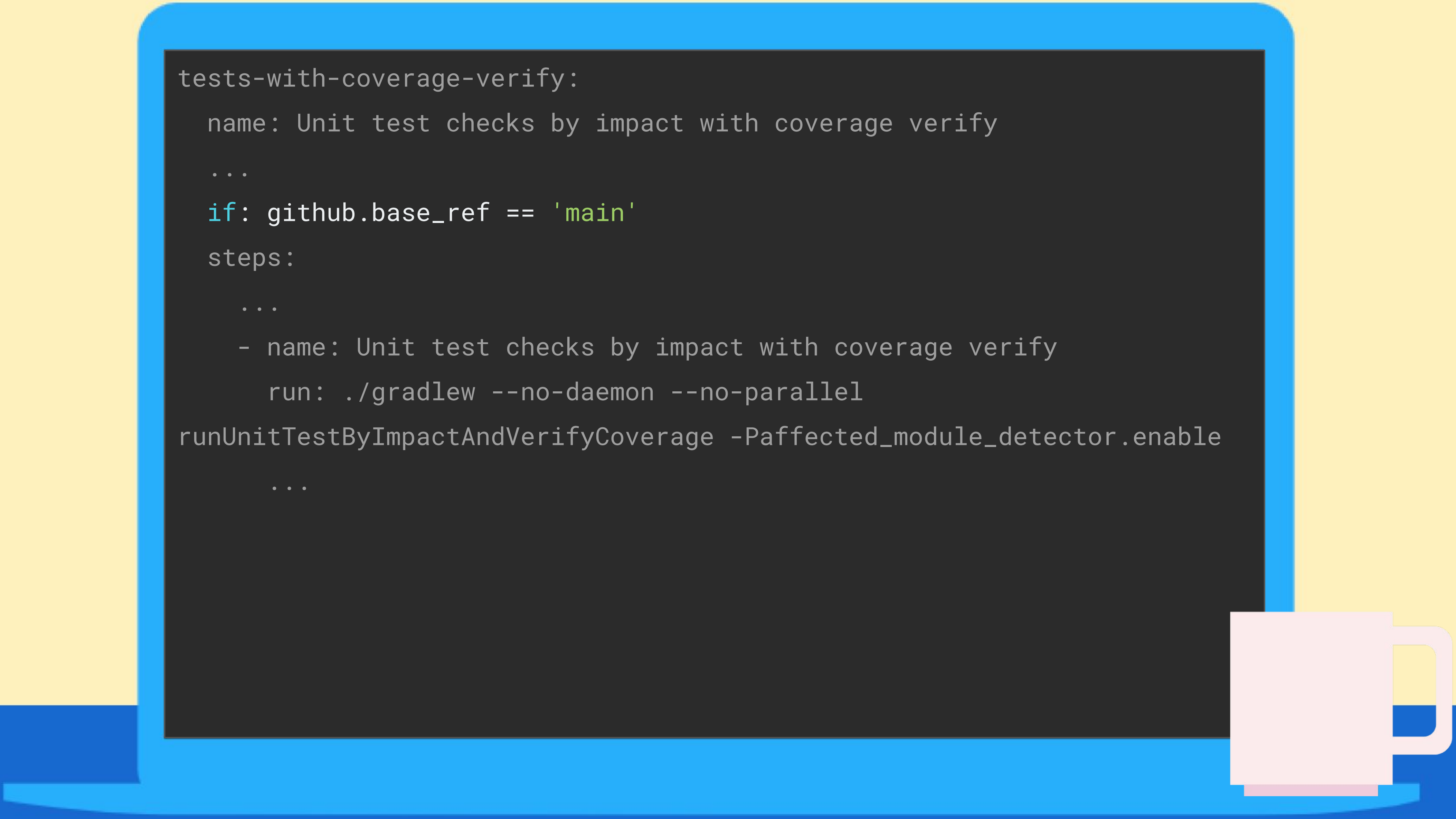
```
tests:
  name: Unit test checks by impact
  if: github.base_ref != 'main'
  steps:
    ...
    - name: Unit Tests by impact
      run: ./gradlew --no-daemon --no-parallel runAffectedUnitTests
-Paffected_module_detector.enable
    ...
```

Настроим CI | как стало



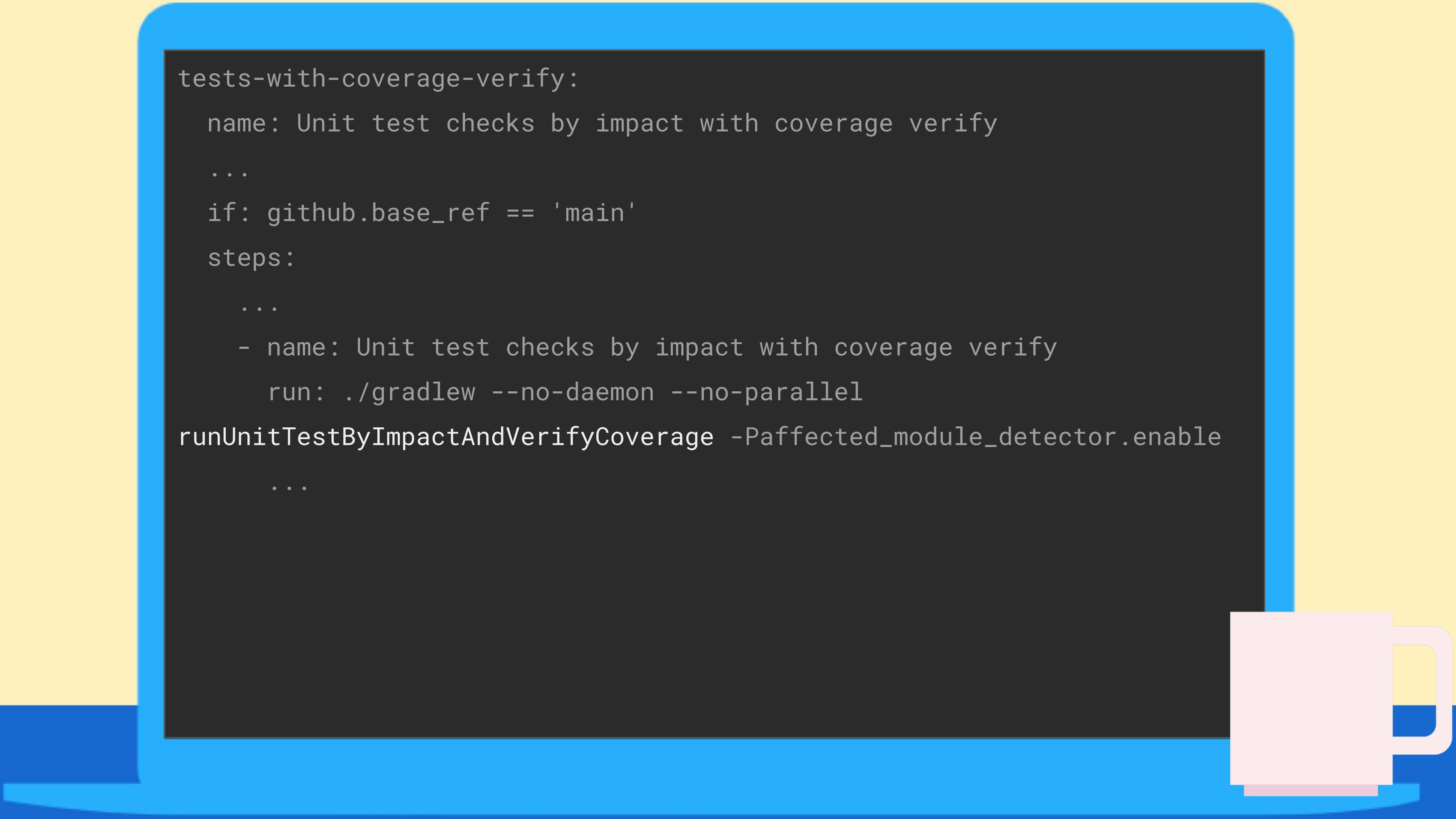
```
tests-with-coverage-verify:
  name: Unit test checks by impact with coverage verify
  ...
  if: github.base_ref == 'main'
  steps:
    ...
    - name: Unit test checks by impact with coverage verify
      run: ./gradlew --no-daemon --no-parallel
runUnitTestByImpactAndVerifyCoverage -Paffected_module_detector.enable
      ...
```

Настроим CI | как стало



```
tests-with-coverage-verify:
  name: Unit test checks by impact with coverage verify
  ...
  if: github.base_ref == 'main'
  steps:
    ...
    - name: Unit test checks by impact with coverage verify
      run: ./gradlew --no-daemon --no-parallel
runUnitTestByImpactAndVerifyCoverage -Paffected_module_detector.enable
      ...
```

Настроим CI | как стало



```
tests-with-coverage-verify:
  name: Unit test checks by impact with coverage verify
  ...
  if: github.base_ref == 'main'
  steps:
    ...
    - name: Unit test checks by impact with coverage verify
      run: ./gradlew --no-daemon --no-parallel
runUnitTestByImpactAndVerifyCoverage -Paffected_module_detector.enable
      ...
```

Требования к фиче

Процент покрытия проекта улучшается каждый PR ✅

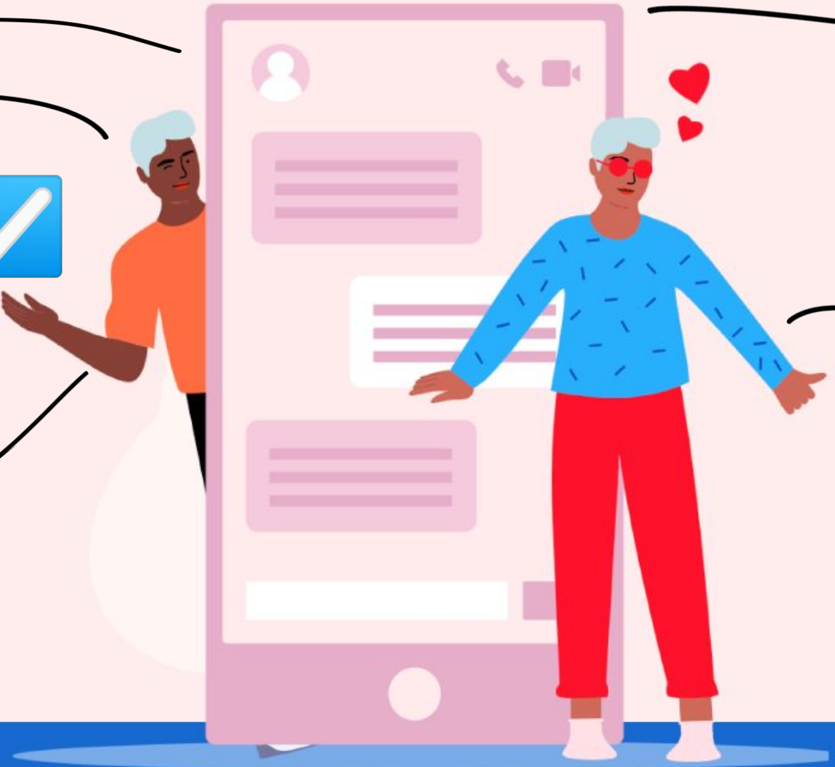
Если модуль новый, вся логика должна быть покрыта тестами ✅

Мы можем посмотреть покрытие ✅

Соблюдение правил контролирует CI, а не человек ✅

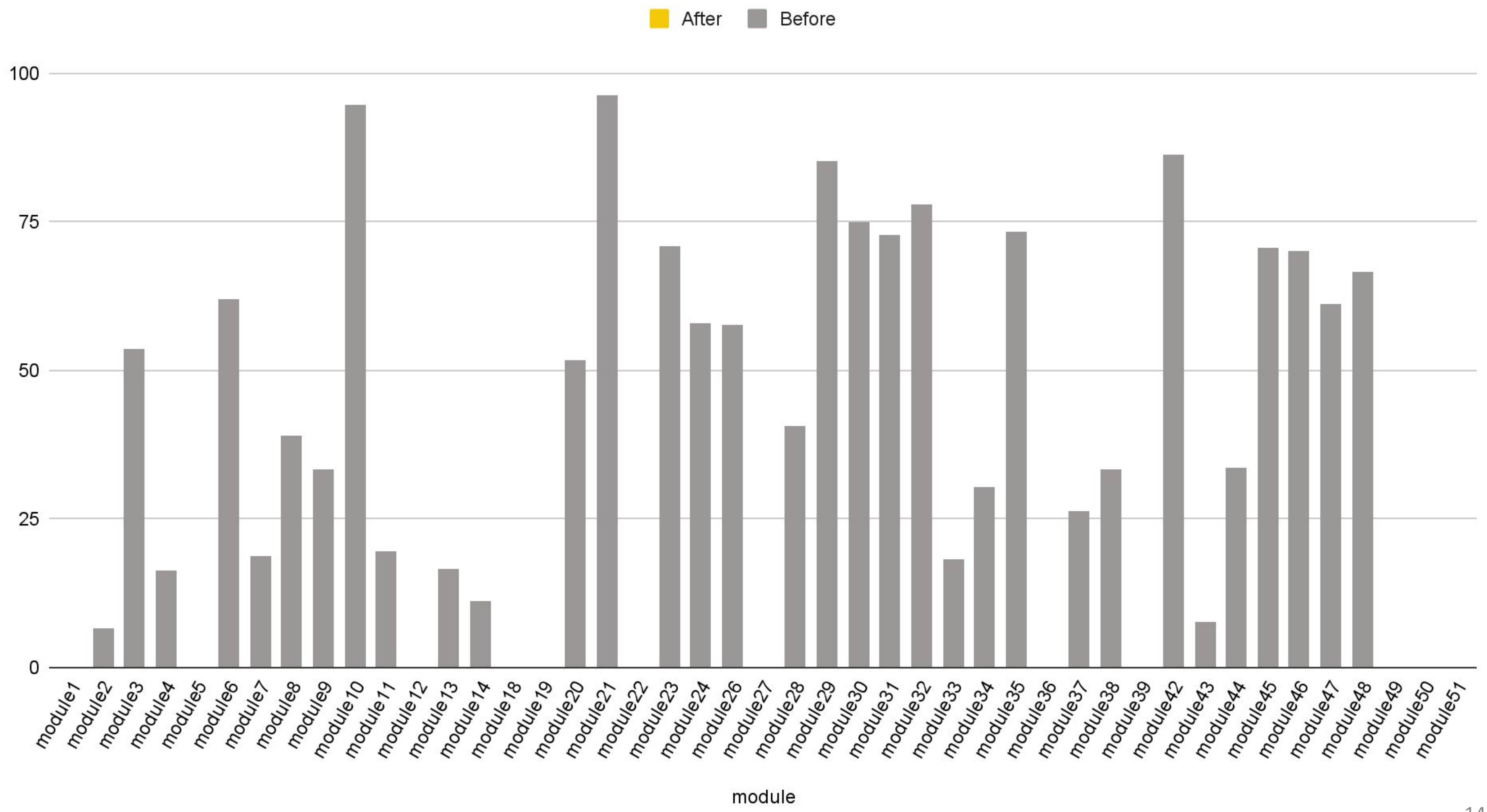
Хотелось бы что бы все это было под impact analysis ✅

Если модуль старый, то вся новая логика должна быть покрыта ✅

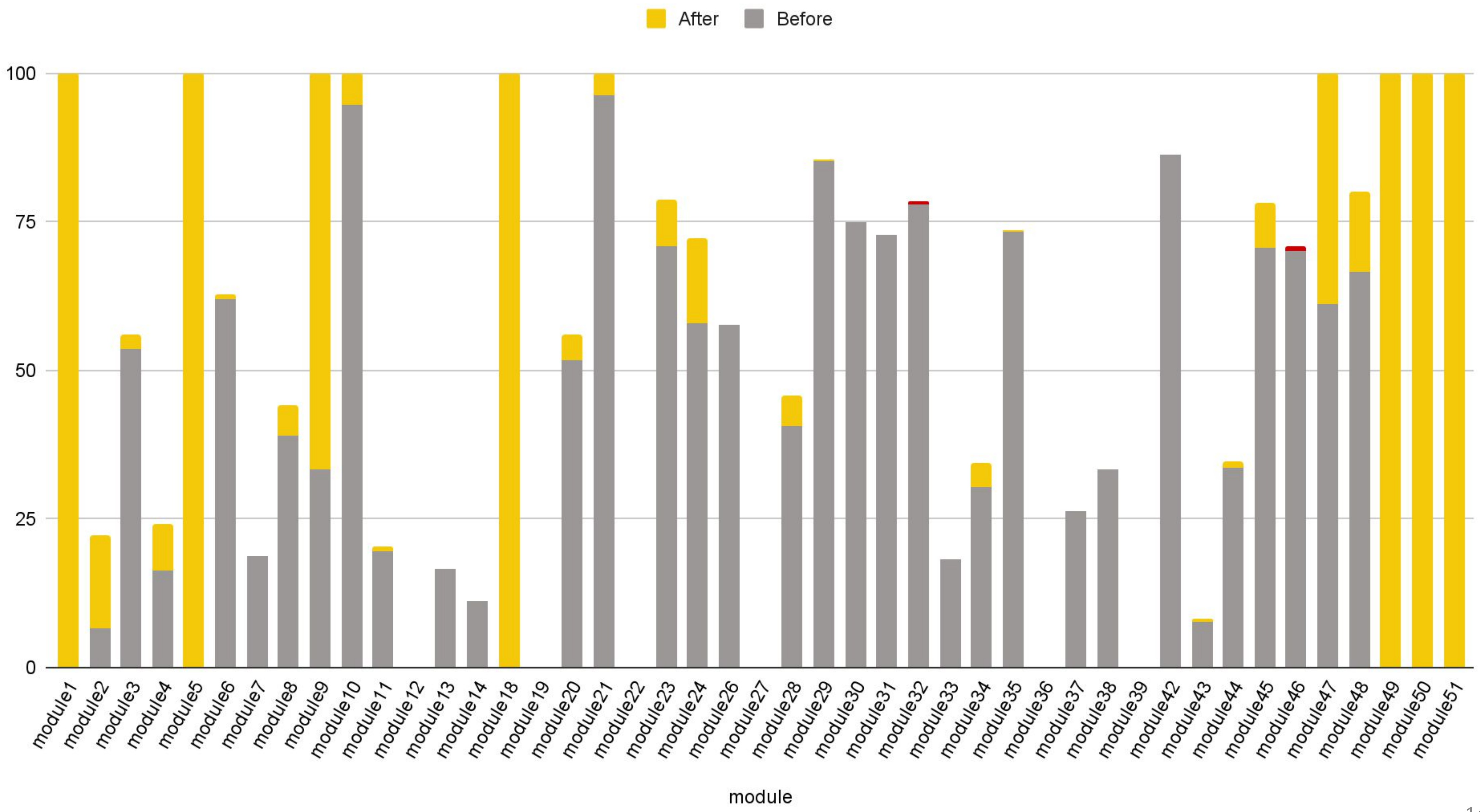


Результаты

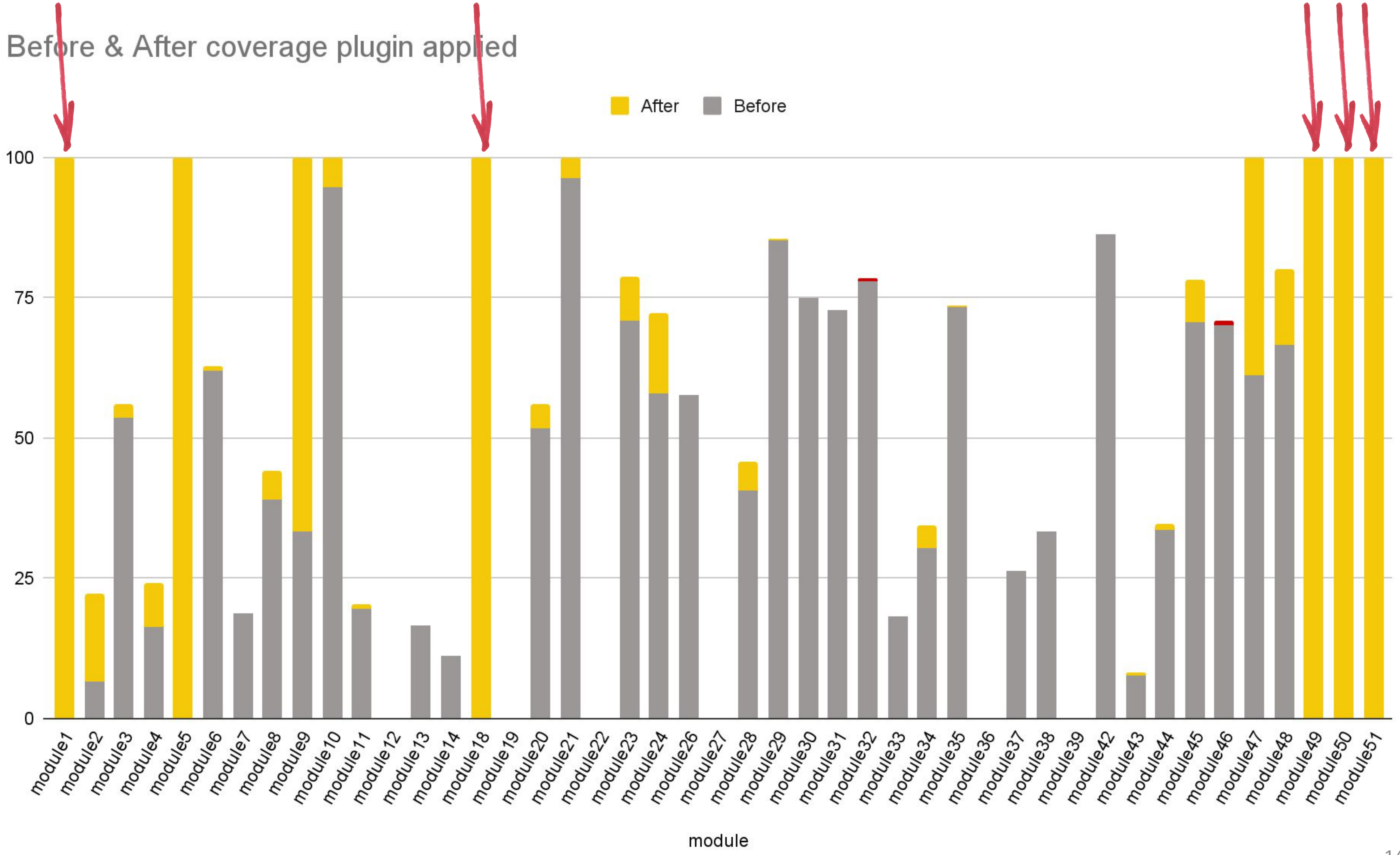
diff и Before



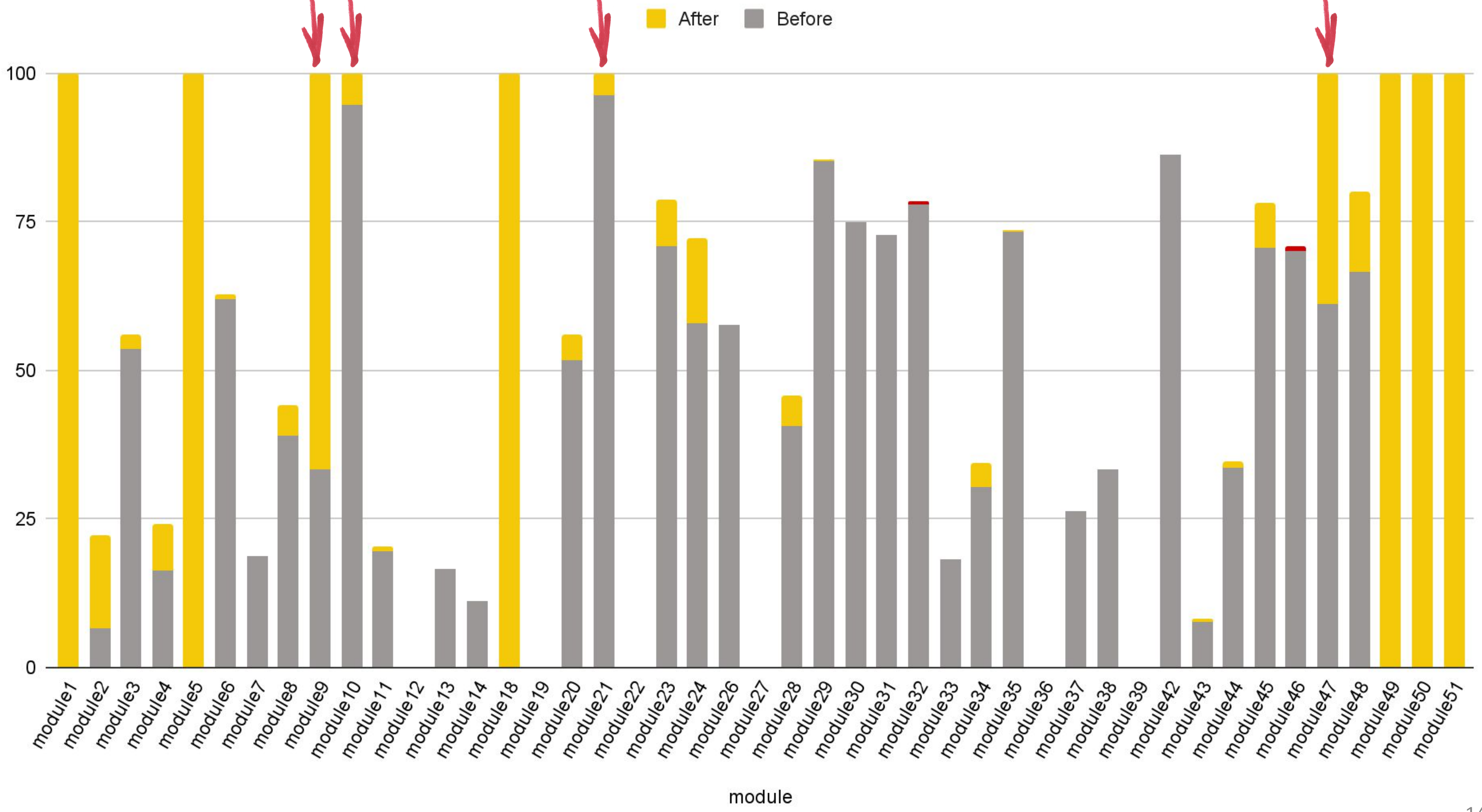
Before & After coverage plugin applied



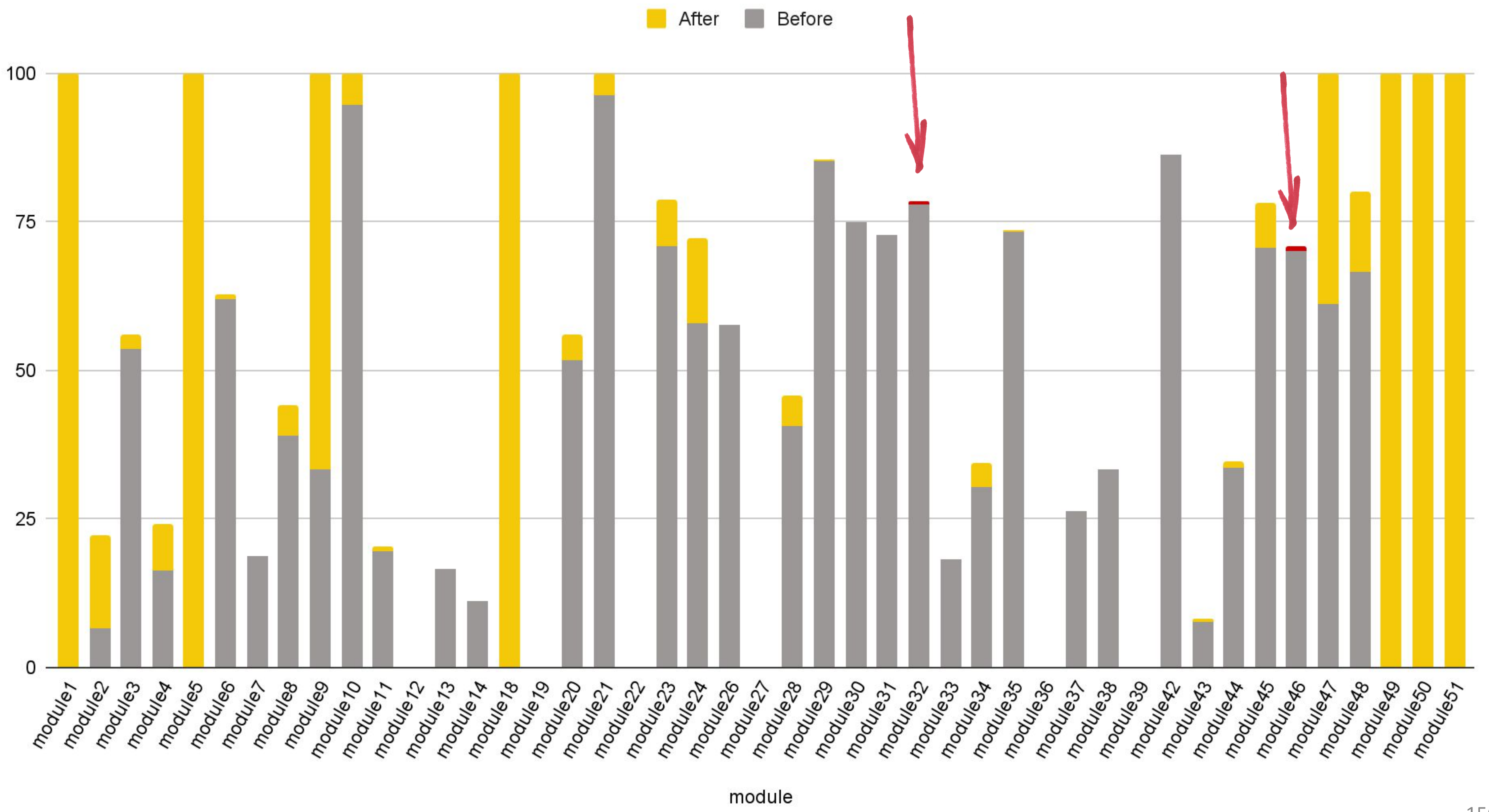
Before & After coverage plugin applied



Before & After coverage plugin applied



Before & After coverage plugin applied



Личные выводы

Роман Аймалетдинов

Q&A

GitHub

Contact:

t.me/raymaletdin

aymaletdinov.job@email.com

linkedin.com/in/raymaletdinov/