

# Stop Debating in Code Reviews Start Enforcing with Lint Rules

Stelios Frantzeskakis - Perry Street Software

droidcon Berlin 2024

“Changes requested”

Hi, I'm Stelios Frantzeskakis

Staff Engineer at Perry Street Software

Publisher of SCRUFF & Jack'd, serving more than 30M members

Working with Android since the release of Android Gingerbread

Passionate about Architecture & Testing

@SteliosFran



*“The ViewModel shouldn't interact directly with the Repository. It's best to communicate with a UseCase instead.”*

*“Didn't we recently agree that UseCases should only expose a single public function?”*

*“Hey, it looks like this Repository is not placed in the correct module.”*

*“Wait I'm confused, are we supposed to access String resources directly from ViewModels?”*

Do not debate architecture in code reviews

No architectural alignment & documentation

We are not designed to remember everything

Lint rules will do that for us



Code styling has been solved long ago

# Architectural lint rules

```

class SampleLintRule : Rule() {
    private val message = "...".

    override val issue: Issue = Issue(
        javaClass.simpleName, Severity.Style,
        message, Debt(days = 0, hours = 3, mins = 0)
    )

    private var ktClass: KtClassOrObject? = null

    override fun visitClassOrObject(classOrObject: KtClassOrObject) {
        if (classOrObject.name?.endsWith("UseCase") == true) {
            ktClass = classOrObject
        }
        super.visitClassOrObject(classOrObject)
    }

    override fun visitClassBody(classBody: KtClassBody) {
        super.visitClassBody(classBody)
        ktClass?.let {
            val hasNonConstantPublicProperty = classBody.properties.any { it.isPublic && !it.isConstant() }
            val publicFunctions = classBody.functions.filter { it.isPublic }.size
            if (hasNonConstantPublicProperty || publicFunctions != 1) {
                report(UseCaseRuleFinding(issue, Entity.from(ktClass as PsiElement), message))
            }
        }
    }

    private class UseCaseRuleFinding(
        override val issue: Issue,
        override val entity: Entity,
        override val message: String,
        override val severity: SeverityLevel = SeverityLevel.ERROR,
        override val metrics: List<Metric> = emptyList(),
        override val references: List<Entity> = emptyList()
    ) : Finding {
        override val id: String = issue.id

        override fun messageOrDescription(): String = message.ifEmpty { issue.description }

        override fun compact(): String = "$id - ${entity.compact()}"
    }
}

```

Konsist: Lint rules as unit tests

Given - When - Then

```
class ViewModelsExtendAndroidxViewModel : BehaviorSpec() {  
    init {  
        Given("All classes in production code") {  
            val classes = Konsist.scopeFromProduction().classes()  
  
            When("There is a ViewModel") {  
                val viewModels = classes.withNameEndingWith("ViewModel")  
  
                Then("It extends the Android Jetpack ViewModel") {  
                    viewModels.assertTrue {  
                        it.hasParentWithName("ViewModel")  
                    }  
                }  
            }  
        }  
    }  
}
```

```
class ViewModelsExtendAndroidxViewModel : BehaviorSpec() {  
    init {  
        Given("All classes in production code") {  
            val classes = Konsist.scopeFromProduction().classes()  
  
            When("There is a ViewModel") {  
                val viewModels = classes.withNameEndingWith("ViewModel")  
  
                Then("It extends the Android Jetpack ViewModel") {  
                    viewModels.assertTrue {  
                        it.hasParentOf(ViewModel::class)  
                    }  
                }  
            }  
        }  
    }  
}
```

Assert 'invokeSuspend' was violated (2 times).

Invalid declarations:

LoginViewModel.kt:3:1 (LoginViewModel ClassDeclaration)

ChatViewModel.kt:3:1 (ChatViewModel ClassDeclaration)



```
class ViewModelsExtendAndroidxViewModel : BehaviorSpec() {

    init {
        Given("All classes in production code") {
            val classes = Konsist.scopeFromProduction().classes().withoutName(*BASELINE)

            When("There is a ViewModel") {
                val viewModels = classes.withNameEndingWith("ViewModel")

                Then("It extends the Android Jetpack ViewModel") {
                    viewModels.assertTrue {
                        it.hasParentOf(ViewModel::class)
                    }
                }
            }
        }
    }

    private companion object {
        private val BASELINE = arrayOf("LoginViewModel", "ChatViewModel")
    }
}
```

```
class ViewModelsExtendAndroidxViewModel : BehaviorSpec() {

    init {
        Given("All classes in production code") {
            val classes = Konsist.scopeFromProduction().classes().withoutName(*BASELINE)

            When("There is a ViewModel") {
                val viewModels = classes.withNameEndingWith("ViewModel")

                Then("It extends the Android Jetpack ViewModel") {
                    viewModels.assertTrue(additionalMessage = MESSAGE) {
                        it.hasParentOf(ViewModel::class)
                    }
                }
            }
        }
    }

    private companion object {
        private val MESSAGE = """
            Always extend the Android Jetpack ViewModel when creating a new ViewModel,
            to take advantage of its lifecycle awareness and other features it provides.
        """.trimIndent()

        private val BASELINE = arrayOf("LoginViewModel", "ChatViewModel")
    }
}
```

Enforcing the architecture with lint rules

```
class ViewModelsDoNotDependOnRepositories : BehaviorSpec() {  
    init {  
        Given("All classes in production code") {  
            val classes = Konsist.scopeFromProduction().classes()  
  
            When("There is a ViewModel") {  
                val viewModels = classes.withNameEndingWith("ViewModel")  
  
                Then("No Repository is listed in the constructor parameters") {  
                    viewModels.constructors.parameters.assertFalse {  
                        it.type.name.endsWith("Repository")  
                    }  
                }  
            }  
        }  
    }  
}
```

```
class ViewModelsDoNotAccessResources : BehaviorSpec() {  
    init {  
        Given("All files in production code") {  
            val files = Konsist.scopeFromProduction().files  
  
            When("There is a ViewModel") {  
                val viewModels = files.withNameEndingWith("ViewModel")  
  
                Then("It does not access resources") {  
                    viewModels.imports.assertFalse {  
                        it.hasNameStartingWith("com.steliosf.konsist.samples.R")  
                    }  
                }  
            }  
        }  
    }  
}
```

```
class UseCasesExposeOnePublicFunction : BehaviorSpec() {  
    init {  
        Given("All classes in production code") {  
            val classes = Konsist.scopeFromProduction().classes()  
  
            When("There is a UseCase") {  
                val useCases = classes.withNameEndingWith("UseCase")  
  
                Then("It exposes a single public function") {  
                    useCases.assertFalse {  
                        it.functions().withPublicOrDefaultModifier().size > 1  
                    }  
                }  
  
                Then("It does not expose any properties") {  
                    useCases.properties().assertTrue {  
                        it.hasPrivateModifier  
                    }  
                }  
            }  
        }  
    }  
}
```

```
class DomainLayerDoesNotDependOnDTOs : BehaviorSpec() {  
    init {  
        Given("All files in the domain module") {  
            val domainModuleFiles = Konsist.scopeFromDirectory("domain").files  
  
            Then("They do not access DTOs") {  
                domainModuleFiles.imports.assertFalse {  
                    it.hasNameStartingWith("com.steliosf.dto")  
                }  
            }  
        }  
    }  
}
```

```
class RepositoriesLocatedInRepositoriesModule : BehaviorSpec() {  
    init {  
        Given("All classes in production code") {  
            val classes = Konsist.scopeFromProduction().classes()  
  
            When("There is a Repository") {  
                val repositories = classes.withNameEndingWith("Repository")  
  
                Then("It is located in the repositories module") {  
                    repositories.assertTrue {  
                        it.resideInModule("data/repository")  
                    }  
                }  
            }  
        }  
    }  
}
```



Preventing bugs at an earlier stage with lint rules

```
@POST(PATH)  
fun postProfile(@Field("id") id: Long, @Field("name") name: String)
```

```
@POST(PATH)  
fun postProfile(@Field("id") id: Long, @Field("name") name: String)
```

```
java.lang.IllegalArgumentException:  
@Field parameters can only be used with form encoding. (parameter #1) for method postProfile
```

```
@FormUrlEncoded
@POST(PATH)
fun postProfile(@Field("id") id: Long, @Field("name") name: String)
```

```
java.lang.IllegalArgumentException:
@Field parameters can only be used with form encoding. (parameter #1) for method postProfile
```

```
class RetrofitFieldParamsUseFormUrlEncoded : BehaviorSpec() {  
    init {  
        Given("All functions in production code") {  
            val functions = Konsist.scopeFromProduction().functions()  
  
            When("There is a function with the @POST annotation") {  
                val functionsWithPost = functions.withAnnotationNamed("POST")  
  
                And("It has at least one @Field parameter") {  
                    val functionsWithFieldParams = functionsWithPost.withParameter {  
                        it.hasAnnotationWithName("Field")  
                    }  
  
                    Then("It has the @FormUrlEncoded annotation") {  
                        functionsWithFieldParams.assertTrue {  
                            it.hasAnnotationWithName("FormUrlEncoded")  
                        }  
                    }  
                }  
            }  
        }  
    }  
}
```

```
class ViewModelsDependencyInjection : BehaviorSpec() {  
    init {  
        Given("All classes in production code") {  
            val classes = Konsist.scopeFromProduction().classes()  
  
            When("There is a ViewModel") {  
                val viewModels = classes.withNameEndingWith("ViewModel")  
  
                Then("It has the necessary dependency injection annotation") {  
                    viewModels.assertTrue {  
                        it.hasAnnotationWithName("HiltViewModel", "KoinViewModel")  
                    }  
                }  
            }  
        }  
    }  
}
```

# Project structure with Konsist

Android ▾



>  app

>  data

>  design-system


>  domain

▾  **konsist**

 main

▾  test


▾  tests


▾  com.steliosf.konsist.rules

 DesignSystemDoesNotDependOnDomain


 DomainLayerDoesNotDependOnDTOs

 RepositoriesLocatedInRepositoriesModule


 RetrofitFieldParamsUseFormUrlEncoded


 UseCasesExposeOnePublicFunction

 ViewModelsDependencyInjection

 ViewModelsDoNotAccessResources

 ViewModelsDoNotDependOnRepositories

 ViewModelsExtendAndroidxViewModel

>  Gradle Scripts



Running lint rules in CI/CD

```
name: Run Konsist lint rules
on:
  pull_request:
    types: [opened, synchronize]

jobs:
  run-lint-rules:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - uses: actions/setup-java@v4
        with:
          distribution: temurin
          java-version: 17

      - name: Setup Gradle
        uses: gradle/actions/setup-gradle@v3

      - name: Run Konsist lint rules Gradle task
        run: ./gradlew :konsist:test --rerun-tasks
```

# Benefits of custom lint rules

Enforcing the architecture

Documenting the architecture

Bugs will be caught earlier

Consistency will make you go faster

Onboarding new engineers will be smoother



# Resources



Slide deck • <https://www.steliosf.com/assets/talks/enforce-with-lint-rules.pdf>

Konsist lint rules samples • <https://github.com/steliosfran/konsist-lint-rules-samples>

Konsist official repository • <https://github.com/LemonAppDev/konsist>

Konsist docs • <https://docs.konsist.lemonappdev.com>

#konsist slack channel • <https://kotlinlang.slack.com/archives/C05QG9FD6KS>

Put Your Tests on a Diet droidcon talk • <https://www.steliosf.com/talks>

@SteliosFran • <https://x.com/SteliosFran> • <https://www.linkedin.com/in/SteliosFran>

Join the Konsist community

Align - Enforce - Automate

You better write some lint rules today,  
To automate your problems away.  
No debates, no code review fights,  
That'll make you sleep much better at nights.