

Isolated Development



Ralf Wondratschek

Principal Engineer

Isolated Development



Ralf Wondratschek

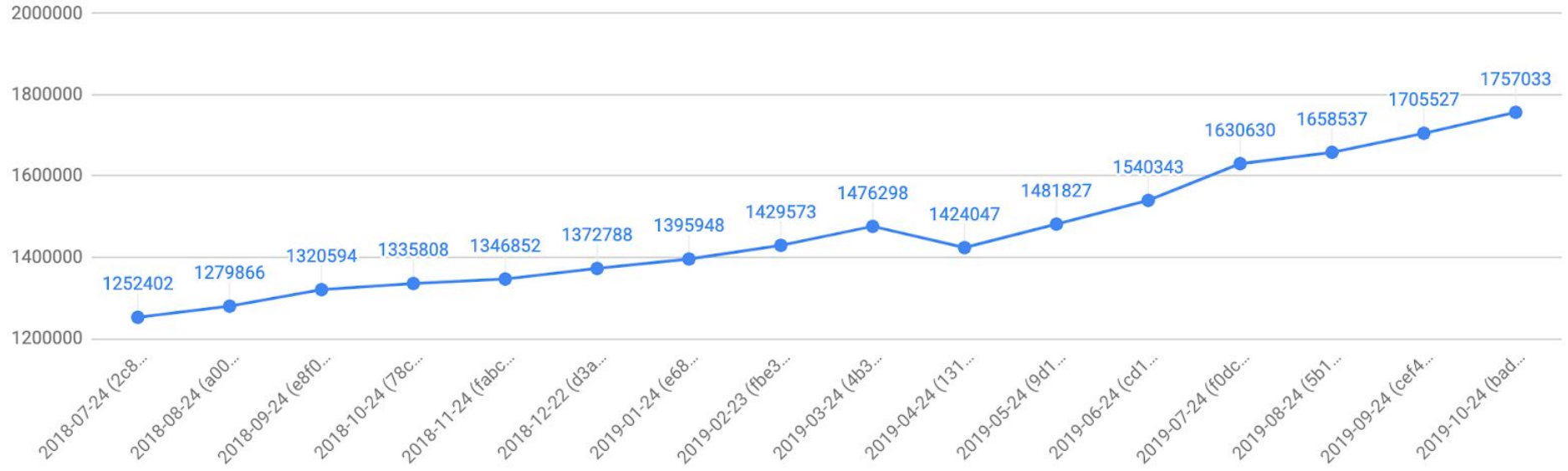
@vRallev

amazon

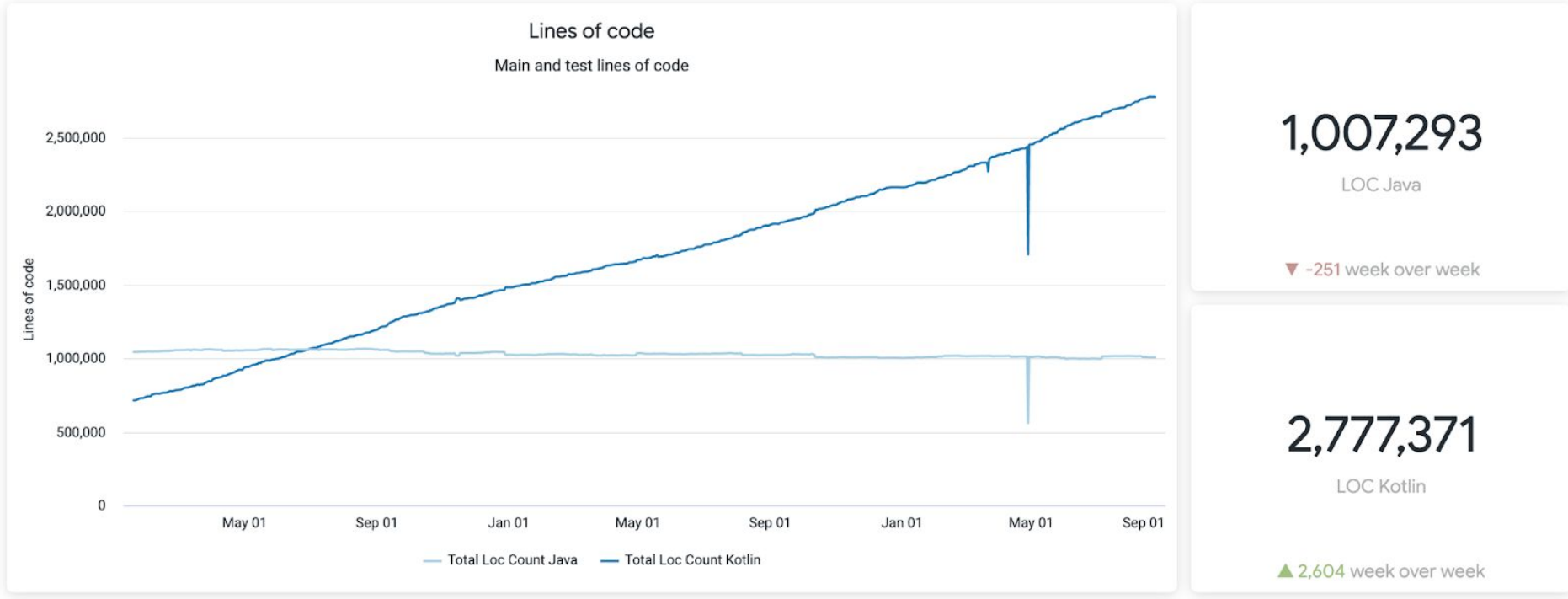
Challenges in 2018

Challenges in 2018: lines of code growth

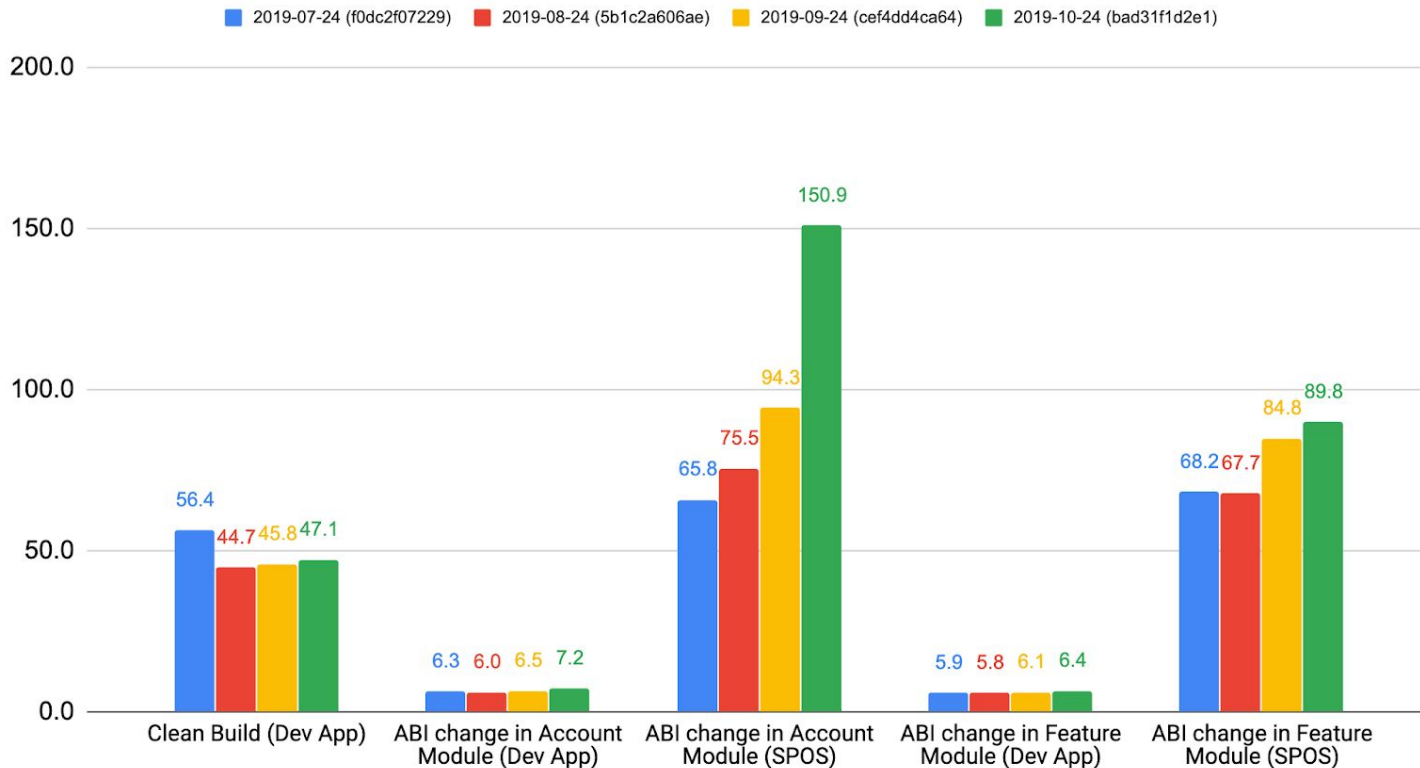
Lines of Code



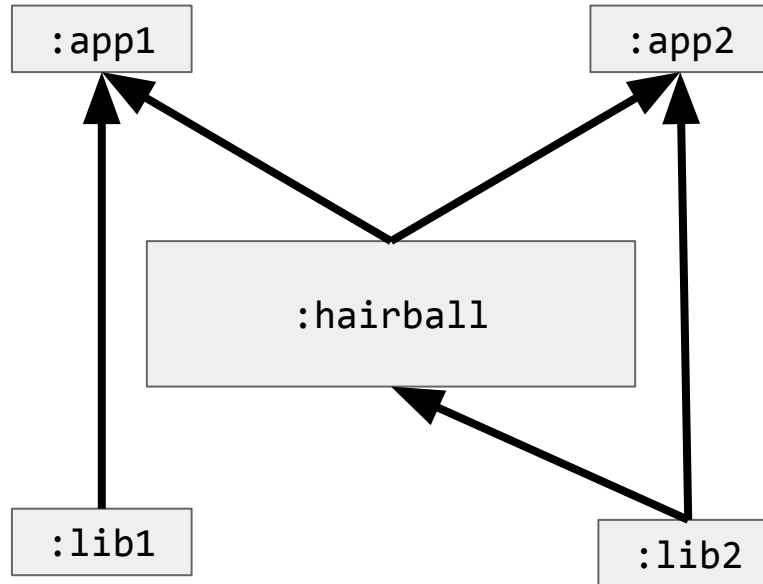
Challenges in 2018: lines of code growth



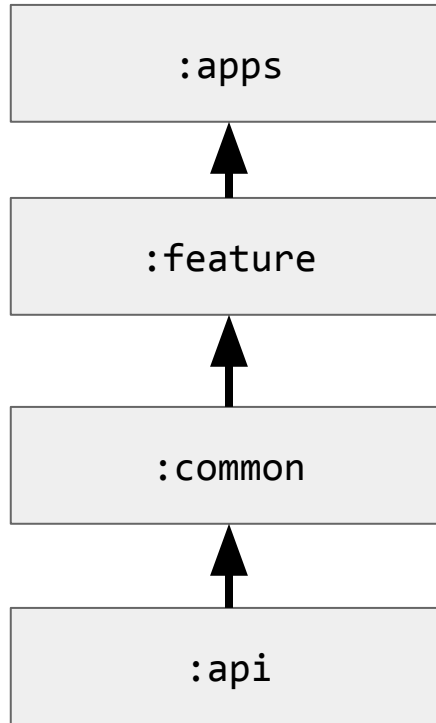
Challenges in 2018: build times are growing



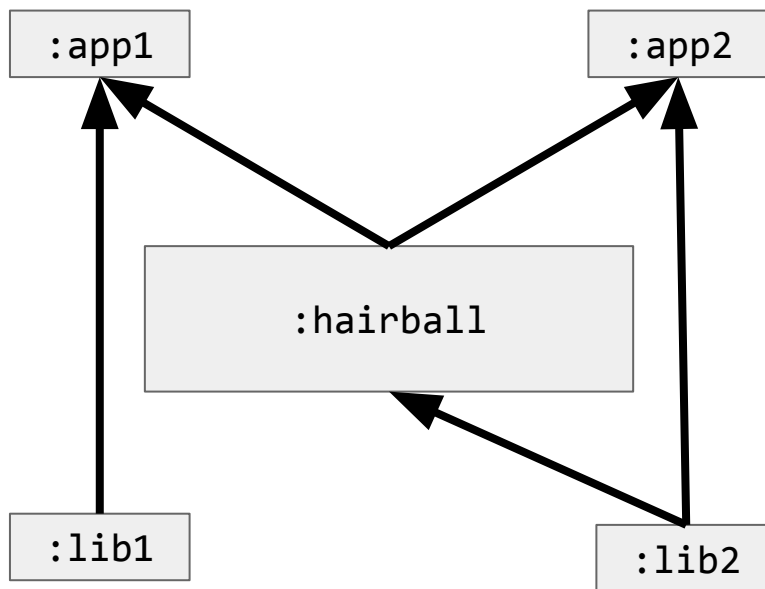
Challenges in 2018: module structure



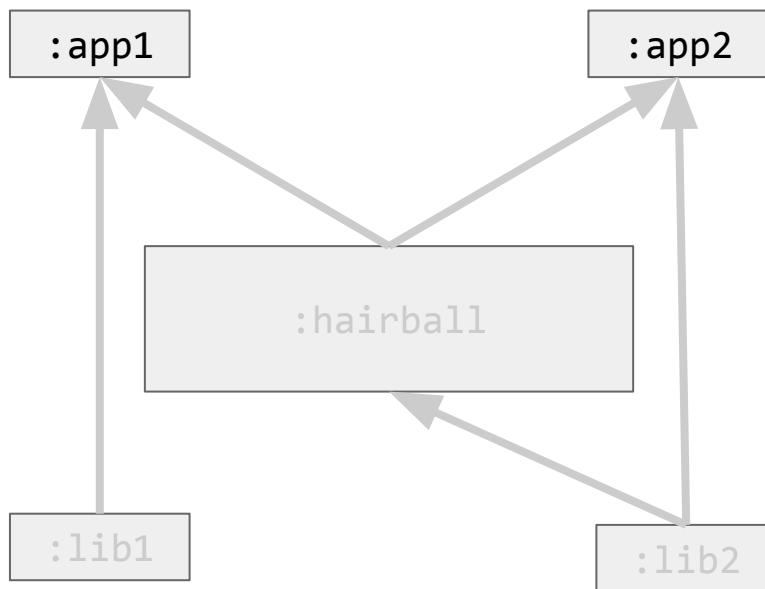
Challenges in 2018: module structure

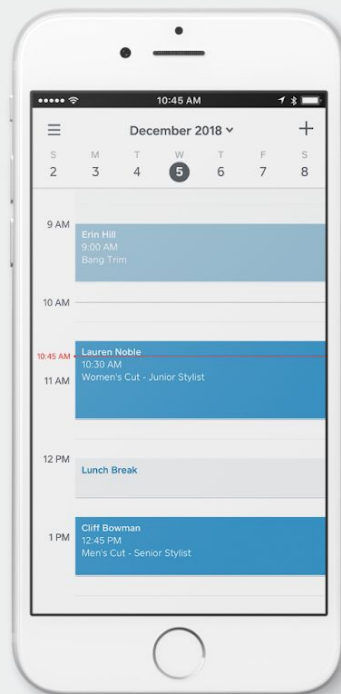


Challenges in 2018: multiple apps with different flavors



Challenges in 2018: multiple apps with different flavors









Android at Scale @Square



Ralf Wondratschek

@vRallev



November 25, 2019



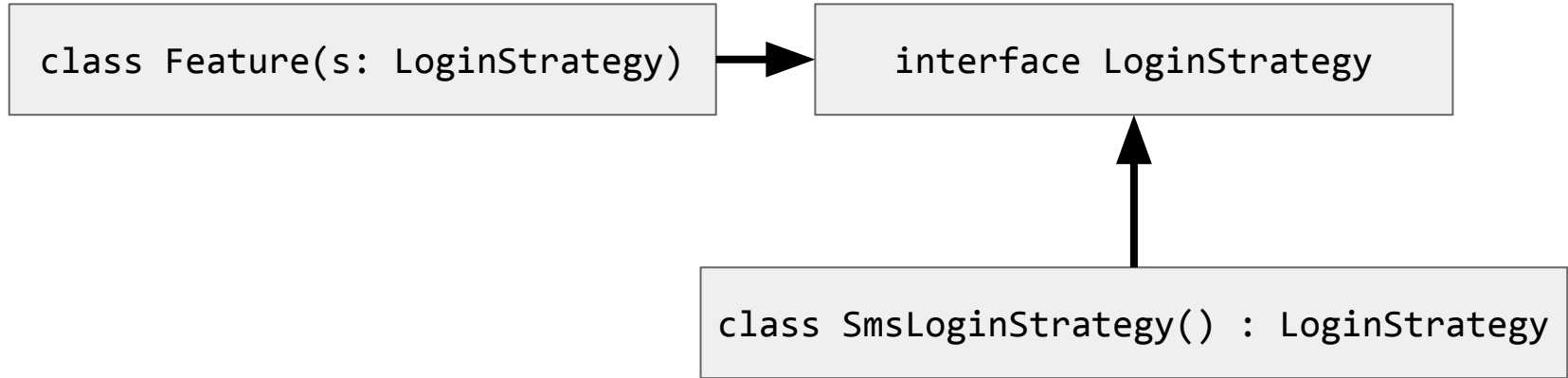
.droidcon
SAN FRANCISCO
#DCSF19



<https://bit.ly/3D8SsUI>

Dependency inversion

Dependency inversion



Dependency inversion

```
class SmsLoginStrategy @Inject constructor(  
    private val helper: Helper,  
    private val helper: Helper1,  
    private val helper: Helper2,  
    private val helper: Helper3,  
    private val helper: Helper4,  
    private val helper: Helper5,  
    private val helper: Helper6,  
    private val helper: Helper7,  
    private val helper: Helper8  
) : LoginStrategy
```

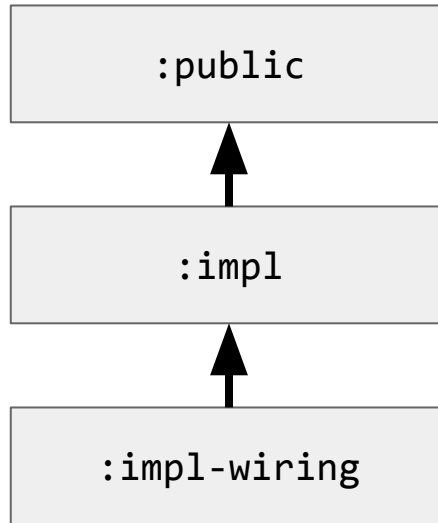
Dependency inversion

```
dependencies {  
  api project(':helper1')  
  api project(':helper2')  
  api project(':helper3')  
  api project(':helper4')  
  api project(':helper5')  
  api project(':helper6')  
  api project(':helper7')  
  api project(':helper8')  
}
```

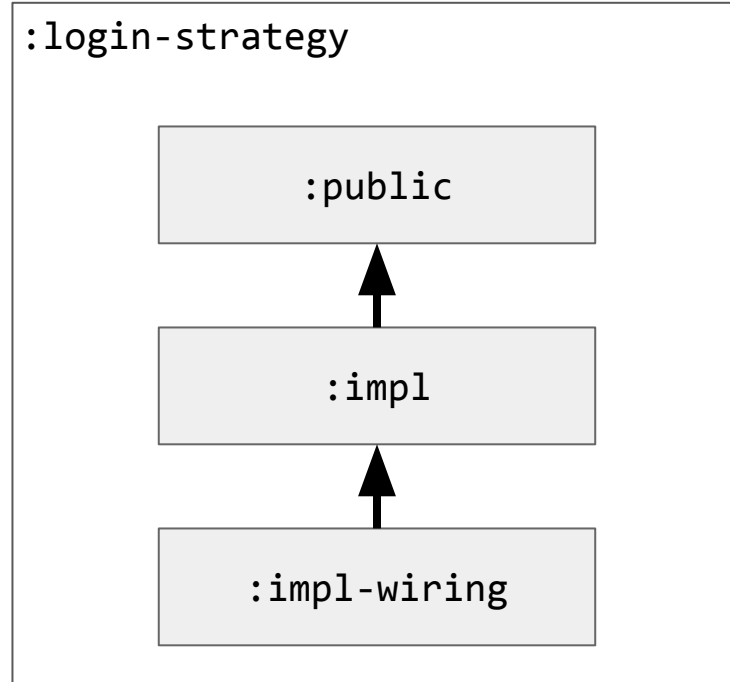
Dependency inversion

```
dependencies {  
  api project(':login-strategy')  
}
```

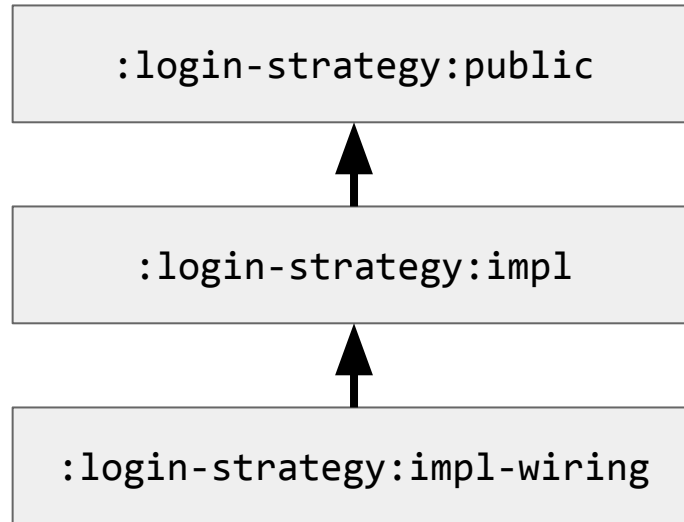
Module Structure



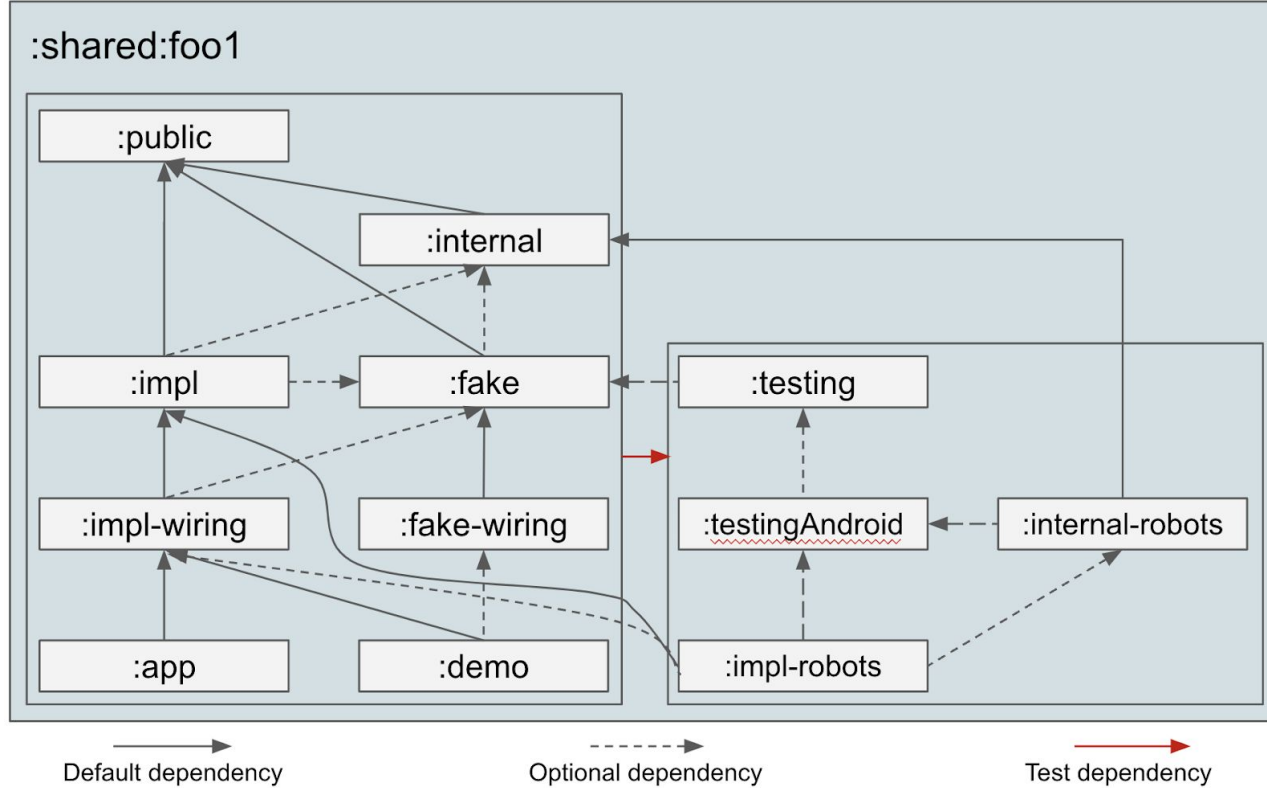
Module Structure



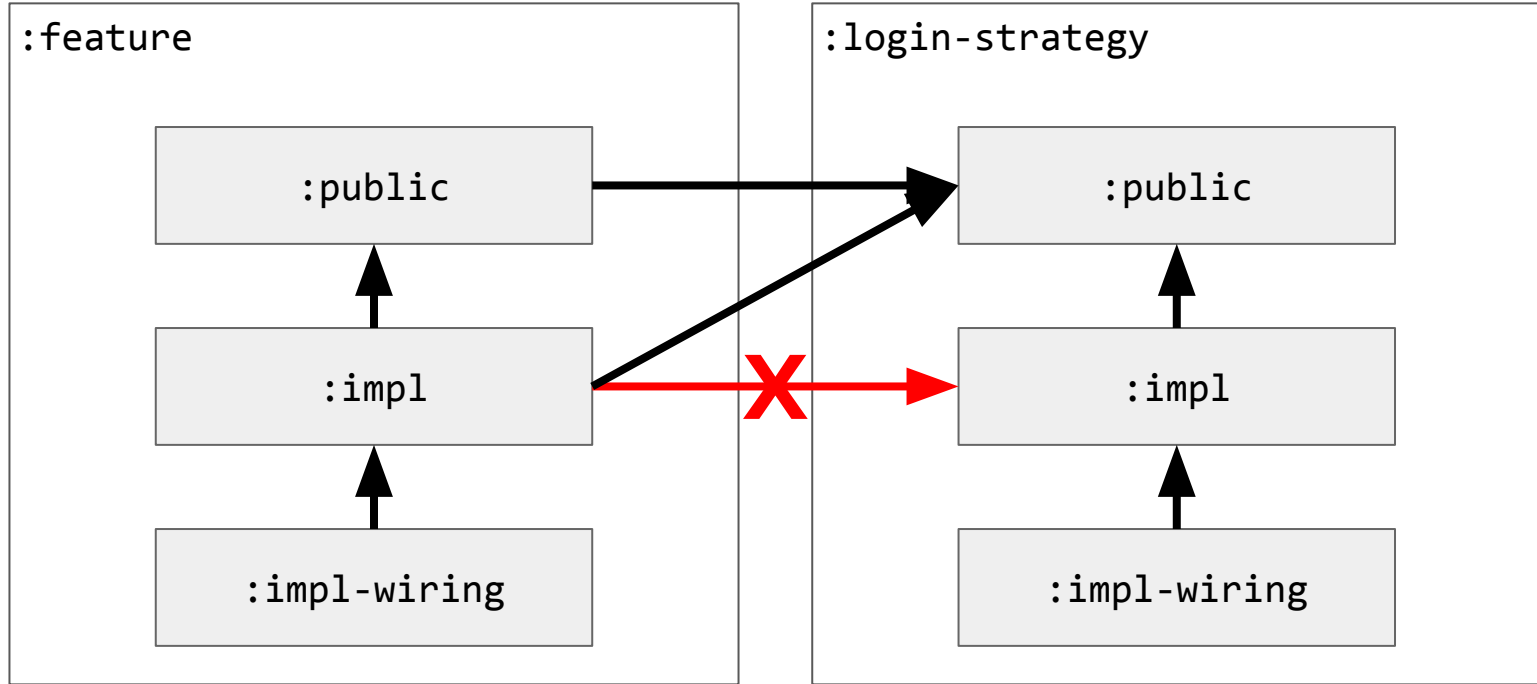
Module Structure



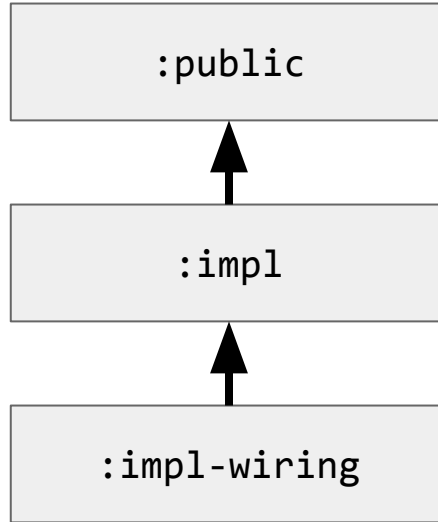
Module Structure



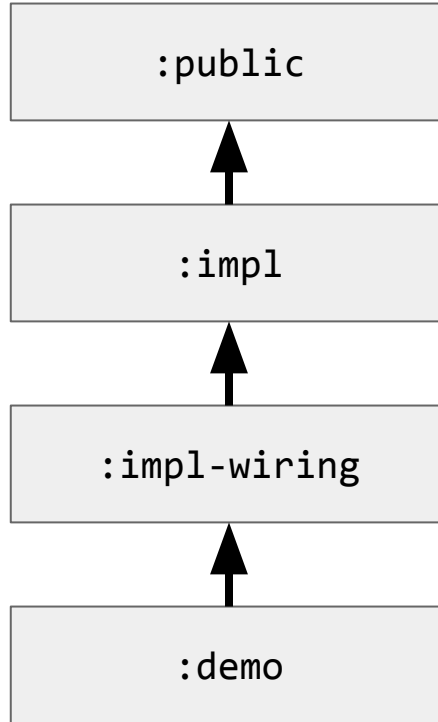
Module Structure



Development Apps



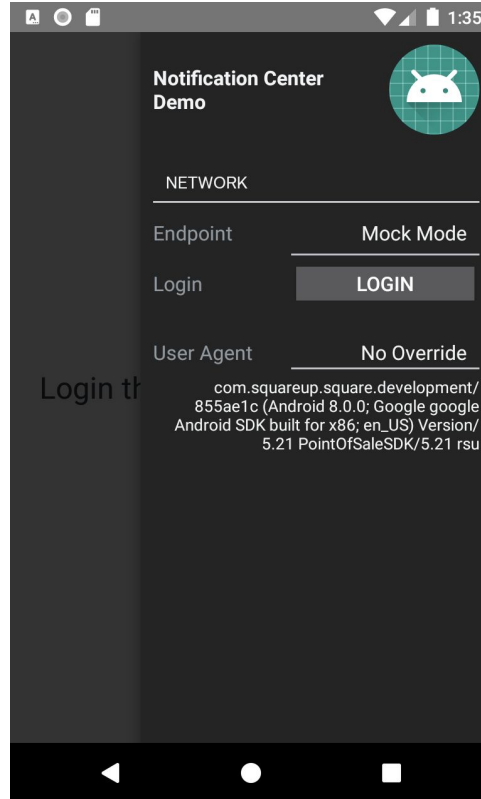
Development Apps



Development Apps



Login through the debug drawer



× Notifications

Important

W Your account is frozen. 

W Deposit cannot be processed. 

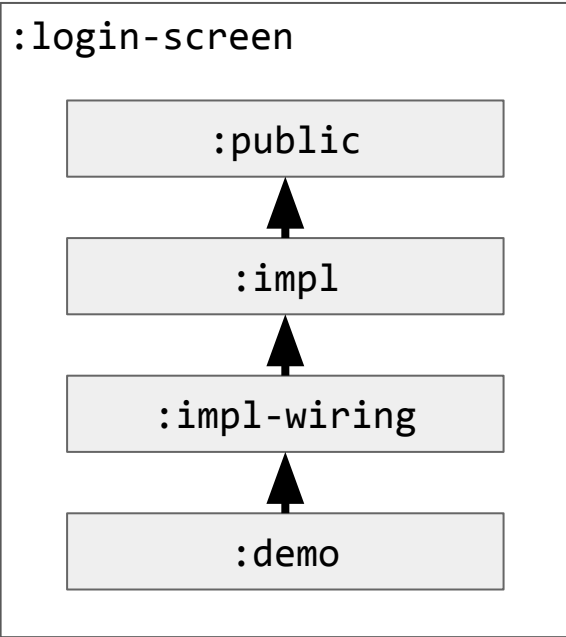
W Welcome to Square! Setup your account now. 



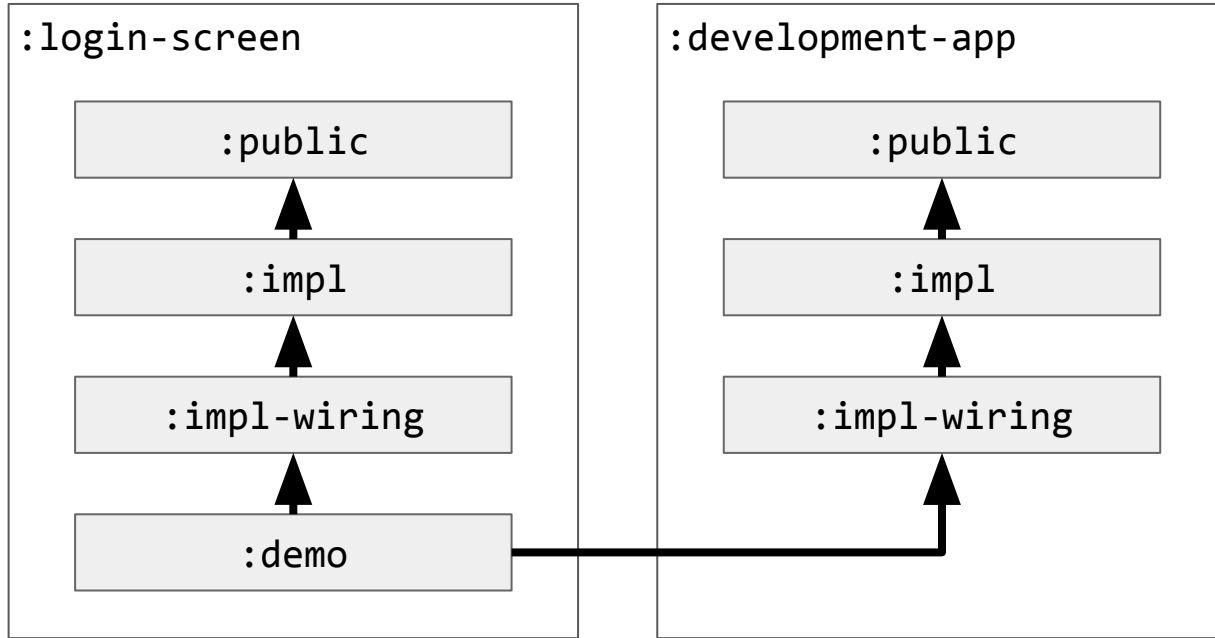
Development Apps

- Install a feature with an isolated application on the device
 - Launch the feature directly
 - Shorter build times
- Easy way to experiment and merge code

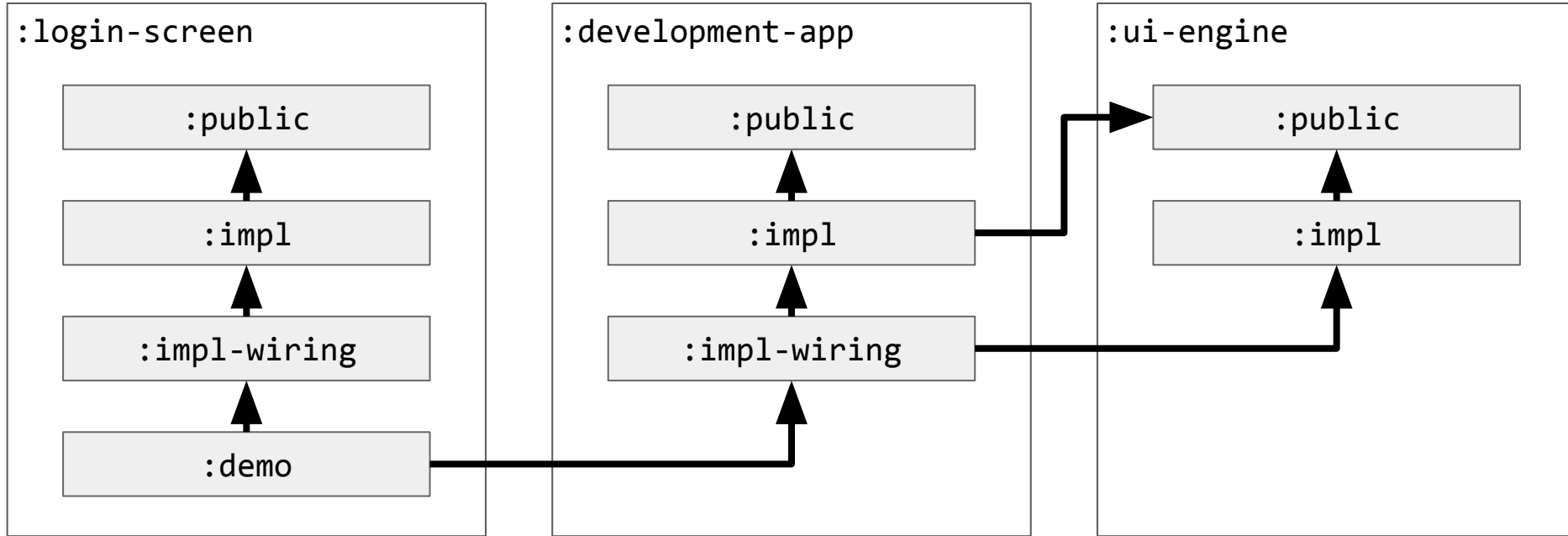
Development Apps



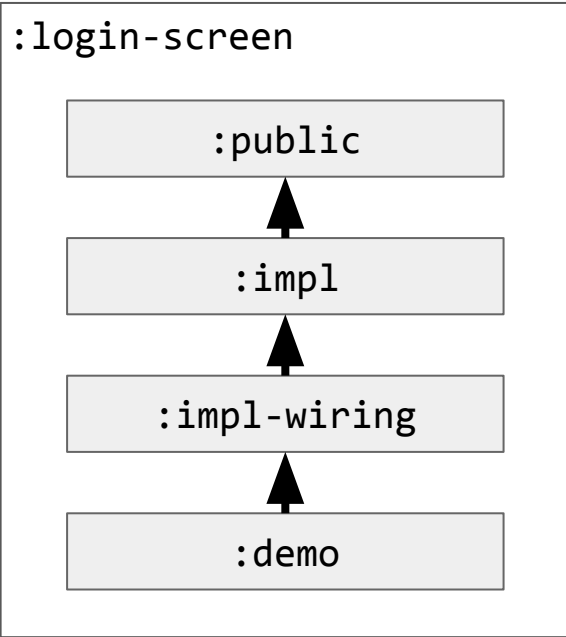
Development Apps



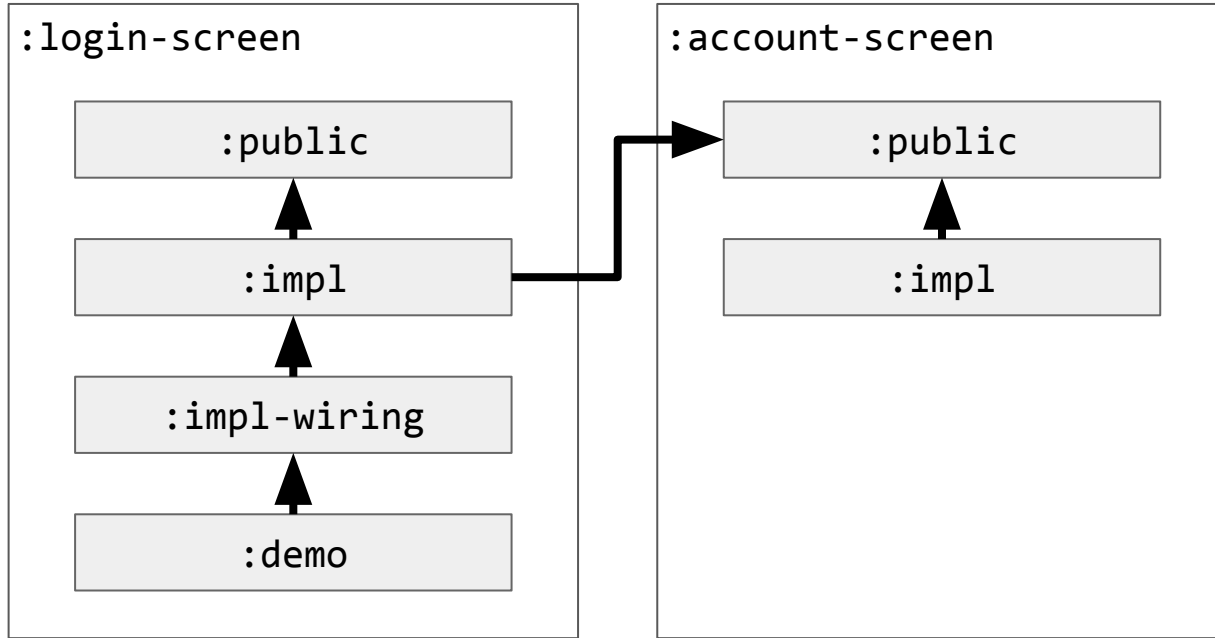
Development Apps



Development Apps



Development Apps



**Square Workflow**

Overview

Why Workflow?

User Guide



Tutorials and Samples



API Reference



Glossary

FAQ

Pre-1.0 Resources

Changelog

Contributing

Code of Conduct

Overview

license Apache License 2.0

Workflow is an application framework that provides architectural primitives.

Workflow is:

- Written in and used for Kotlin and Swift
- A unidirectional data flow library that uses immutable data within each Workflow. Data flows in a single direction from source to UI, and events in a single direction from the UI to the business logic.
- A library that supports writing business logic and complex UI navigation logic as state machines, thereby enabling confident reasoning about state and validation of correctness.
- Optimized for composability and scalability of features and screens.
- Corresponding UI frameworks that bind Rendering data classes for “views” (including event callbacks) to Mobile UI frameworks for Android and iOS.
- A corresponding testing framework that facilitates simple-to-write unit tests for all application business logic and helps ensure correctness.

**Table of contents**

Using Workflows in your project

Swift

Kotlin

Resources

Support & Contact

Releasing and Deploying

License

:account-screen:public

interface AccountScreenWorkflow : Workflow<Unit, Unit, LayeredScreen<PosLayering>>

:account-screen:impl

```
class RealAccountScreenWorkflow @Inject constructor(  
    ...  
) : AccountScreenWorkflow
```

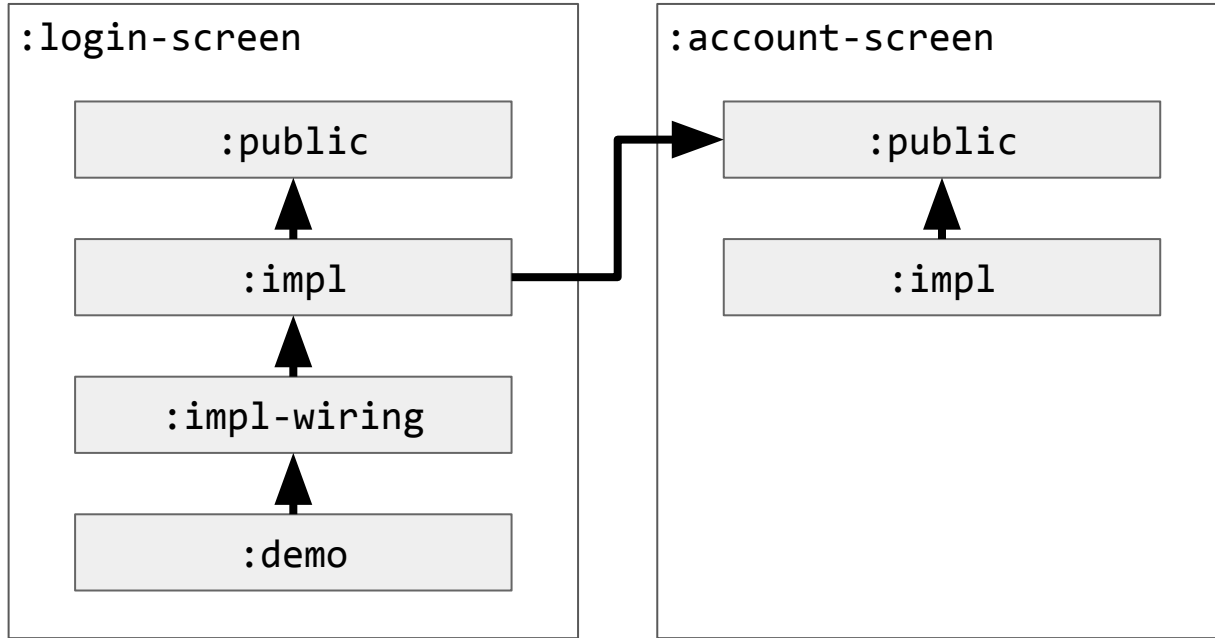
:login-screen:public

```
interface LoginScreenWorkflow : Workflow<Unit, Unit, LayeredScreen<PosLayering>>
```

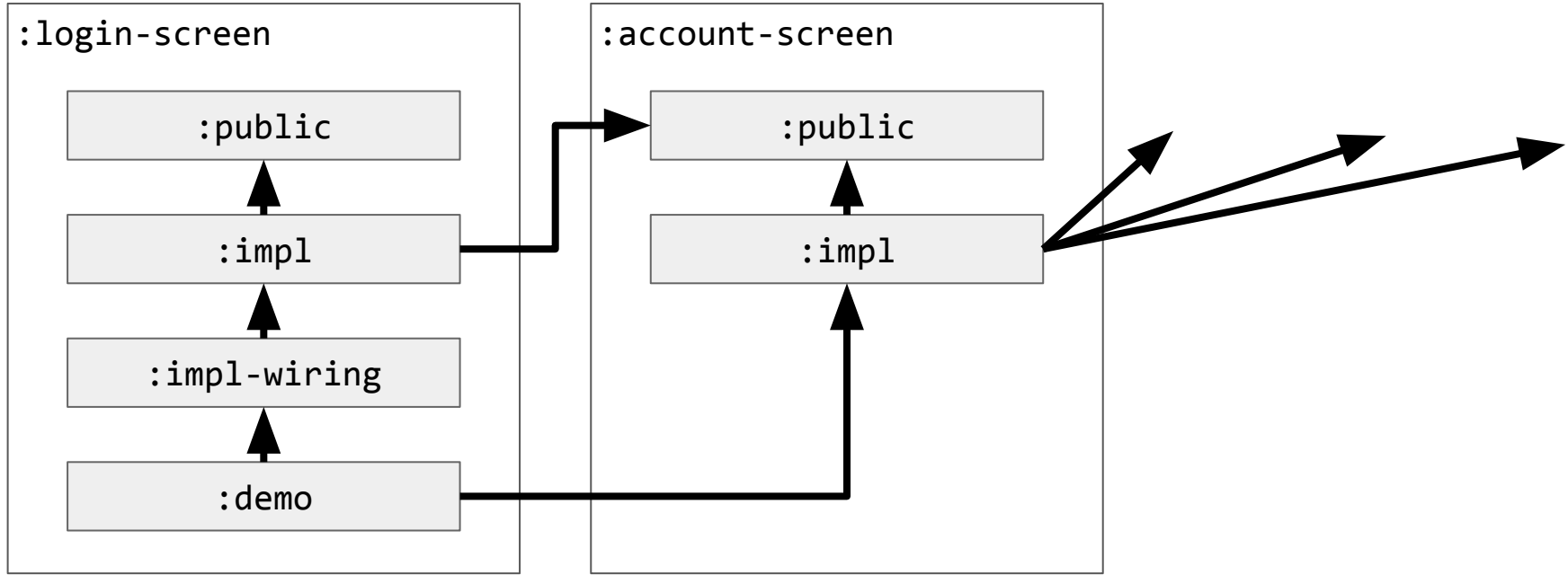
:login-screen:impl

```
class RealLoginScreenWorkflow @Inject constructor(  
    private val accountScreenWorkflow: Provider<AccountScreenWorkflow>  
) : LoginScreenWorkflow
```

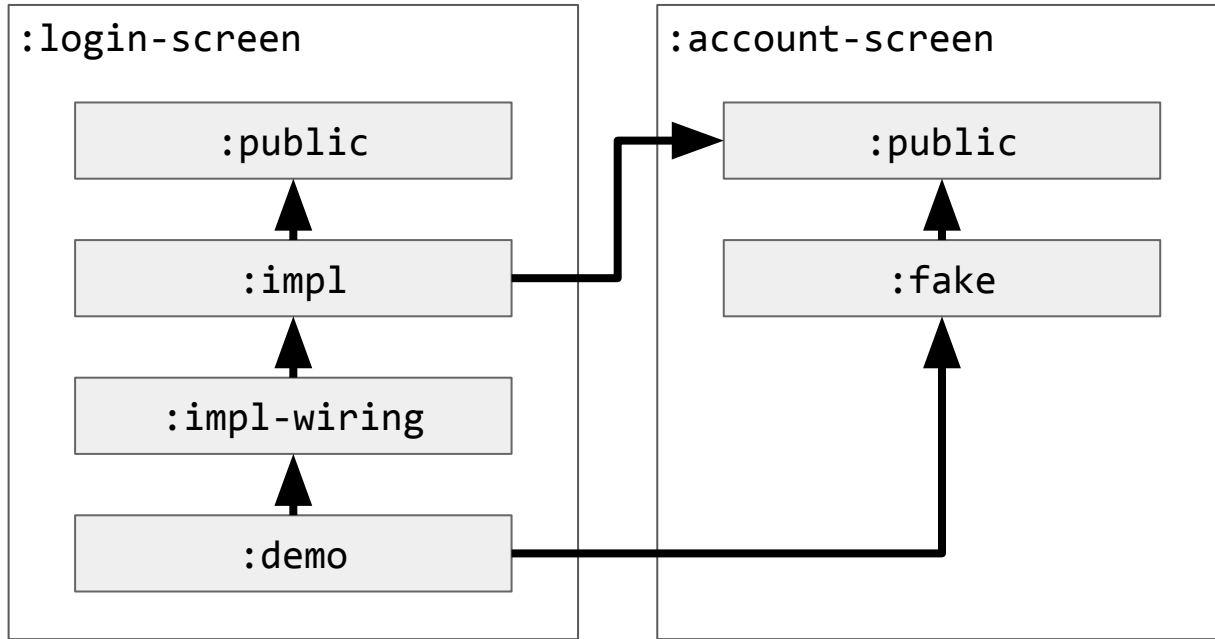
Development Apps



Development Apps



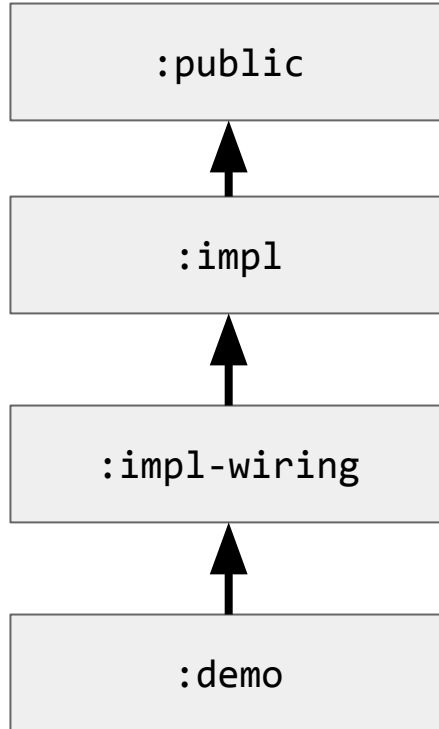
Development Apps



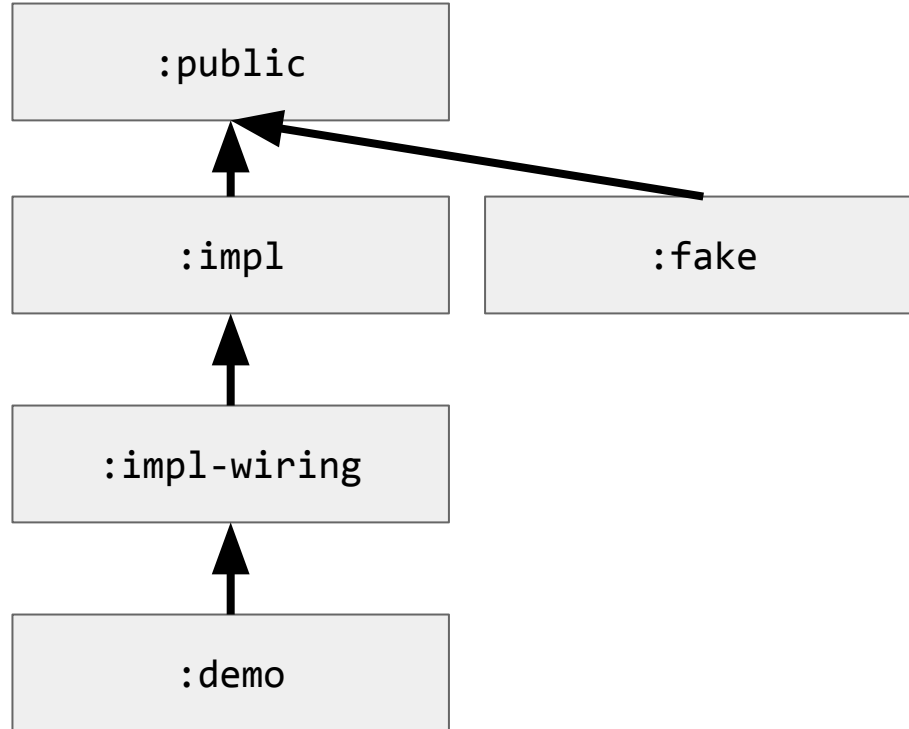
:account-screen:fake

```
class FakeAccountScreenWorkflow @Inject constructor() : AccountScreenWorkflow {  
    // ...  
}
```

Module Structure



Module Structure



:login-screen:demo

```
android {  
    defaultConfig {  
        applicationId "com.squareup.sample.login.screen.demo"  
    }  
}  
  
dependencies {  
    // ...  
    implementation project(':account-screen:fake')  
    implementation project(':development-app:impl-wiring')  
}
```

:login-screen:demo

```
@DevelopmentAppComponent
class DemoLoginApp : DevelopmentApplication()

@Module
@ContributesTo(ActivityScope::class)
object DemoLoginMainActivityModule {
    @Provides
    fun provideFeatureWorkflowProvider(
        workflow: LoginScreenWorkflow
    ): FeatureProvider =
        FeatureWorkflowProvider(
            workflow = workflow,
            props = Unit,
        )
}
```

:login-screen:demo

```
@DevelopmentAppComponent
class DemoLoginApp : DevelopmentApplication()

@Module
@ContributesTo(ActivityScope::class)
object DemoLoginMainActivityModule {
    @Provides
    fun provideFeatureWorkflowProvider(
        workflow: LoginScreenWorkflow
    ): FeatureProvider =
        FeatureWorkflowProvider(
            workflow = workflow,
            props = Unit,
        )
}
```

:login-screen:demo

```
@DevelopmentAppComponent
class DemoLoginApp : DevelopmentApplication()

@Module
@ContributesTo(ActivityScope::class)
object DemoLoginMainActivityModule {
    @Provides
    fun provideFeatureWorkflowProvider(
        workflow: LoginScreenWorkflow
    ): FeatureProvider =
        FeatureWorkflowProvider(
            workflow = workflow,
            props = Unit,
        )
}
```


UI Tests

- Part of the development apps
 - Faster builds
 - Tend to be very stable
 - More build cache hits
 - Automatic test sharding
- Integration tests live in the final applications
 - Test robots are shared across all apps

Mechanisms

Mechanisms: module structure through convention plugins

```
plugins {  
    id 'com.android.library'  
    id 'org.jetbrains.kotlin.android'  
    id 'org.jetbrains.kotlin.kapt'  
    id 'com.squareup.anvil'  
}  
  
dependencies {  
    api project(':account-screen:public')  
    ...  
}  
  
tasks.withType(KotlinCompile).configureEach {  
    kotlinOptions {  
        jvmTarget = JavaVersion.VERSION_11  
    }  
}
```

Mechanisms: module structure through convention plugins

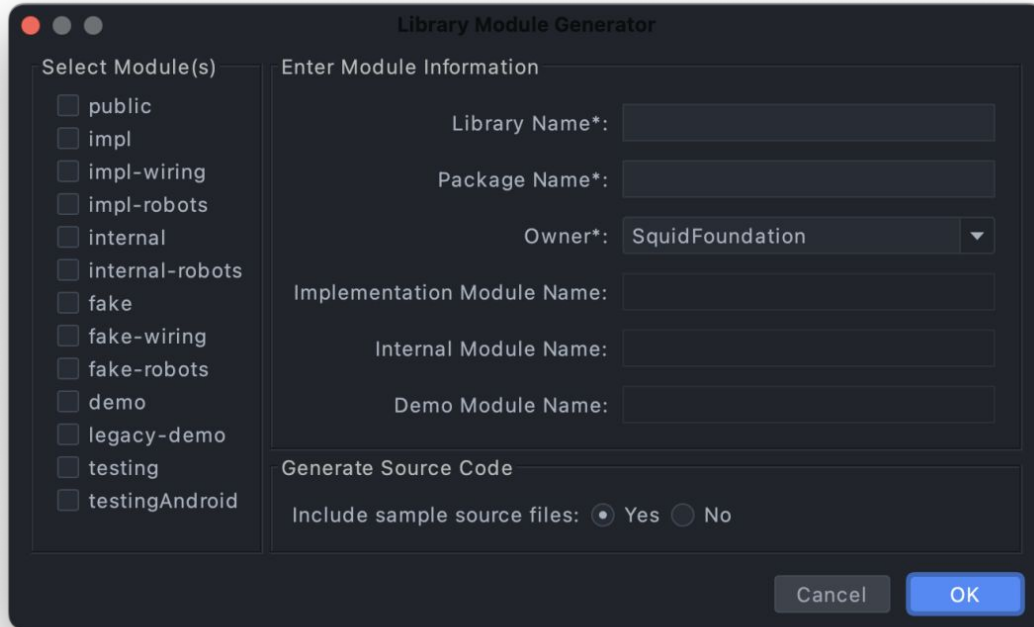
```
// :login-screen:impl
plugins {
    id 'com.squareup.android.lib'
}
```

```
// :login-screen:demo
plugins {
    id 'com.squareup.demo-app'
}
```

Mechanisms: module structure enforced through Lint rules

```
class NoImplDependencyModuleCheck : ModuleCheck {  
    override fun runCheck(project: Project) {  
        ...  
  
        fail(  
            "No :impl dependency is allowed. Requested: " +  
            "${implDependencies.joinToString { it.dependencyName }} in project: $project."  
        )  
    }  
}
```

Mechanisms: library generator



A dark-themed dialog box titled "Library Module Generator". It is divided into three main sections. The left section, titled "Select Module(s)", contains a list of 15 modules, each with an unchecked checkbox: public, impl, impl-wiring, impl-robots, internal, internal-robots, fake, fake-wiring, fake-robots, demo, legacy-demo, testing, and testingAndroid. The right section, titled "Enter Module Information", contains five input fields: "Library Name*" (empty), "Package Name*" (empty), "Owner*" (a dropdown menu showing "SquidFoundation"), "Implementation Module Name:" (empty), and "Internal Module Name:" (empty). Below these is a "Demo Module Name:" field (empty). The bottom section, titled "Generate Source Code", contains a label "Include sample source files:" followed by two radio buttons, "Yes" (which is selected) and "No". At the bottom right of the dialog are two buttons: "Cancel" and "OK".

Library Module Generator

Select Module(s)

- ☐ public
- ☐ impl
- ☐ impl-wiring
- ☐ impl-robots
- ☐ internal
- ☐ internal-robots
- ☐ fake
- ☐ fake-wiring
- ☐ fake-robots
- ☐ demo
- ☐ legacy-demo
- ☐ testing
- ☐ testingAndroid

Enter Module Information

Library Name*:

Package Name*:

Owner*:

Implementation Module Name:

Internal Module Name:

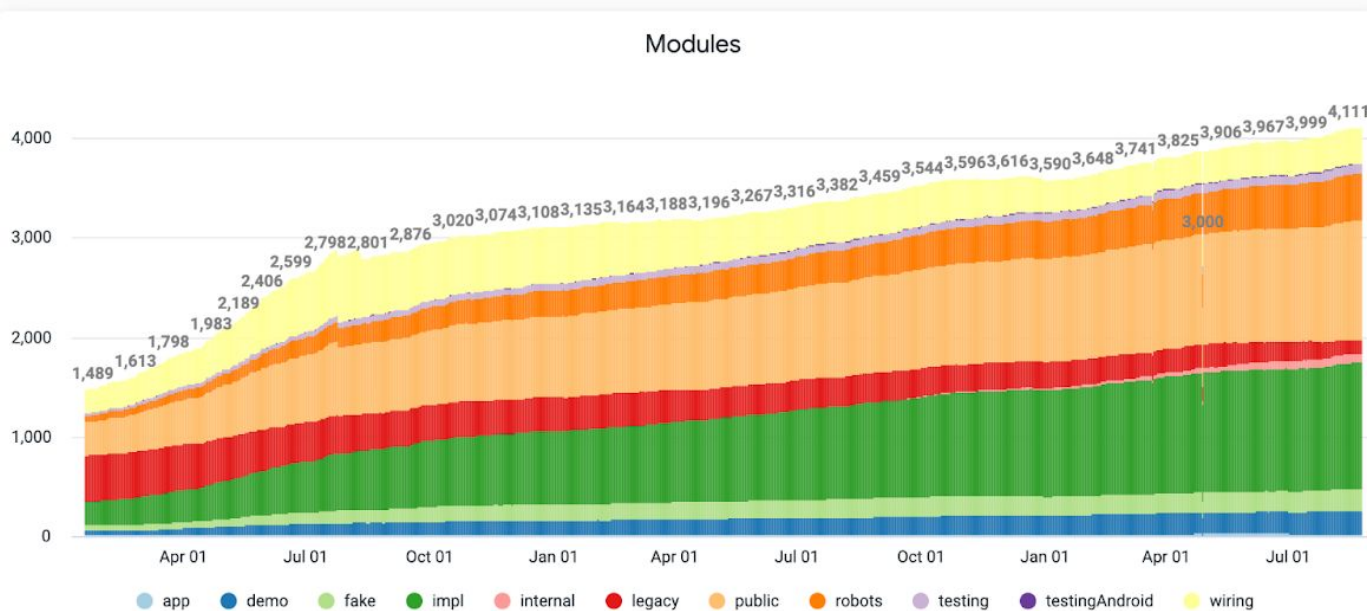
Demo Module Name:

Generate Source Code

Include sample source files: ☒ Yes ☐ No

Cancel OK

Mechanisms: library generator



4,119

total modules

▲ 6 week over week

Mechanisms: library generator

111

Monthly Active Users

1092.3

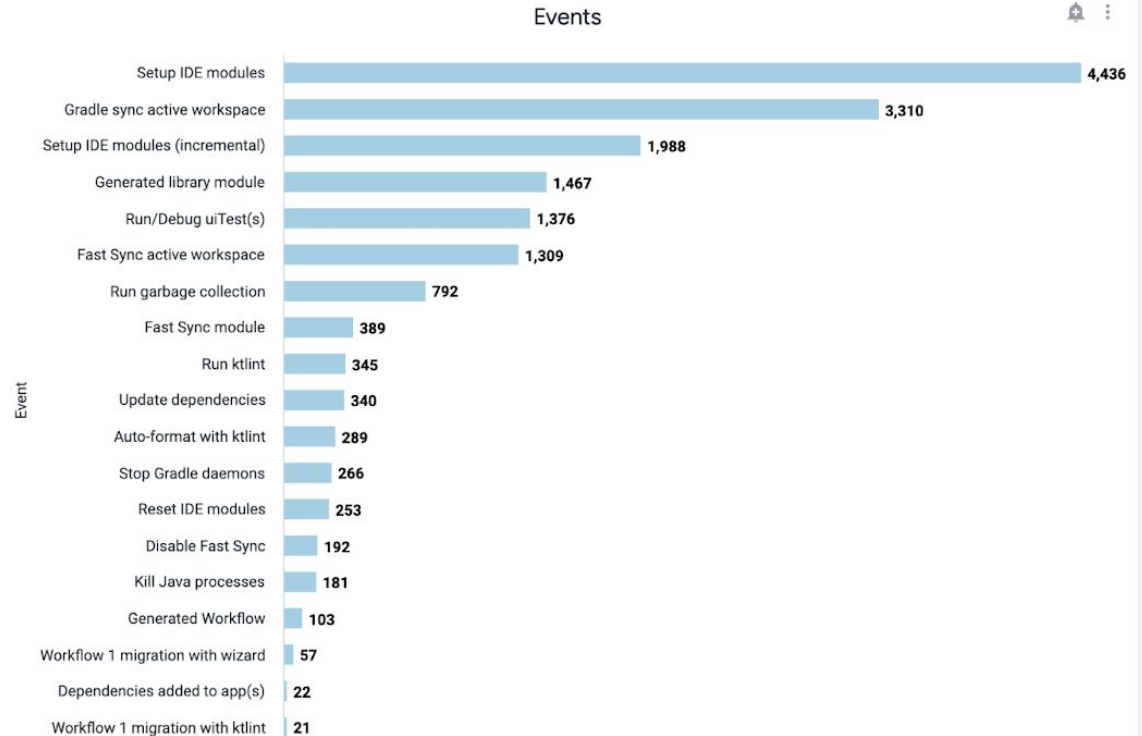
Total Eng Hours Saved

1,467

Modules Generated

103

Workflows Generated



Mechanisms: design patterns & frameworks

- Workflow
 - Composition was key
 - <https://square.github.io/workflow/>
- Dependency inversion
 - Enforced through the module structure
- Dependency Injection
 - Anvil: <https://github.com/square/anvil/>
 - Dagger 2: <https://dagger.dev/>

Mechanisms: design patterns & frameworks: Anvil

```
class RealLoginScreenWorkflow @Inject constructor(  
    private val accountScreenWorkflow: Provider<AccountScreenWorkflow>  
) : LoginScreenWorkflow
```

Mechanisms: design patterns & frameworks: Anvil

```
class RealLoginScreenWorkflow @Inject constructor(  
    private val accountScreenWorkflow: Provider<AccountScreenWorkflow>  
) : LoginScreenWorkflow
```

```
@Module  
abstract class LoginScreenWorkflowModule {  
    @Binds abstract fun bindLoginScreenWorkflow(  
        workflow: RealLoginScreenWorkflow  
    ) : LoginScreenWorkflow  
}
```

Mechanisms: design patterns & frameworks: Anvil

```
class RealLoginScreenWorkflow @Inject constructor(  
    private val accountScreenWorkflow: Provider<AccountScreenWorkflow>  
) : LoginScreenWorkflow  
  
@Module  
abstract class LoginScreenWorkflowModule {  
    @Binds abstract fun bindLoginScreenWorkflow(  
        workflow: RealLoginScreenWorkflow  
    ) : LoginScreenWorkflow  
}  
  
@Component(  
    modules = {  
        LoginScreenWorkflowModule::class,  
    }  
)  
interface AppComponent
```

Mechanisms: design patterns & frameworks: Anvil

```
@ContributesBinding(ActivityScope::class)
class RealLoginScreenWorkflow @Inject constructor(
    private val accountScreenWorkflow: Provider<AccountScreenWorkflow>
) : LoginScreenWorkflow
```

Mechanisms: design patterns & frameworks: Anvil

```
@ContributesBinding(ActivityScope::class)
class RealLoginScreenWorkflow @Inject constructor(
    private val accountScreenWorkflow: Provider<AccountScreenWorkflow>
) : LoginScreenWorkflow
```

Mechanisms: design patterns & frameworks: Anvil

```
@DevelopmentAppComponent
class DemoLoginApp : DevelopmentApplication()

@Module
@ContributesTo(ActivityScope::class)
object DemoLoginMainActivityModule {
    @Provides
    fun provideFeatureWorkflowProvider(
        workflow: LoginScreenWorkflow
    ): FeatureProvider =
        FeatureWorkflowProvider(
            workflow = workflow,
            props = Unit,
        )
}
```

Mechanisms: design patterns & frameworks: Anvil

```
@DevelopmentAppComponent
class DemoLoginApp : DevelopmentApplication()

@Module
@ContributesTo(ActivityScope::class)
object DemoLoginMainActivityModule {
    @Provides
    fun provideFeatureWorkflowProvider(
        workflow: LoginScreenWorkflow
    ): FeatureProvider =
        FeatureWorkflowProvider(
            workflow = workflow,
            props = Unit,
        )
}
```




Dagger + Anvil:

Learning to Love Dependency Injection



**Gabriel
Peal**

Lottie + MvRx
Tonal



**Ralf
Wondratschek**

Software Engineer
Square



JUNE 2nd - 3rd, 2022

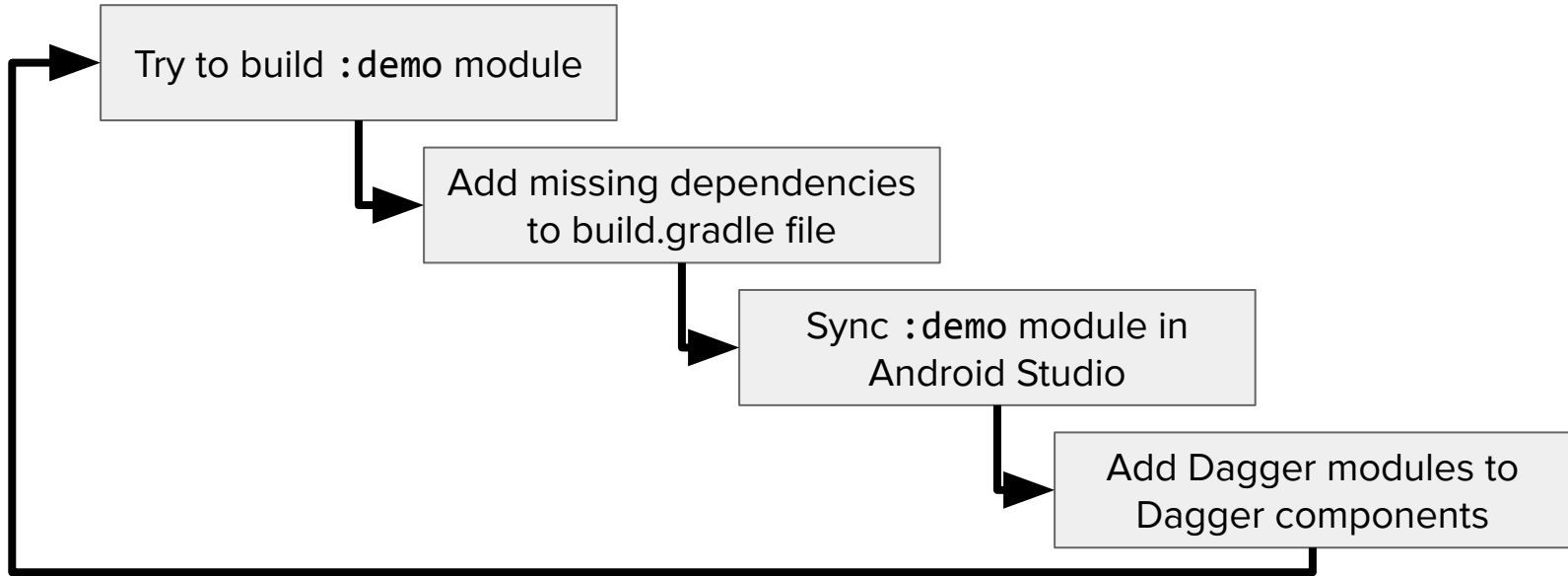


00:01

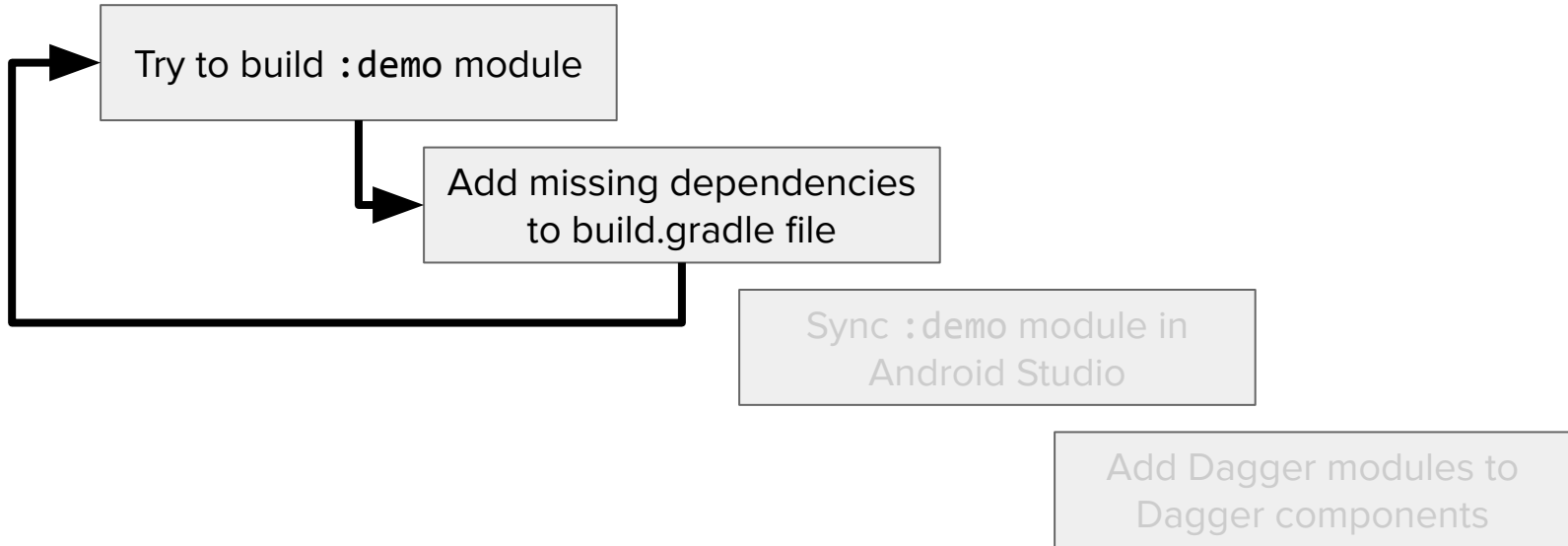


<https://bit.ly/3WbPHdm>

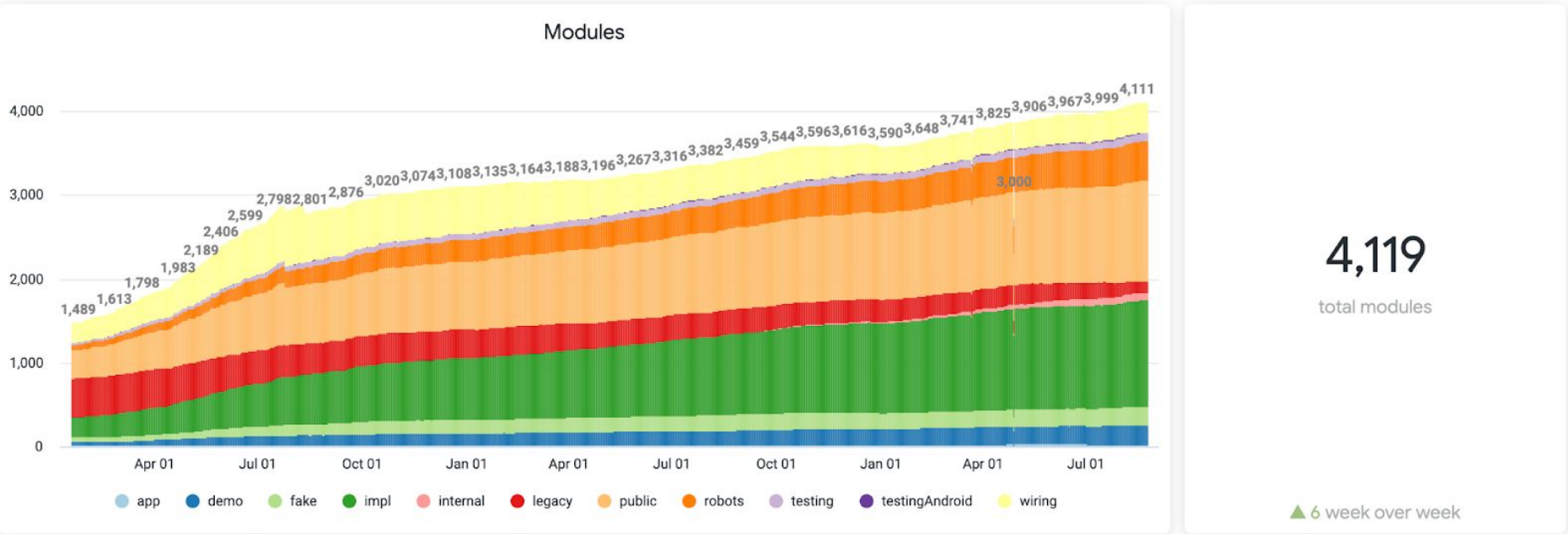
Mechanisms: design patterns & frameworks: Anvil



Mechanisms: design patterns & frameworks: Anvil



Mechanisms: dashboards



Mechanisms: dashboards

1,201

public modules

▲ 4 week over week

1,272

impl modules

▲ 5 week over week

219

fake modules

▲ 0 week over week

90

internal modules

▲ 0 week over week

212

demo development mod...

▲ 1 week over week

235

demo non-development ...

▼ -2 week over week

140

legacy modules

▼ -1 week over week

31

app modules

▲ 0 week over week

94

testing modules

▲ 0 week over week

11

testingAndroid modules

▲ 0 week over week

468

robots modules

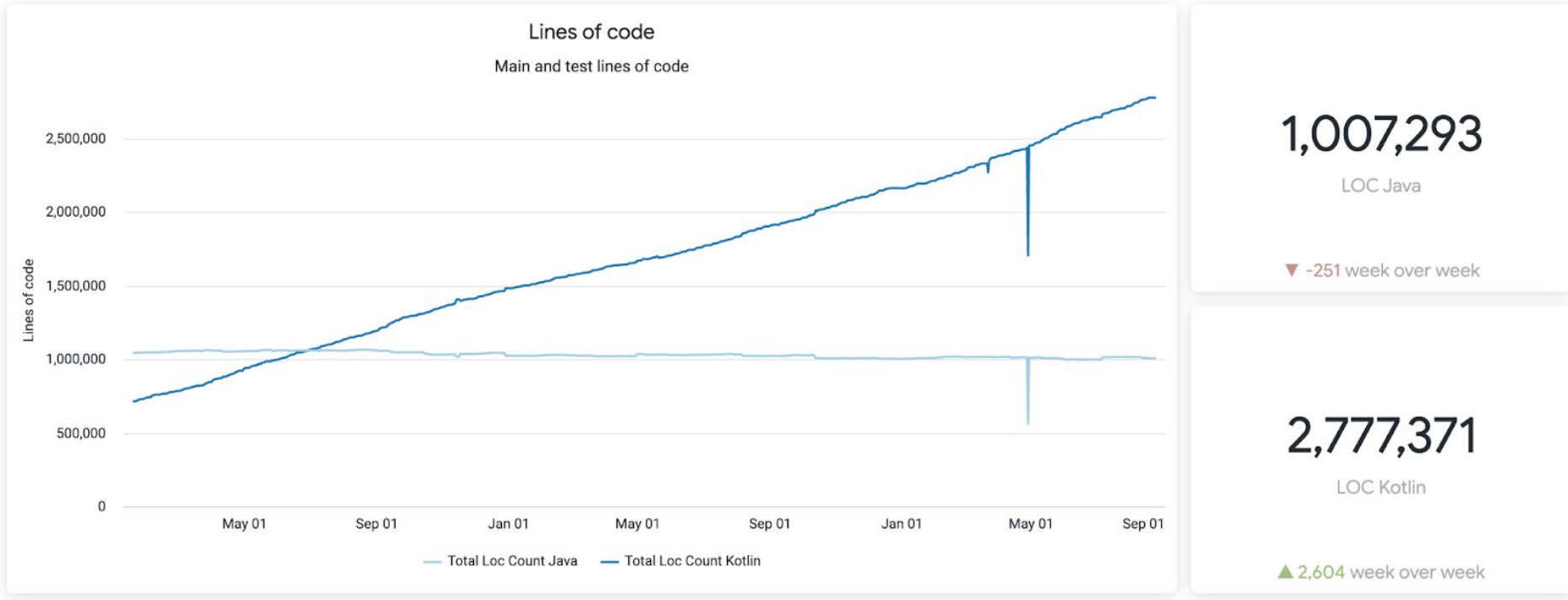
▲ 0 week over week

358

wiring modules

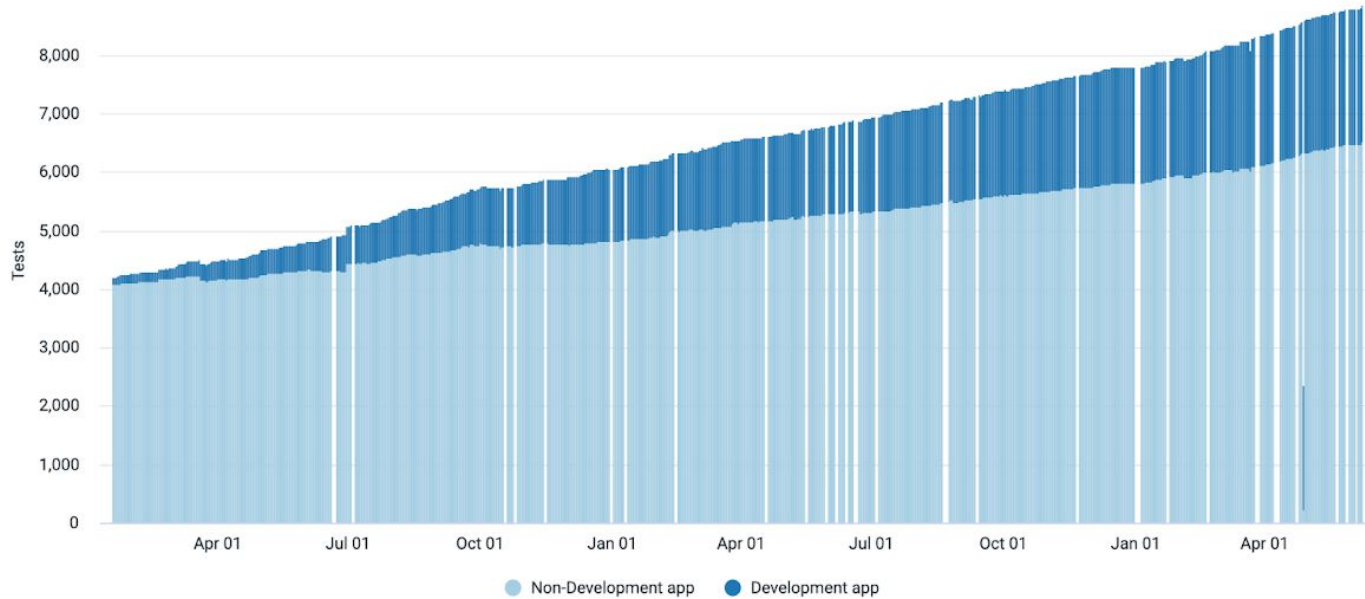
▲ 0 week over week

Mechanisms: dashboards



Mechanisms: dashboards

Total Android tests by development and non-development apps



2,338

Android development tests

▲ 10 week over week

9,386

Android non-development tests

▲ 6 week over week

Mechanisms: dashboards

212

Development Apps

▲ 1 week over week

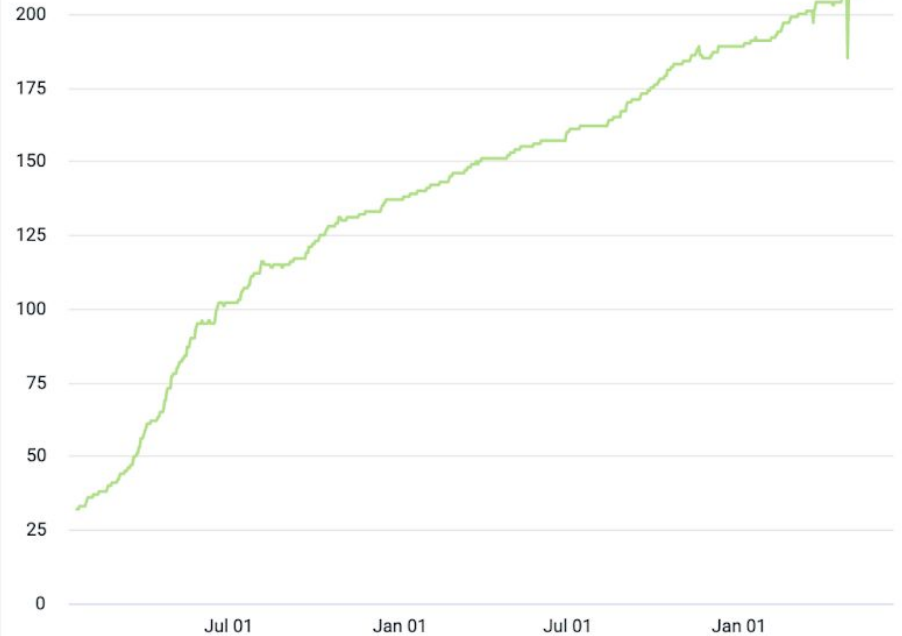
47.0

Dev App Median Build Time (sec)

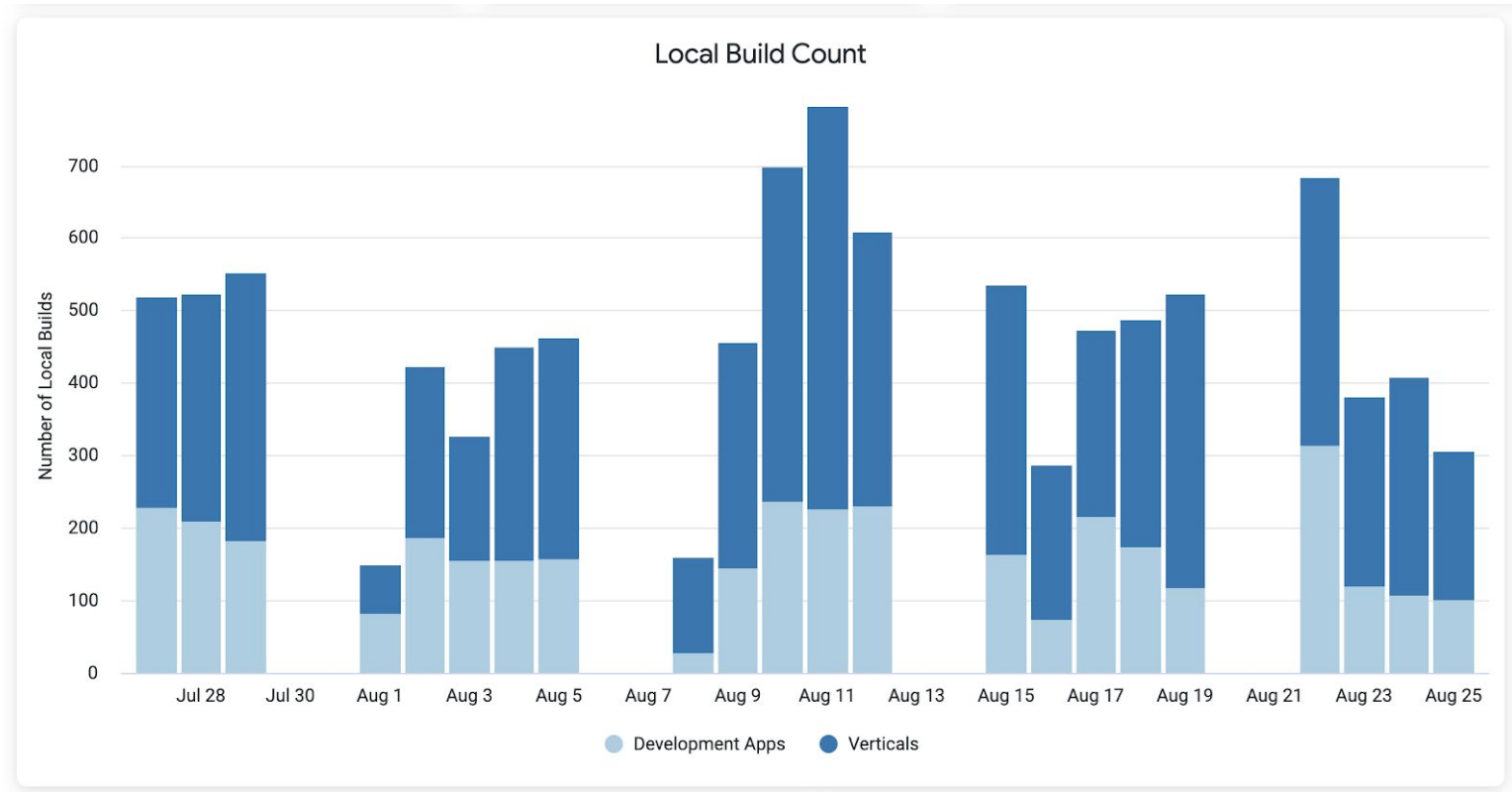
375.0

Vertical Median Build Time (sec)

Development Apps



Mechanisms: dashboards



Mechanisms: dashboards

111

Monthly Active Users

1092.3

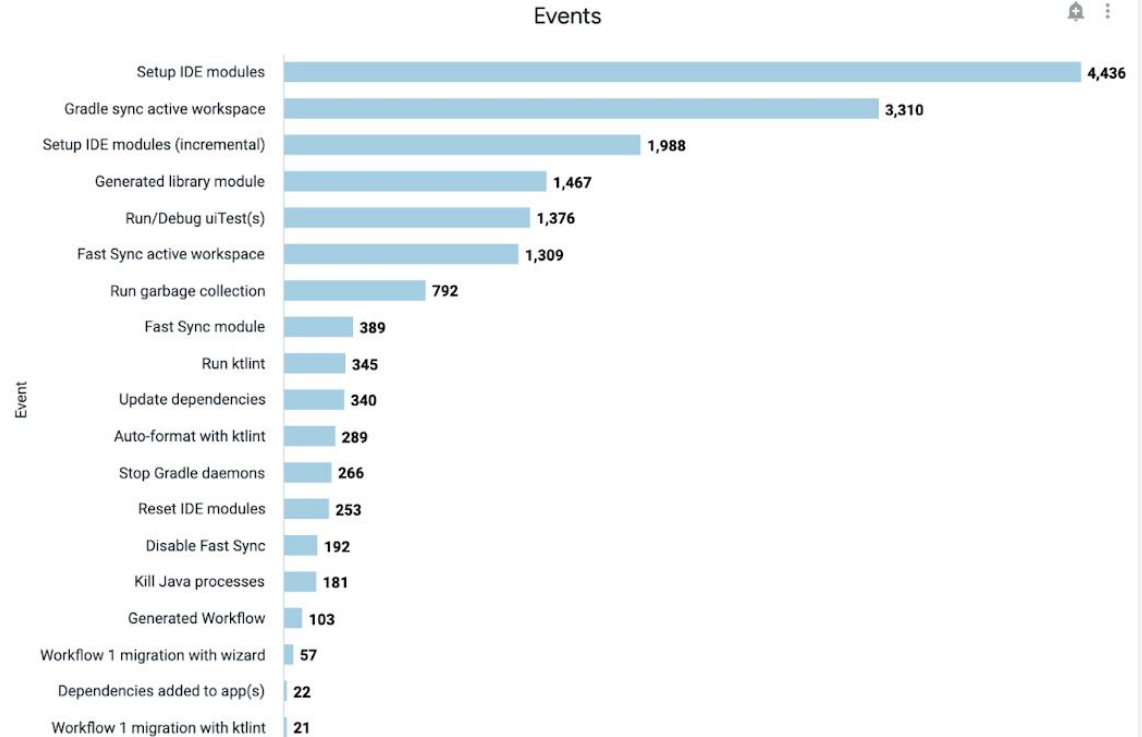
Total Eng Hours Saved

1,467

Modules Generated

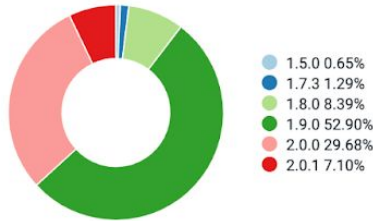
103

Workflows Generated

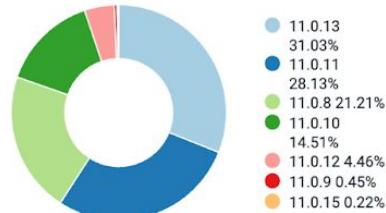


Mechanisms: dashboards

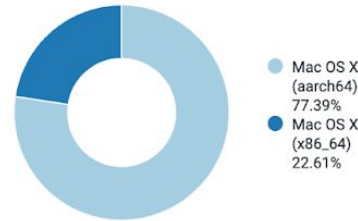
Toolkit Plugin Versions



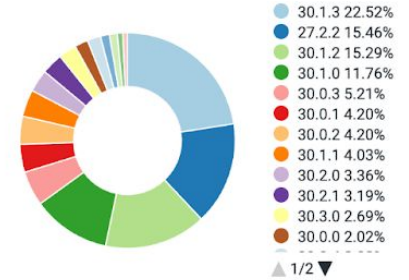
JDK Versions



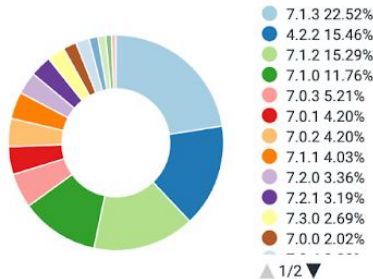
OS Architectures



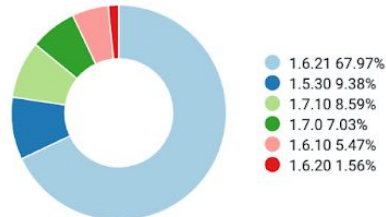
Build Tools Versions



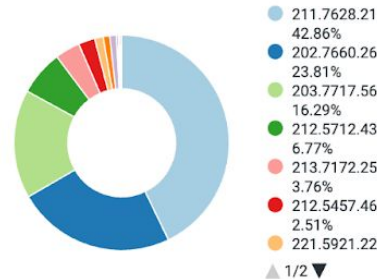
Android Gradle Plugin Versions



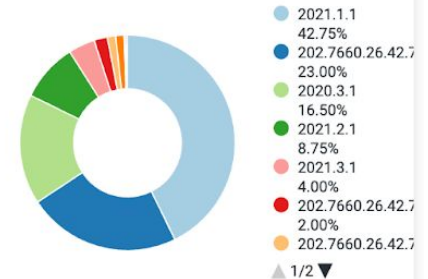
Kotlin Gradle Plugin Versions



IntelliJ Core Versions

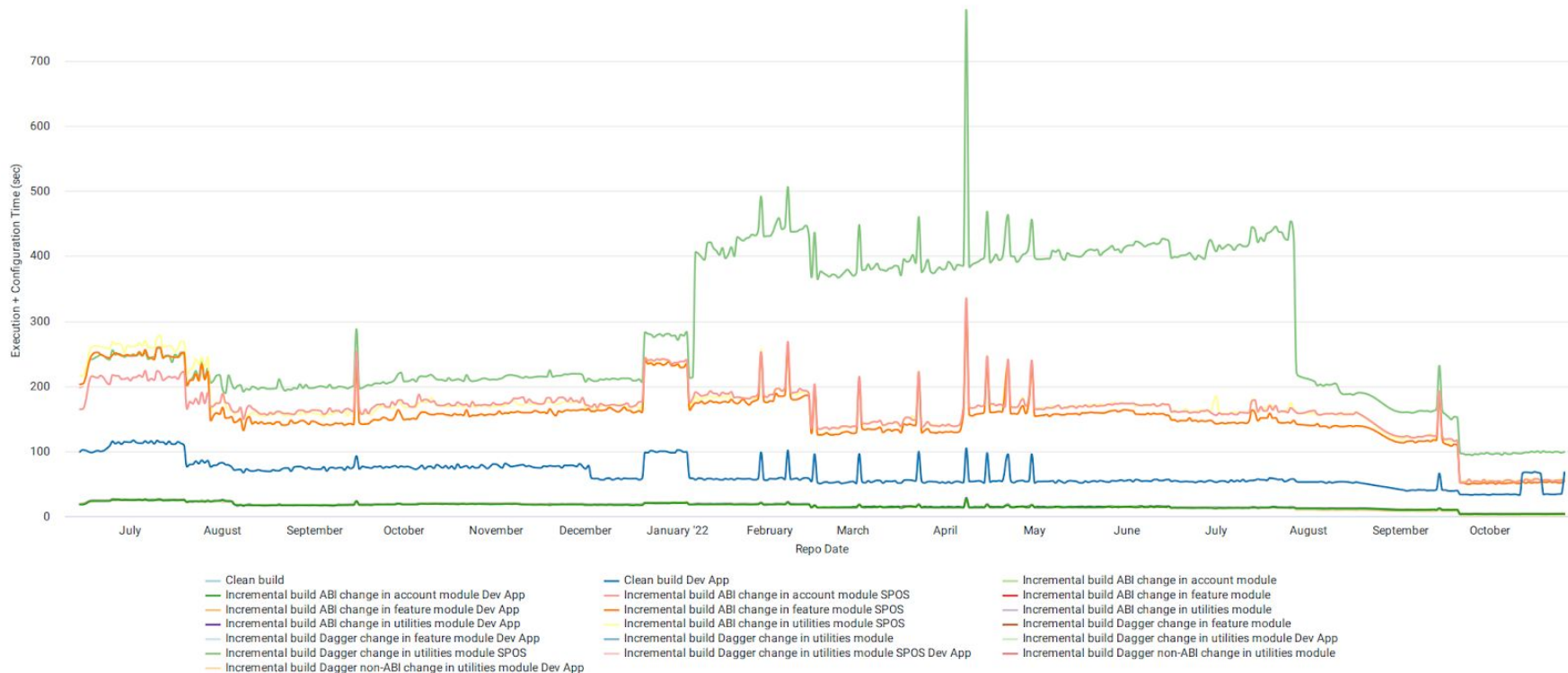


Android Studio Versions



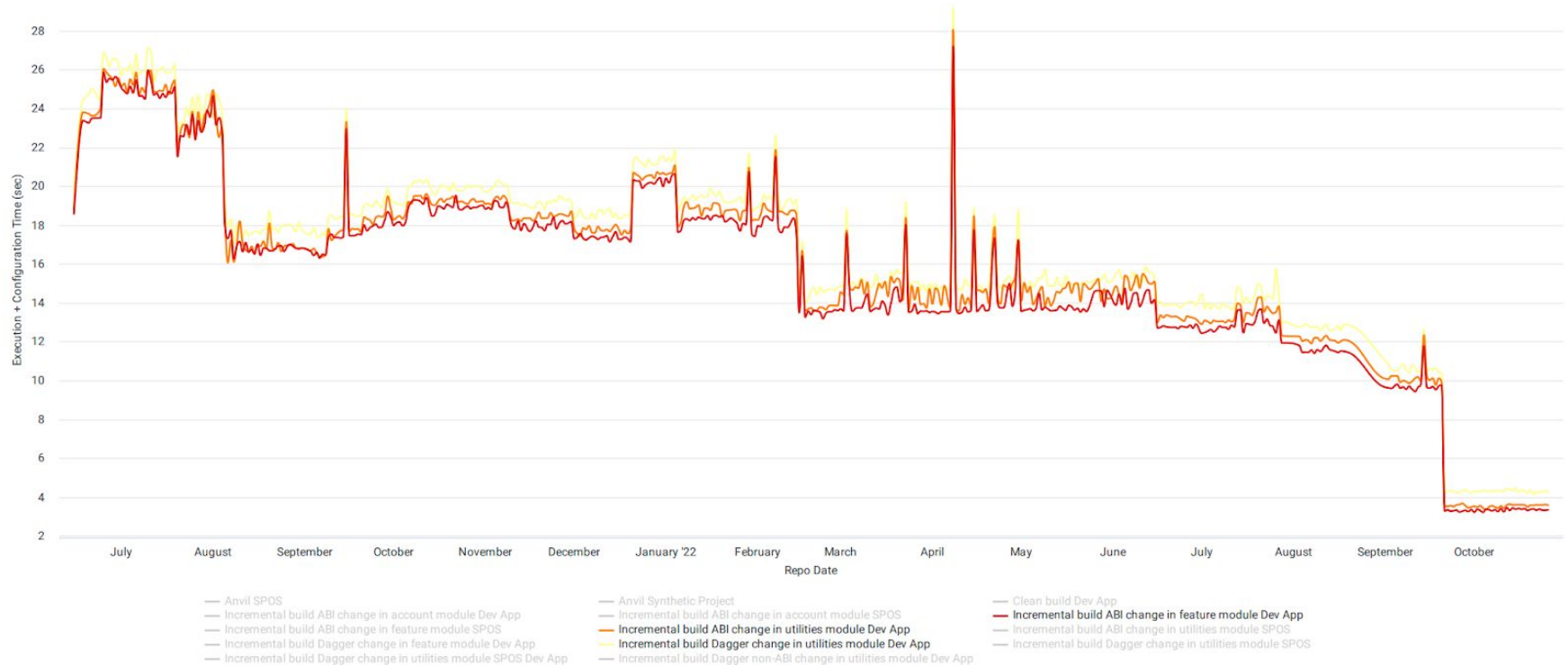
Mechanisms: benchmarks

Benchmark Results



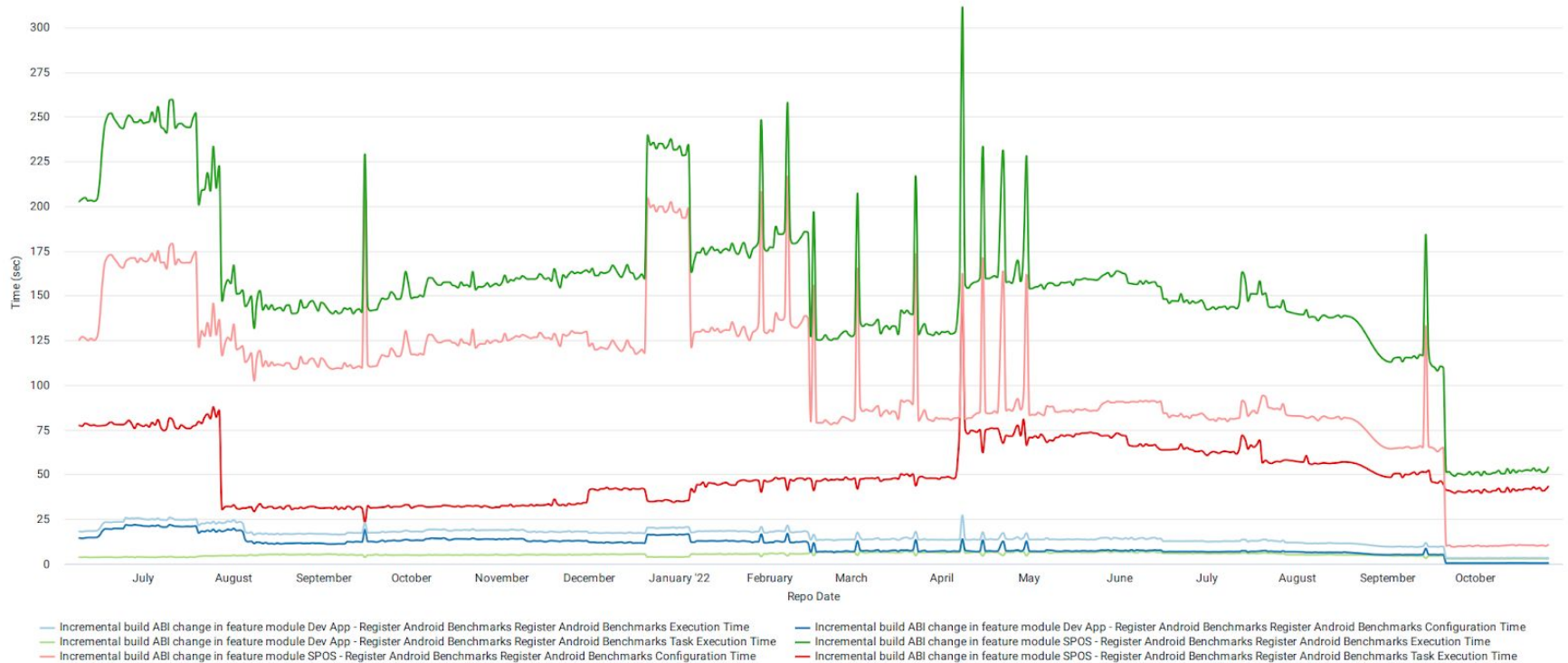
Mechanisms: benchmarks

Benchmark Results Dev Apps



Mechanisms: benchmarks

Configuration & Execution Times

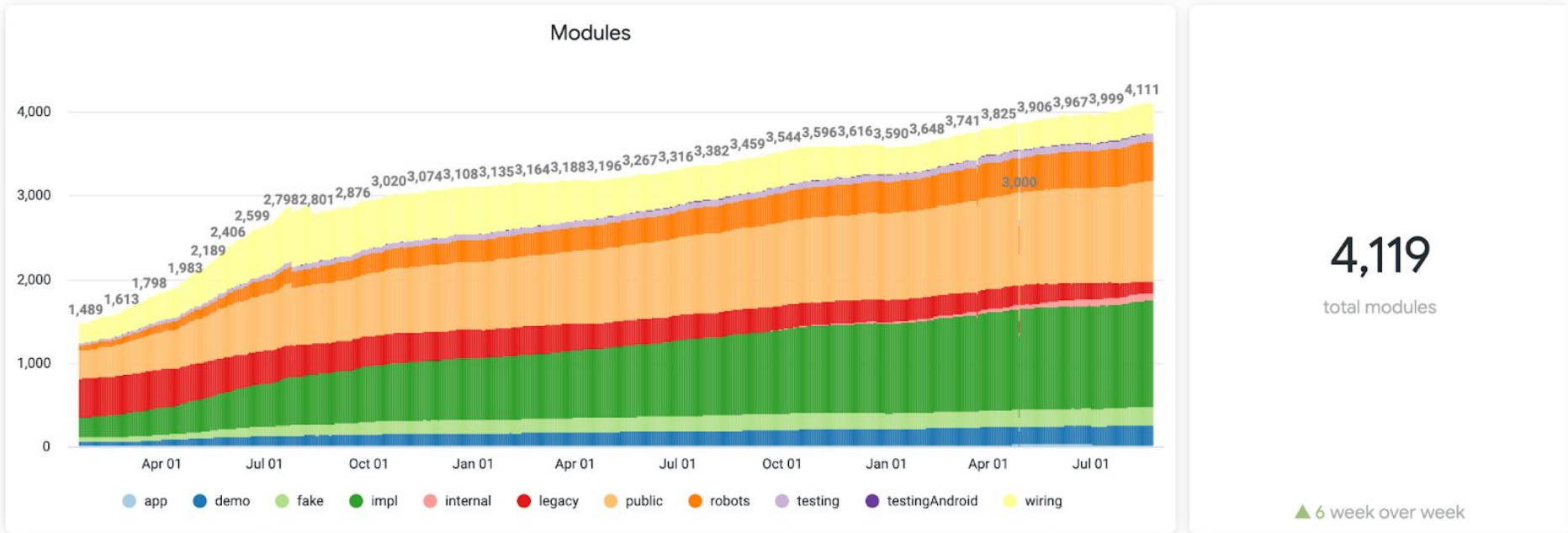


Mechanisms: migrations

- We were a small team and relied on the help of feature teams to contribute to common goals
 - Legacy module migration
 - Adopt Anvil
 - Extract UI tests into :demo apps

Challenges

Challenges: module count



Challenges: module count

Solution:

- Convention plugins
 - Benchmarks
 - Anvil
 - Configuration caching
 - Partial IDE sync
- <https://github.com/dropbox/focus>

Challenges: legacy modules

Legacy modules break our dependency rules

Solution:

- Finish migration

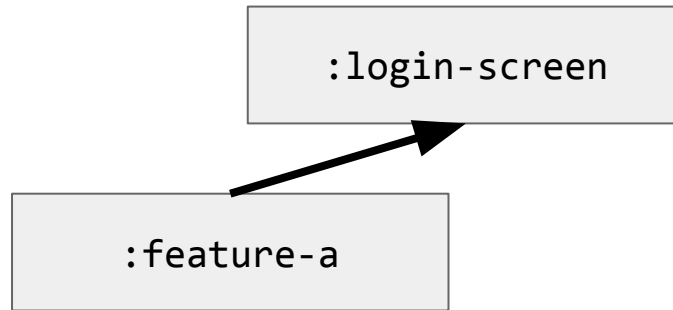
Challenges: granular modules

Anti-pattern

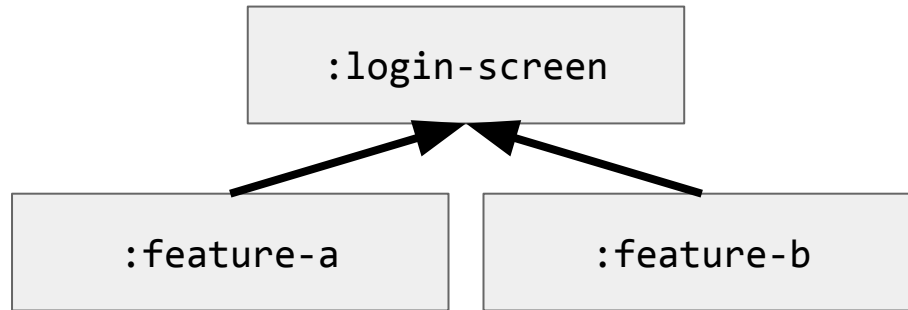
Challenges: granular modules

:login-screen

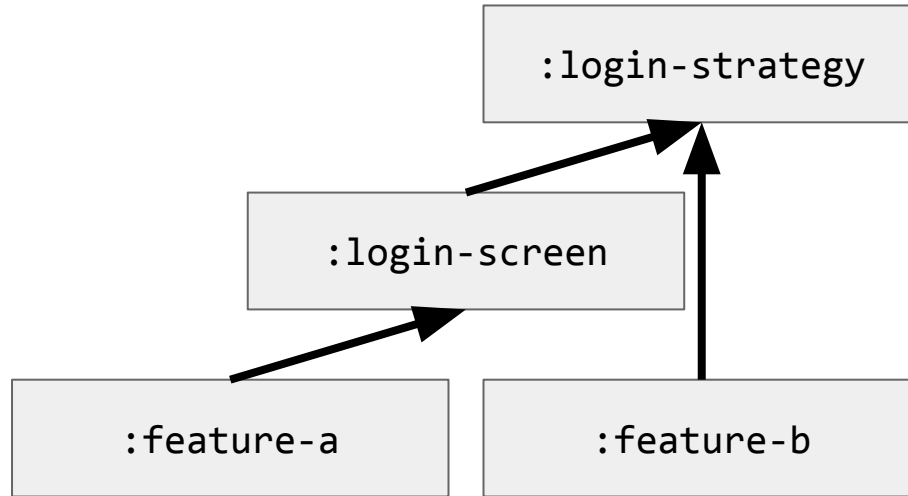
Challenges: granular modules



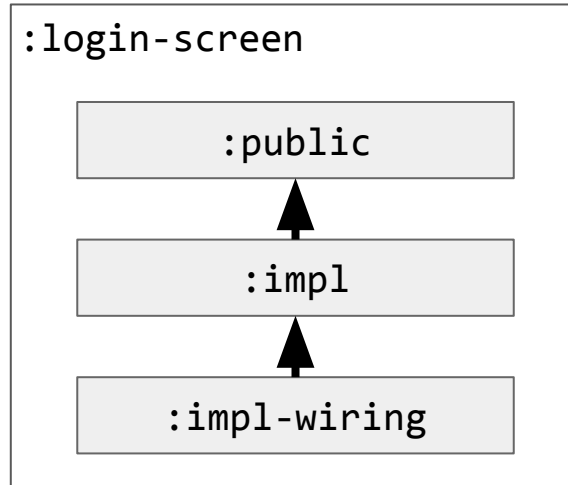
Challenges: granular modules



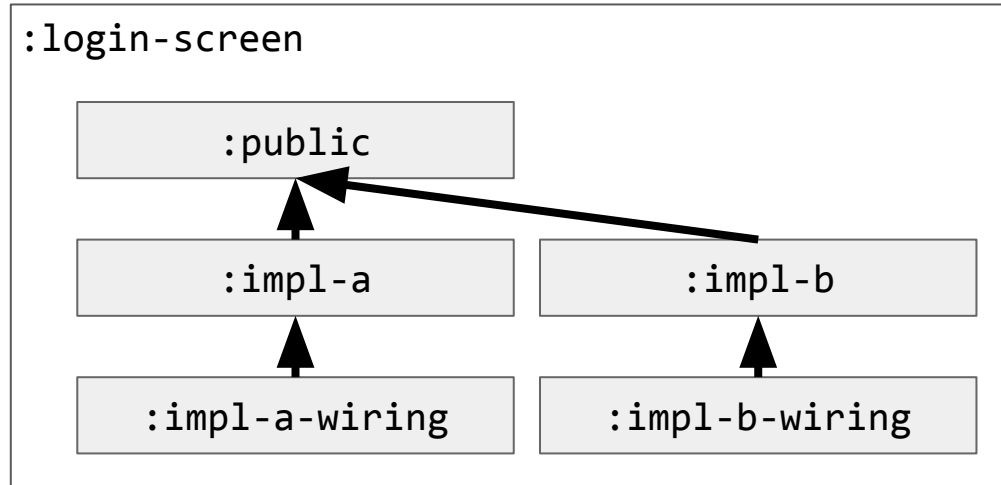
Challenges: granular modules



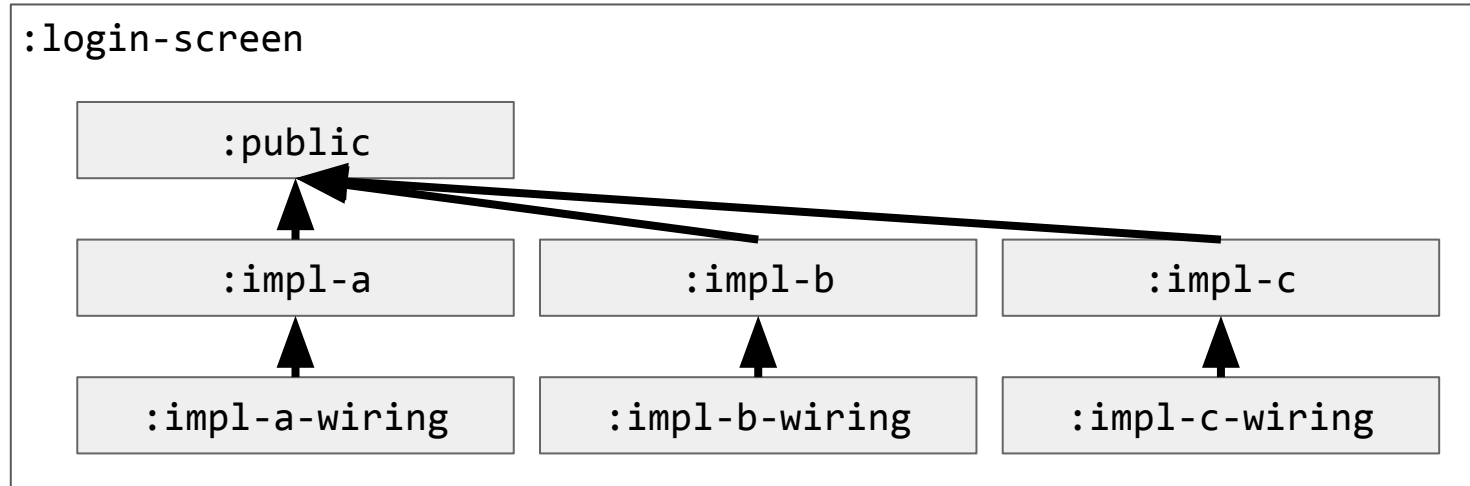
Challenges: granular modules



Challenges: granular modules



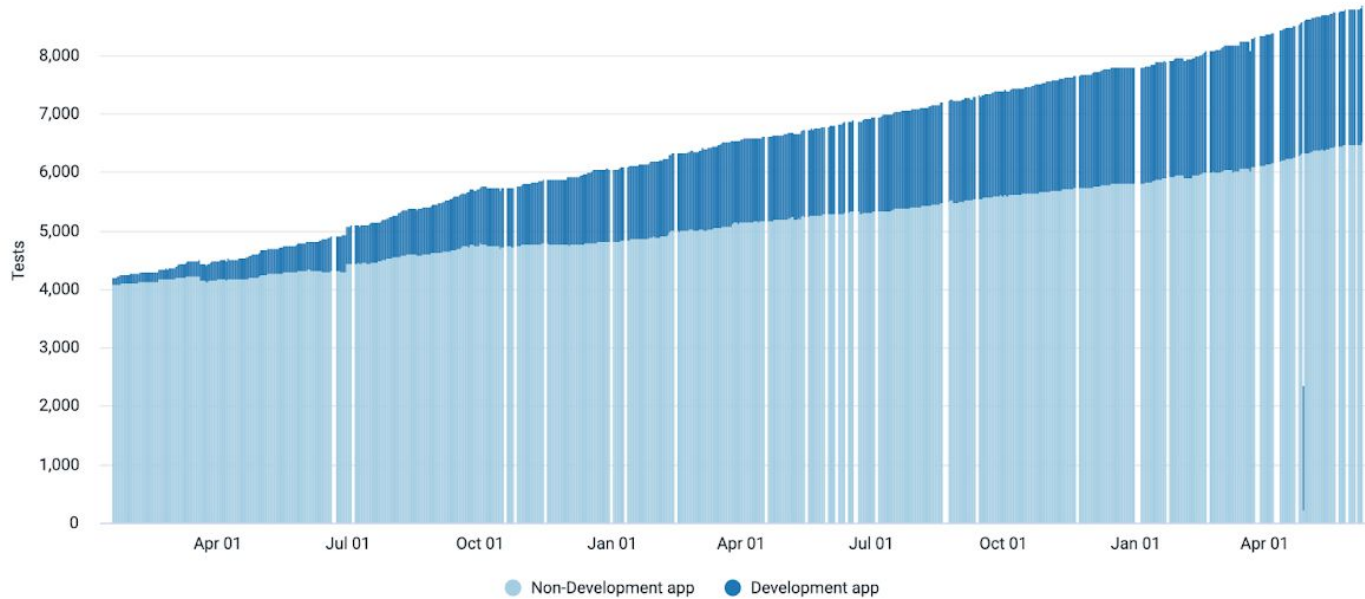
Challenges: granular modules



Challenges: development app shell

Challenges: extract UI tests

Total Android tests by development and non-development apps



2,338

Android development tests

▲ 10 week over week

9,386

Android non-development tests

▲ 6 week over week

Isolated Development



Ralf Wondratschek

@vRallev

amazon