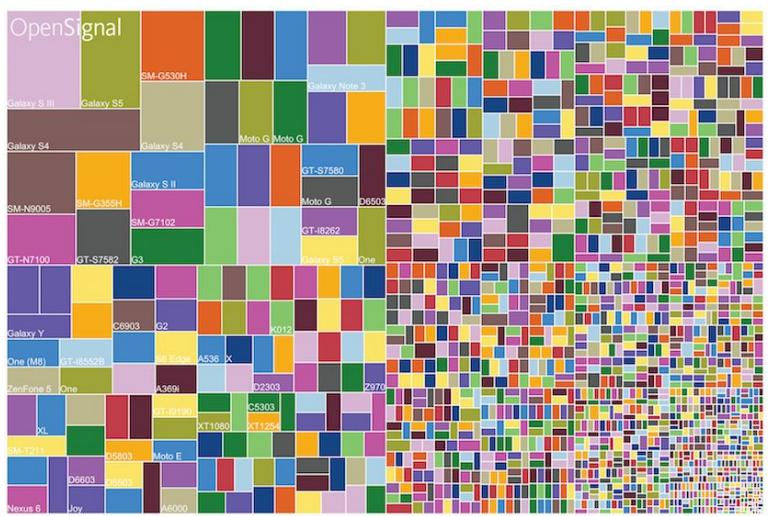


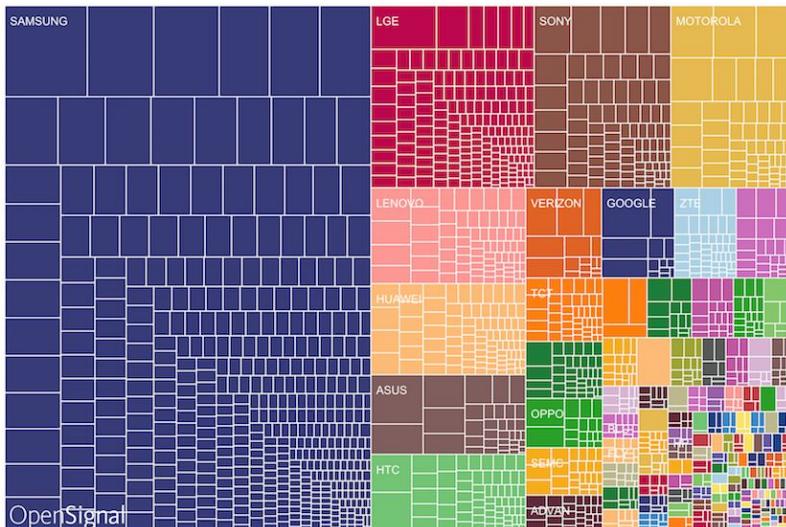


- 1 Rétrospective L1
- 2 Architecture App. Mobile
- 3 Exposition de ressources (REST)
- 4 Intro. à Android
- 5 Travail à faire pour L2

Ceci n'est pas un cours de développement mobile en Android

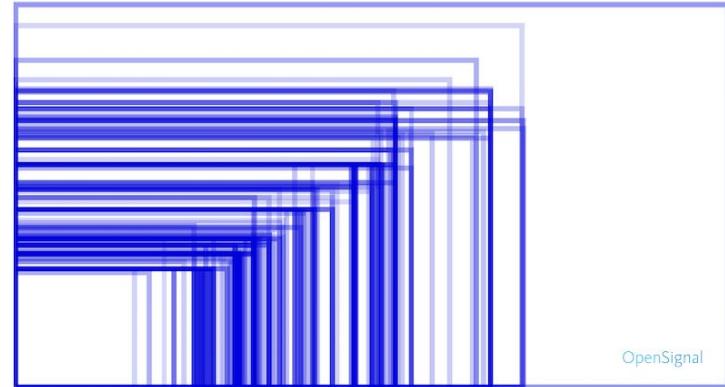


Répartition par fabricants



https://www.opensignal.com/sites/opensignal-com/files/data/reports/global/data-2015-08/2015_08_fragmentation_report.pdf

Fragmentation des tailles d'écrans



OpenSignal

https://www.opensignal.com/sites/opensignal-com/files/data/reports/global/data-2015-08/2015_08_fragmentation_report.pdf

Android versus iOS

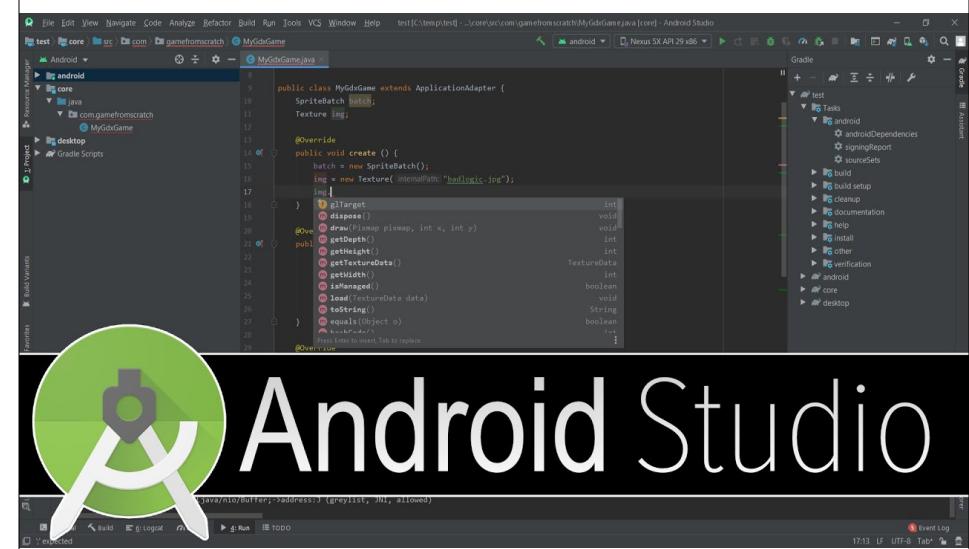
• Philosophie Apple

- Maîtrise du matériel, fermeture du système d'exploitation
- Validation préalable des applications dans l'AppStore

• Philosophie Android

- Matériel ouvert, système d'exploitation Open-source
 - Validation a posteriori si une application est signalée
- Il est possible de développer du multiplateforme (Flutter, Cordova)

AndroidStudio pour le développement



Utilisation d'émulateurs

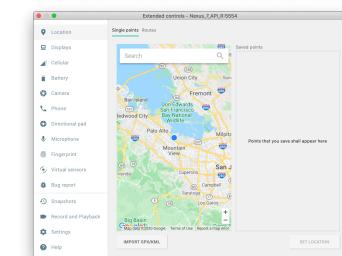
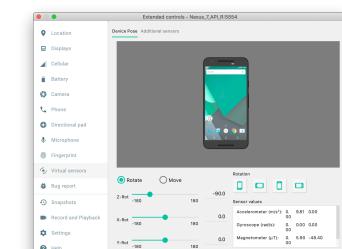
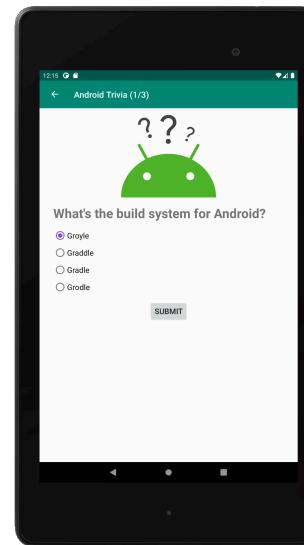


L'émulateur fonctionne comme un "vrai" périphérique.

On va téléverser les applications développées dans l'émulateur pour les besoins des tests locaux

On peut aussi envoyer l'application sur un via périphérique en USB

Exemple d'utilisation de l'émulateur

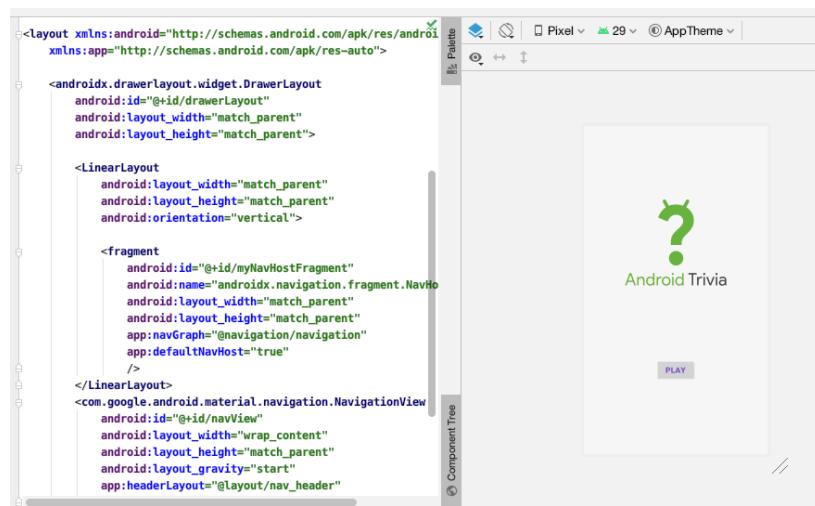


Activity: Point d'entrée de l'application

```
class MainActivity : AppCompatActivity() {  
    private lateinit var drawerLayout: DrawerLayout
```

```
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        @SuppressLint("UNUSED_VARIABLE")  
        val binding = DataBindingUtil.setContentView<ActivityMainBinding>(this, R.layout.activity_main)  
  
        drawerLayout = binding.drawerLayout  
  
        val navController = this.findNavController(R.id.myNavHostFragment)  
  
        NavigationUI.setupActionBarWithNavController(this, navController, drawerLayout)  
  
        NavigationUI.setupWithNavController(binding.navView, navController)  
    }  
  
    override fun onSupportNavigateUp(): Boolean {  
        val navController = this.findNavController(R.id.myNavHostFragment)  
        return NavigationUI.navigateUp(navController, drawerLayout)  
    }  
}
```

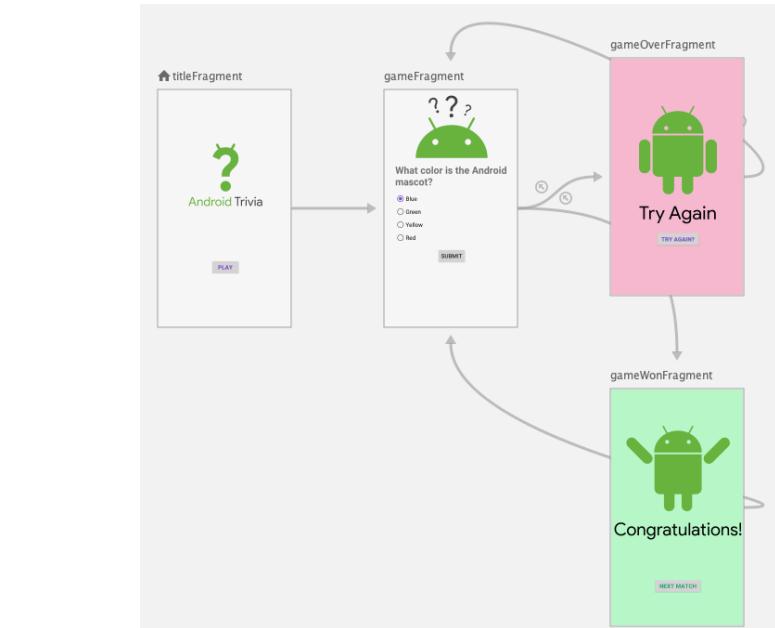
Lien entre activity & layout



app/res/layout/activity_main.xml

Principes de Fragments

- Un fragment est un "morceau" d'interface réutilisable
 - Il est hébergé dans un hôte (une activité)
- On le définit par :
 - Son code (~ comme une activité)
 - Son interface (~ comme une activité)
- On définit la navigation entre fragments au sein de l'hôte



app/res/navigation/navigation.xml

The screenshot shows the "Codelabs" section of the "Android Kotlin Fundamentals" course. It lists several codelabs:

- Android Kotlin Fundamentals: Welcome to the course** (Updated Sep 19, 2019) - Start
- Android Kotlin Fundamentals 01.0: Install Android Studio** (Updated Feb 12, 2020) - Start
- Android Kotlin Fundamentals 01.1: Get started** (Updated Feb 12, 2020) - Start
- Android Kotlin Fundamentals 01.2: Basic app anatomy** (Updated Feb 21, 2020) - Start
- Android Kotlin Fundamentals 01.3: Image resources and compatibility** (Updated Feb 12, 2020) - Start
- Android Kotlin Fundamentals 01.4: Learn to help yourself** (Updated Feb 12, 2020) - Start

Pendant ce temps là, au labo ...

A delta-oriented approach to support the safe reuse of black-box code rewriters

Benjamin Benni¹ | Sébastien Mosser² | Naouel Moha² | Michel Rivelli¹

Abstract
Large-scale corrective and perfective maintenance is often automated thanks to rewriting rules using tools such as Pythontools, JBoss, or Gherkin. Such tools consider these rules as a set of constraints to be checked against the current state of the system, and then applying rewriting rule as input to the next one. It is up to the developer to identify the right order if it exists among all the different rules to yield the right program. In this paper, we define a formal model for rewriting rules and propose a delta-oriented approach to support their reuse. In each rule, leveraging these, we propose a way to safely compose multiple rules when applied to the same program to (a) ensure an isolated application of the different rules and (b) support the reuse of the same rule in different contexts. We illustrate our approach through two on large-scale case studies in identifying conflicts in the Linux source-code automated maintenance and (b) fixing memory artifacts existing in Android applications using Gherkin.

Keywords
code rewriting, conflict detection, rule composition, software reuse

1 | INTRODUCTION
It is now common to state that “software evolves,” and it is part of software developers’ duty to support and operate such an evolution. The adaptive, preventive, and perfective evolution¹ of a piece of software to address new requirements is taken into account by software development methodologies and project management methods.² But this evolution is not related to the addition of immaterialities value in the software, but rather to the evolution of the system and its environment. In this paper, we focus on the perfective evolution (i.e., evolution 2–x), framework upgrade (e.g., supporting upcoming versions of Android for a mobile application), implementation of best practices that change along time (e.g., following version guidelines to review a deployment descriptor), code refactoring (e.g., to reinforce design patterns), or bug fixes.

It is useful to automate the evolution as much as possible, using tools working directly on the source code. These tools can be used to detect conflicts between different rules, but they do not always contain enough information to make it work (e.g., APCharger). For example, migrating from Python 2.7 to 3.6 can be done using the `2to3` shell command.³ This provides 50 functions and implementing a specific evolution made to the Python API. By iterating over a collection, modification made to the `2to3`, shell command can be used to automatically fix the code. However, this tool does not support the reuse of the same rule in different contexts. The tool to obtain a Python 3 compatible version of an initial Python 2 program with respect to the available rules. In a broader way, any up-to-date IDE provides automated refactoring options to ease the work of software developers.

Code reuse is a well-known concept in software engineering that appears when developing Linux drivers. The kernel libraries continuously evolve, and device-specific drivers must be ported to support the new API. To tame this challenge, they develop the `checkpatch` tool,⁴ used to review the code and generate patches automatically applied to the Linux kernel to correct bugs of stated drivers. In a similar way, the Java code reuse is done through the application of patches to code drivers to work with the latest Java version. The Java source code in version was, such as code refactoring, automated bug fixing, or annotation fixing. At runtime, these three tools (i.e., OpenJDK, Eclipse, and IntelliJ IDEA) are used to support the reuse of Java code.



<https://ace-design.github.io/>

Réparation des anti-patrons

- Pourquoi écrire des programmes ?
 - Quand on peut écrire des programmes qui les écrivent à notre place ?

```
1 public class C {  
2  
3     private String data;  
4  
5     public String setData(String s) {  
6         this.data = s;  
7     }  
8  
9     public void doSomething() {  
10        // ...  
11        setData(newValue) /* <<< */  
12        // ...  
13    }  
14}
```

(A) Example of a Java class (C.java, p_j)

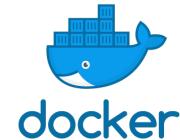
```
1 public class C {  
2  
3     private String data;  
4  
5     public String setData(String s) {  
6         this.data = s;  
7     }  
8  
9     public void doSomething() {  
10        // ...  
11        this.data = newValue /* <<<< */;  
12        // ...  
13    }  
14}
```

$$(\mathbf{B}) \quad p_{igs} = R_{igs}(p_j)$$

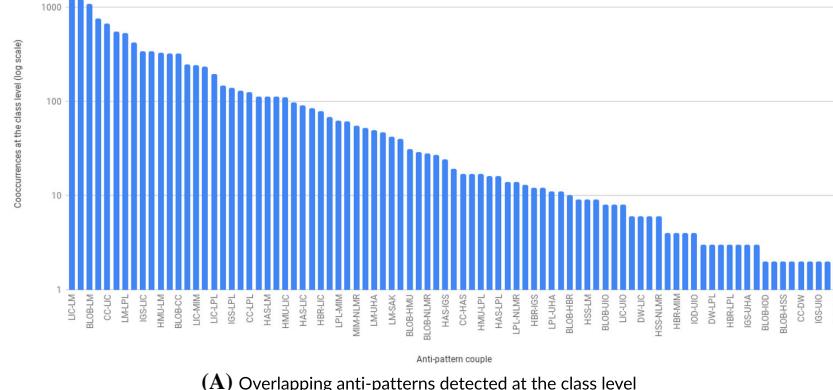
Travail de recherche : Co-localisation

- Comment réparer si deux anti-patrons sont localisé au même endroit dans l'application Android ?
 - Le problème se généralise à tout système de réécriture

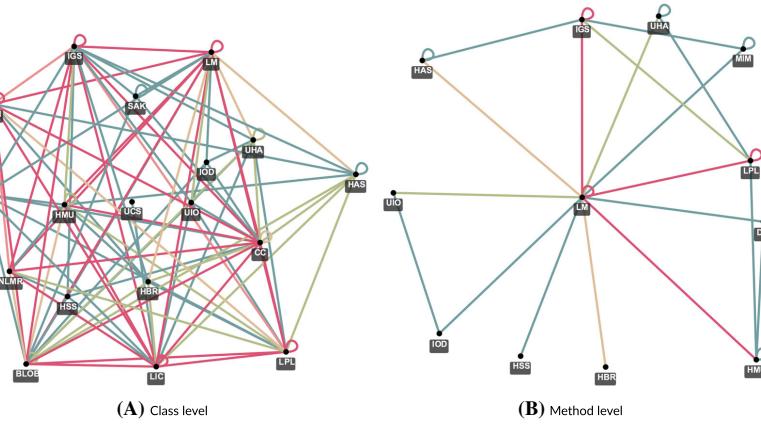
$$\begin{aligned}\oplus : \text{AST} \times \text{A}_<^* &\rightarrow \text{AST} \\ (p, S) &\mapsto \begin{cases} S = \emptyset & \Rightarrow p \\ S = \alpha | S' & \Rightarrow \text{exec}(\alpha, p) \oplus S', \quad \text{exec being language specific.} \end{cases} \\ \ominus : \text{AST} \times \text{AST} &\rightarrow \text{A}_<^* \\ (p', p) &\mapsto \Delta, \quad \text{where } p' = p \oplus \Delta.\end{aligned}$$



Anti-patrons co-localisés



Paires d'anti-patrons



Rejoignez l'équipe !

Faculty



Sébastien Mosser
Professor, UQAM

Researchers



Benjamin Benni
Postdoc, Concordia University

Cats



Jinx
European cat (COVID-19 funding)

Ravioli
European cat (COVID-19 funding)

Graduate Students



Sébastien Bonniew
PhD Student, UCA



Jean-Philippe Caissey
MSc Student, UQAM



Jérémie Fornarino
MSc Student, ESIEE



Pierre Froidevaux
MSc Student, Exia.CESI / UQAM



Rayan Lakhdar
MSc Student, Exia.CESI / UQAM



Lou-Théo Laurent
MSc Student, Exia.CESI / UQAM

Interns

Alison Lecuyer B.Sc (IUT), Université Côte d'Azur
Olivier Levasseur B.Sc, UQAM (NSERC grant)



Sami Lazreg
PhD Student, UCA



Maxime Mulder
MSc Student, Exia.CESI / UQAM



Juliano Augusto Pereira
MSc.A. Student, UQAM



Florian Vouters
MSc Student, Exia.CESI / UQAM



Avril de Goërs de Herve
École Normale Supérieure (ENS) de Lyon

INF6200, Stages, Projets de maîtrise ... Venez discuter !