olivierlegoaer

GDR Groupement de recherche
GPL Génie de la programmation et du logiciel

Université de Pau et des Pays de l'Adour

The road to green code (with *Sonar*)

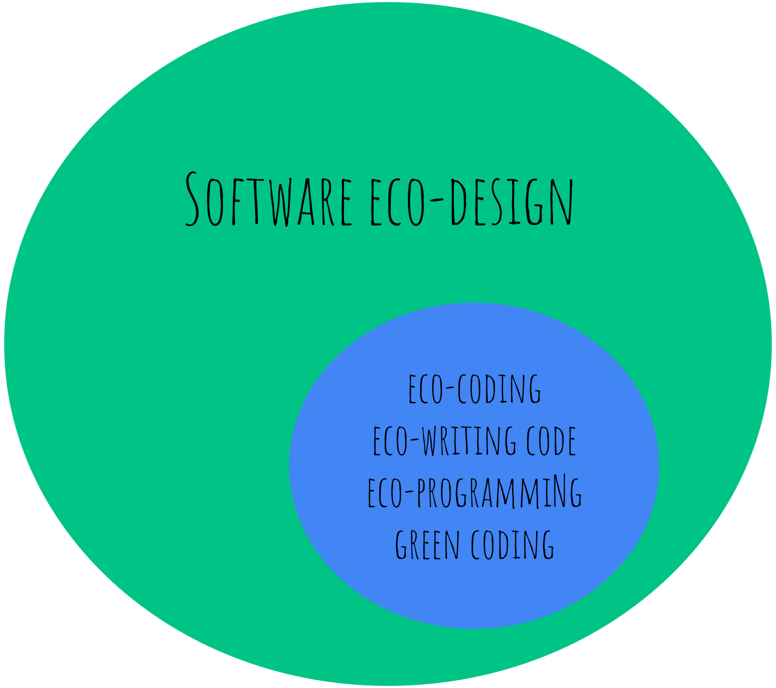
The limits to (software) growth

How it started (2011)

How it's going (2024)

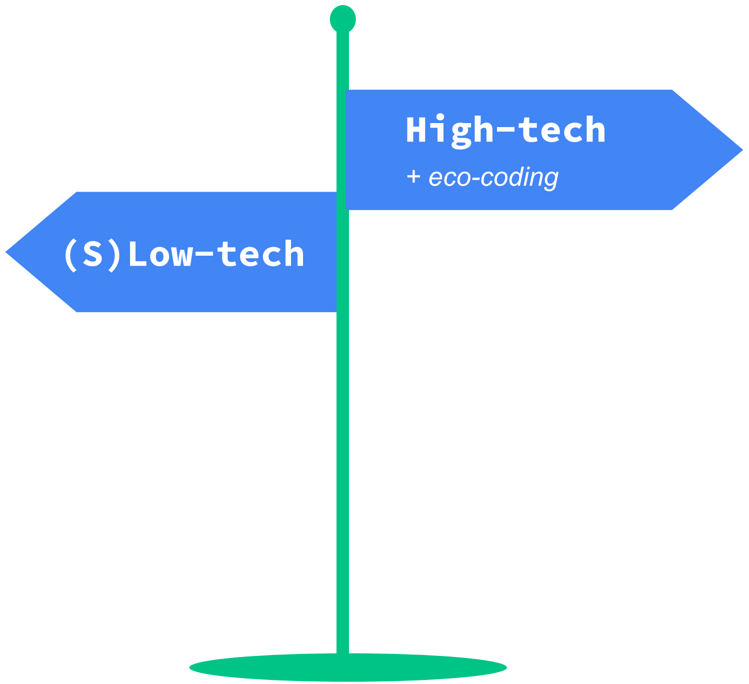
©Tristan Nitot

The software eco-*



"the most responsible software is the one we don't build"

Fork in the road



First law of eco-coding

$$\text{energy} = (\text{more code})^2$$

$$e = mc^2$$

Energy versus Performance

Computer/Device	A	B
Energy (in joules)	30	20
Time (in seconds)	10	20

Energy-efficiency vs Run-time-efficiency

Basic eco-coding incentives

💰 Money

The fewer resources SmartContracts* consume, the lower the costs

```
contract ERC20 is Context, IERC20, IERC20Metadata {
    mapping(address => uint256) private _balances;

    mapping(address => mapping(address => uint256)) private _allowances;

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;
```

⭐ Reputation

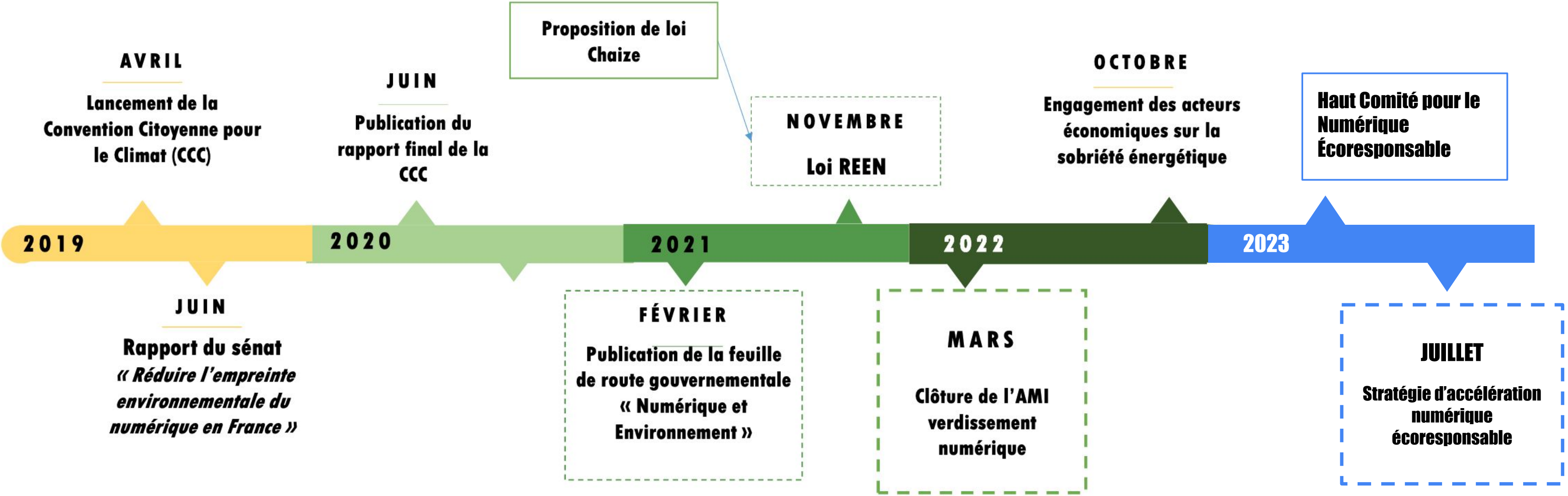
Bad reviews left on app stores can ruin your business

*programs running within a Blockchain

French roadmap

MINISTÈRE
DE LA TRANSITION
ÉCOLOGIQUE
ET DE LA COHÉSION
DES TERRITOIRES
Liberté
Égalité
Fraternité

MINISTÈRE
DE L'ÉCONOMIE,
DES FINANCES
ET DE LA SOUVERAINETÉ
INDUSTRIELLE ET NUMÉRIQUE
Liberté
Égalité
Fraternité



Towards an eco-score...

cyber-score (effective in oct. 2023)

[source]

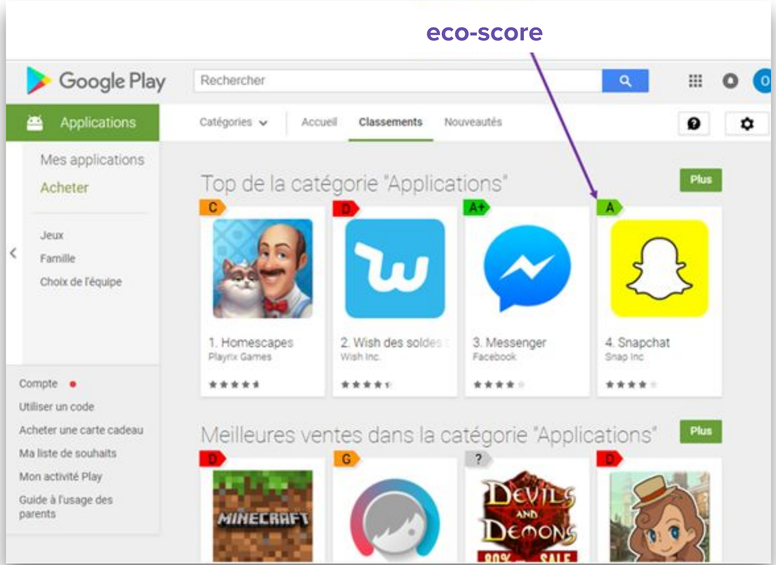
eco-score (elusive goal)

[source]

What if an eco-score?

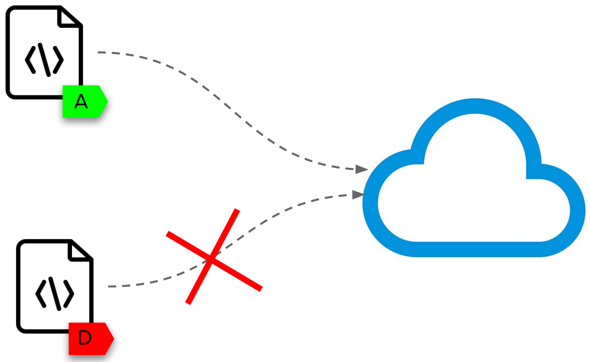
Information

App stores display the eco-score to the end-users (and include it in their ranking algorithm)

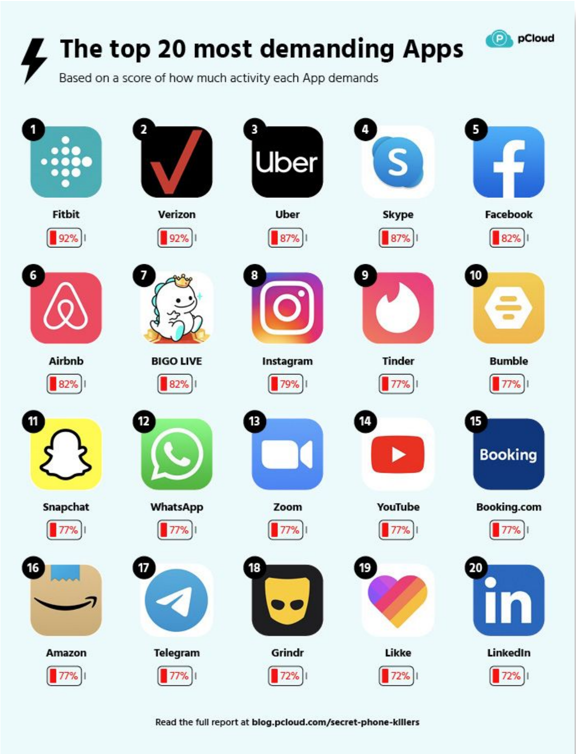


Regulation

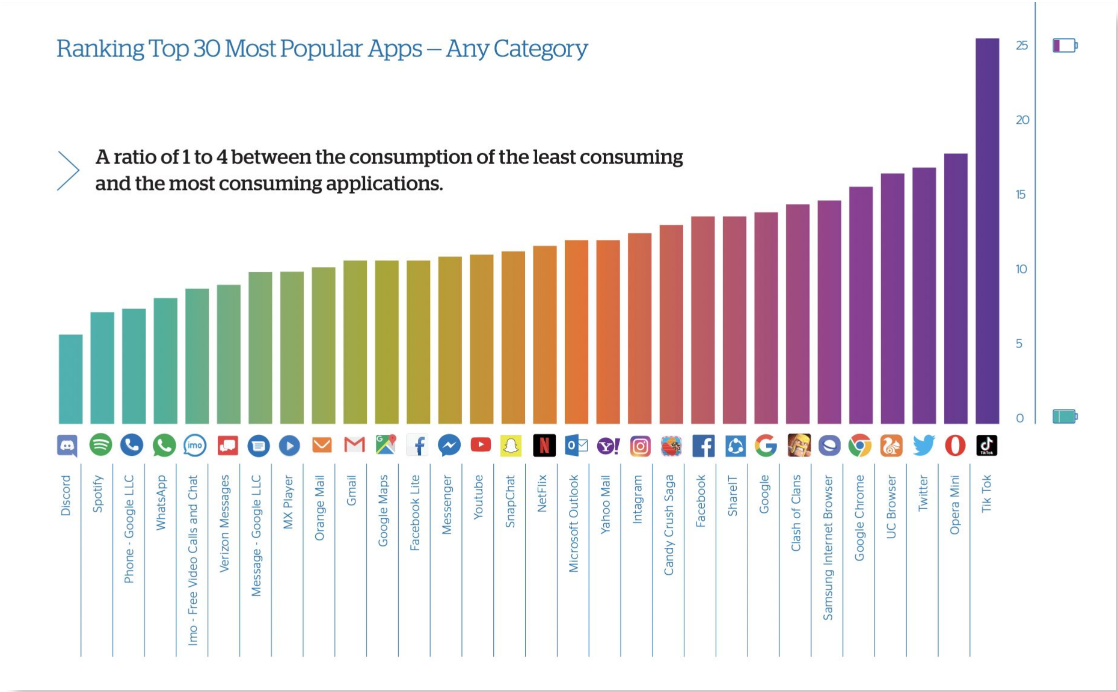
(Truly sustainable) Cloud providers refuse the deployment of program lower than D



Ranking eco-friendly apps

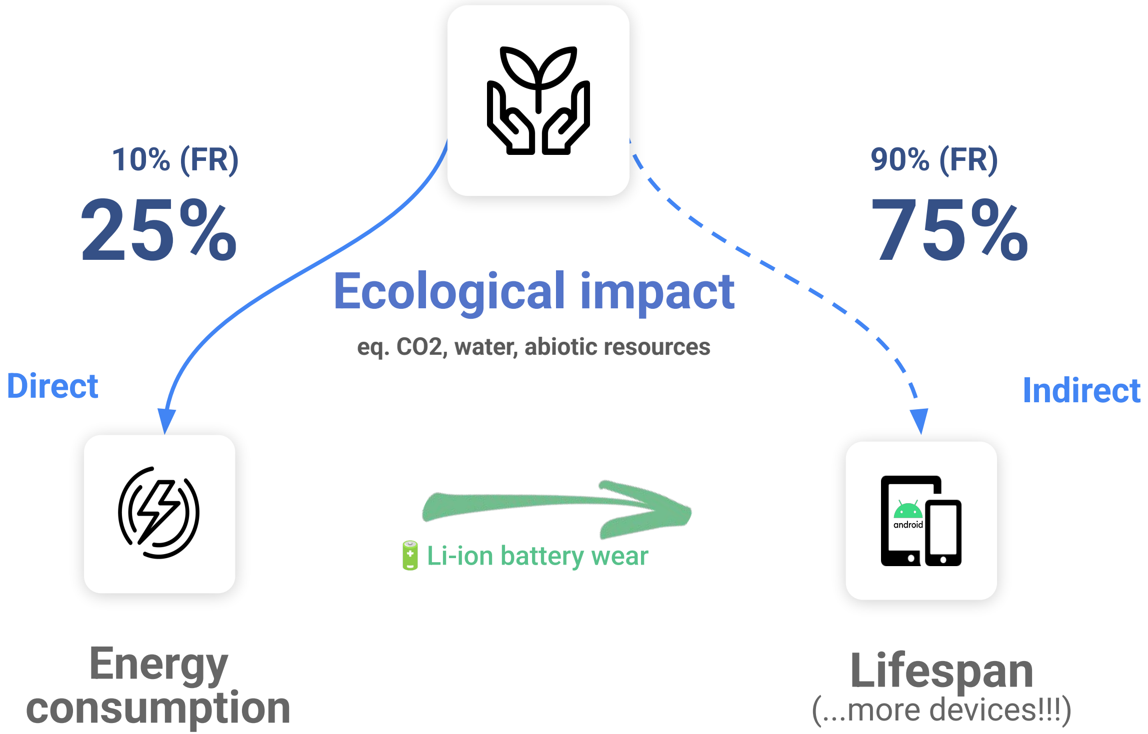


source: pCloud (2021)



source: Atos & Greenspector (2020)

Think global & mobile-first



Loosely adapted from "iNUM : impacts environnementaux du numérique en France" (Jan. 2021)

Limited impacts

Green programming languages?

You don't always have a choice!

What about the runtime?

Mobile apps are programs, but rarely algorithms*

Programming language: there is no silver bullet!

	Energy (J)
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

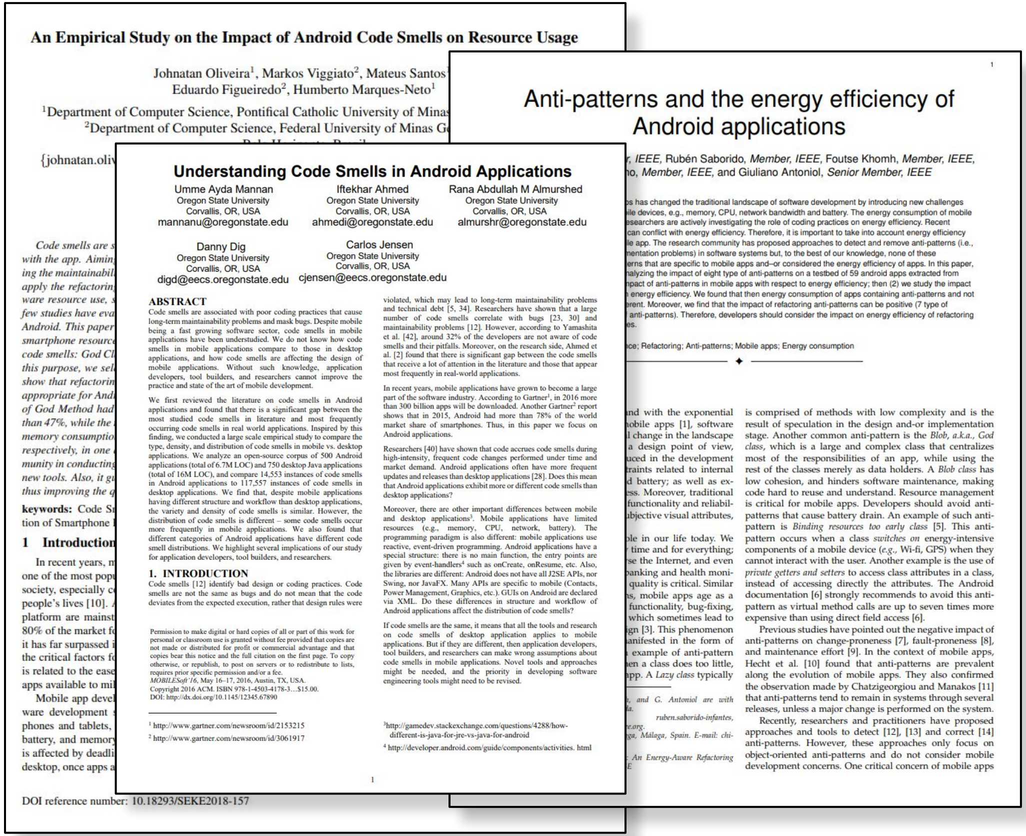
Rui Pereira *et al*. “Ranking Programming Languages by Energy Efficiency”. Science of Computer Programming, volume 205. Elsevier, 2021.

*mathematically provable object

Code smells: The good old classics

- Feature Envy
- God Class
- Blob Class
- Long Method
- Long Parameter List

...


An Empirical Study on the Impact of Android Code Smells on Resource Usage

Johnatan Oliveira[1], Markos Viggiato[2], Mateus Santos[1]
Eduardo Figueiredo[2], Humberto Marques-Neto[1]

DOI reference number: 10.18293/SEKE2018-157

Anti-patterns and the energy efficiency of Android applications

Understanding Code Smells in Android Applications

Umme Ayda Mannan, Oregon State University, Corvallis, OR, USA, mannanu@oregonstate.edu
Iftekhar Ahmed, Oregon State University, Corvallis, OR, USA, ahmedi@oregonstate.edu
Rana Abdullah M Almurshed, Oregon State University, Corvallis, OR, USA, almurshr@oregonstate.edu
Danny Dig, Oregon State University, Corvallis, OR, USA, digd@eecs.oregonstate.edu
Carlos Jensen, Oregon State University, Corvallis, OR, USA, cjensen@eecs.oregonstate.edu

ABSTRACT

1. INTRODUCTION

Code smells: The new challengers

- Internal Setter
- Leaking Thread
- Leaking Inner Class
- Member Ignoring Method
- No Low Memory Resolver
- Hashmap Usage
- Init OnDraw

...

Greater impacts

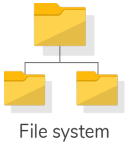
Android-specific matters

Battery-killers are nestled at the platform-level, not the language-level

Every Android project has a well-defined, meaningful structure

There are lots of interesting things to inspect:

Java Gradle XML



Energy-greedy components

Hardware-related Component	Avg. energy consumption (J)
display	139.784567875382
camera	84.1856142588254
microphone	81.8998646885348
gravity	71.3078291080087
magnetic_field	69.6877663025097
gyroscope	69.3777997221
accelerometer	67.9535327322522
cpu	66.6925401713931
room_database	66.0762976599094
speaker	65.6659164078901
gps	65.6478179873468
local_storage	64.5536233840085
ambientlight	63.0030057575923
networking	62.6477966616013

A Framework for the Automatic Execution of Measurement-based Experiments on Android Devices

Ivano Malavolta[1], Eoin Martino Grua[1], Cheng-Yu Lam[1], Randy de Vries[1], Franky Tan[1], Eric Zielinski[1], Michael Peters[2], Luuk Kaandorp[1]
[1] Vrije Universiteit Amsterdam, The Netherlands. i.malavolta@vu.nl, e.m.grua@vu.nl, c2.lam@student.vu.nl, randy.de.vries@student.vu.nl, k.h.tan@vu.nl, e.a.zielinski@student.vu.nl, l.kaandorp@student.vu.nl
[2] M2mobi, The Netherlands. m.peters@m2mobi.com

ABSTRACT

Conducting measurement-based experiments is fundamental for assessing the quality of Android apps in terms of, *e.g.*, energy consumption, CPU, and memory usage. However, orchestrating such experiments is not trivial as it requires large boilerplate code, careful setup of measurement tools, and the adoption of various empirical best practices scattered across the literature. All together, those factors are slowing down the scientific advancement and harming experiments' replicability in the mobile software engineering area.

In this paper we present Android Runner (AR), a framework for automatically executing measurement-based experiments on native and web apps running on Android devices. In AR, an experiment is defined once in a descriptive fashion, and then its execution is fully automatic, customizable, and replicable. AR is implemented in Python and it can be extended with third-party profilers.

AR has been used in more than 25 scientific studies primarily targeting performance and energy efficiency.

1 INTRODUCTION

Android is the leading mobile platform today and the majority of scientific contributions on mobile software engineering focuses on Android [1]. When dealing with quality properties like energy efficiency and performance, practitioners and researchers rely on the measurement of run-time metrics such as battery discharge, CPU and memory usage, number and type of network requests, etc. [7, 9, 10] In this context, considerable effort and time are spent on setting up infrastructures and software pipelines for conducting measurement-based experiments. Moreover, when available, existing pipelines are either (i) ad-hoc for a specific experiment or (ii) tailored to one specific quality property (*e.g.*, energy consumption).

This paper presents Android Runner (AR)[1], a framework to automatically execute measurement-based experiments on native and web apps. The main goal of AR is to *streamline the execution of measurement-based experiments involving Android devices*. In AR, experiments are defined in a descriptive fashion, and then their execution is fully *automatic*, *customizable*, and *replicable*. We designed AR with the following design drivers in mind:

- **Automation**: after an initial configuration, the experiment can be executed without any interaction from the user;
- **Incremental experiments**: AR always persists the intermediate results of the experiment and, if interrupted, it is able to resume it and continue with the remaining runs;
- **Usability**: users define the experiment in a *descriptive* manner, without writing boilerplate code or knowing the internals of AR;
- **Customizability**: users have the possibility to include their own business logic and automated testing tools [6] at specific points within the experiment execution (*e.g.*, before the whole experiment begins, before or after each run, etc.);
- **Profiler independence**: in AR, run-time measures can be collected both via hardware (*e.g.*, the Monsoon power monitor[2]) and software (*e.g.*, Trepn[3]). Profilers can produce different data points and can interact with apps and the Android device in their own way; moreover, AR makes straightforward to use multiple profilers within a single experiment;
- **Experiments replicability**: given the configuration, subject apps, and available Android devices, AR can execute an already-performed experiment with low effort, even if the experiment has been performed by a third party.

We are aware that frameworks like AR must be as accurate as possible and that their accuracy must be independently verifiable. In order to facilitate the validation of AR, we created a set of 27 **benchmarking apps**, each of them stressing a specific hardware component of an Android mobile device, such as its accelerometer, camera, CPU, display, GPS, etc.. We use those apps on a regular basis for validating the accuracy of AR. The full set of benchmarking apps is publicly available[4] (together with their source code) and can be reused by researchers in the area of mobile software engineering, also independently of AR.

The **target audience** of AR includes (i) researchers who need to conduct empirical evaluations of software engineering methods and techniques involving Android apps, (ii) researchers developing new run-time profilers for Android devices, and (iii) practitioners needing to quantitatively assess the quality of their own apps.

The remainder of the paper is organized as follows. Section 2 provides an overview of the proposed framework, Section 3 presents its

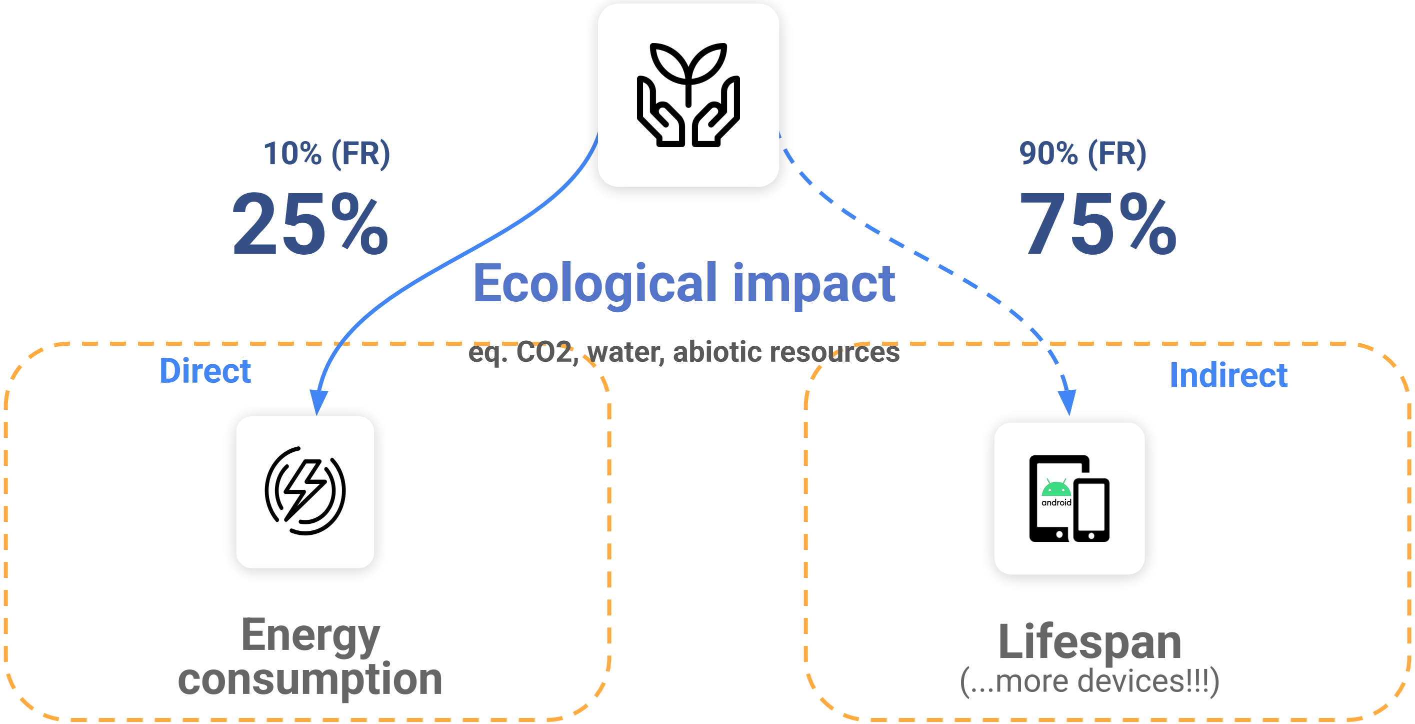
[1] https://github.com/S2-group/android-runner

ASEW '20, September 21–25, 2020, Virtual Event, Australia

ACM ISBN 978-1-4503-8128-4/20/09…$15.00
https://doi.org/10.1145/3417113.3422184

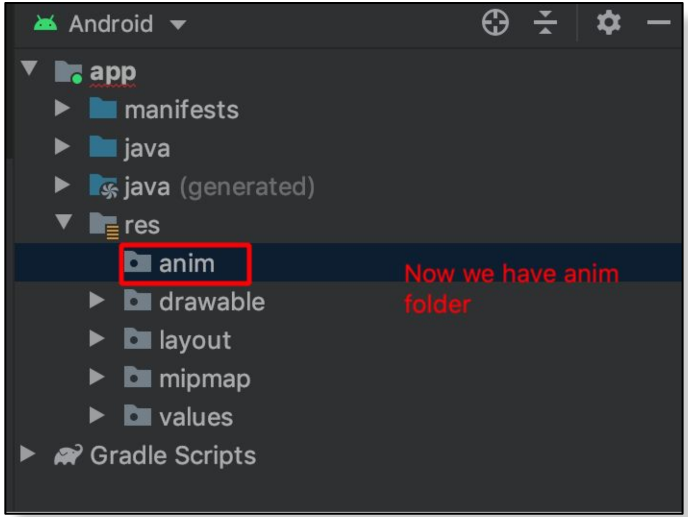
[2] https://www.msoon.com/high-voltage-power-monitor
[3] https://developer.qualcomm.com/forums/software/trepn-power-profiler
[4] https://github.com/S2-group/android-apps-benchmark

Back to the 2 scopes



Scope #1: energy consumption

🪄 Avoid extraneous animation



💤 Avoid keep screen on

```java
public class MainActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
  }
}
```

Scope #2: device lifespan

⚖️ Fight software obesity

```
android {
    defaultConfig {
        ...
        minSdkVersion 15
        targetSdkVersion 33
        multiDexEnabled true
    }
    ...
}

dependencies
    implementation "androidx.multidex:multidex:2.0.1"
}
```

Gradle

♻️ Support aging devices

```xml
<uses-sdk android:minSdkVersion="integer"
          android:targetSdkVersion="integer"
          android:maxSdkVersion="integer" />
```

</> XML

Green code smells

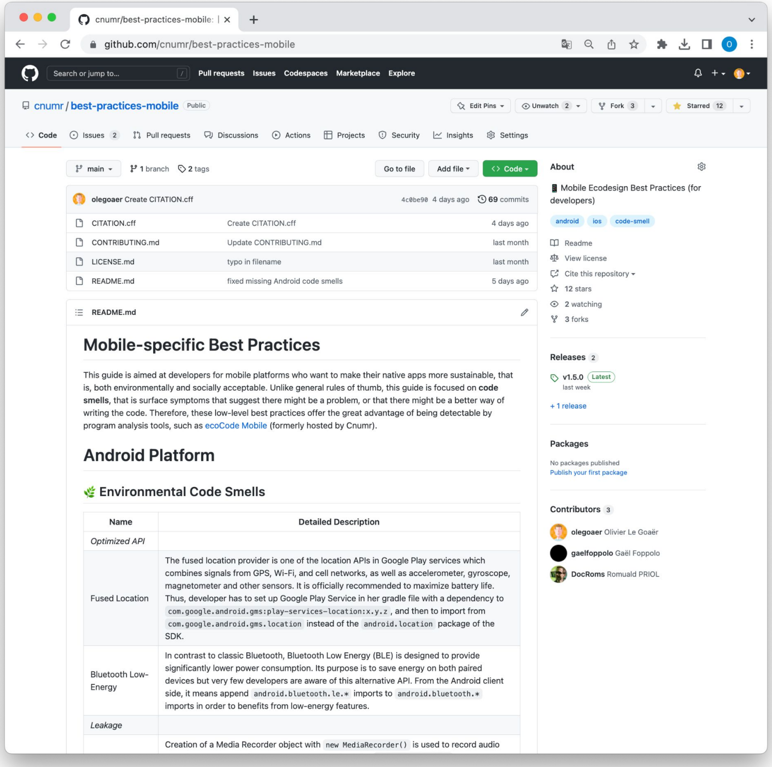
Open source catalog

License CC BY-NC-ND

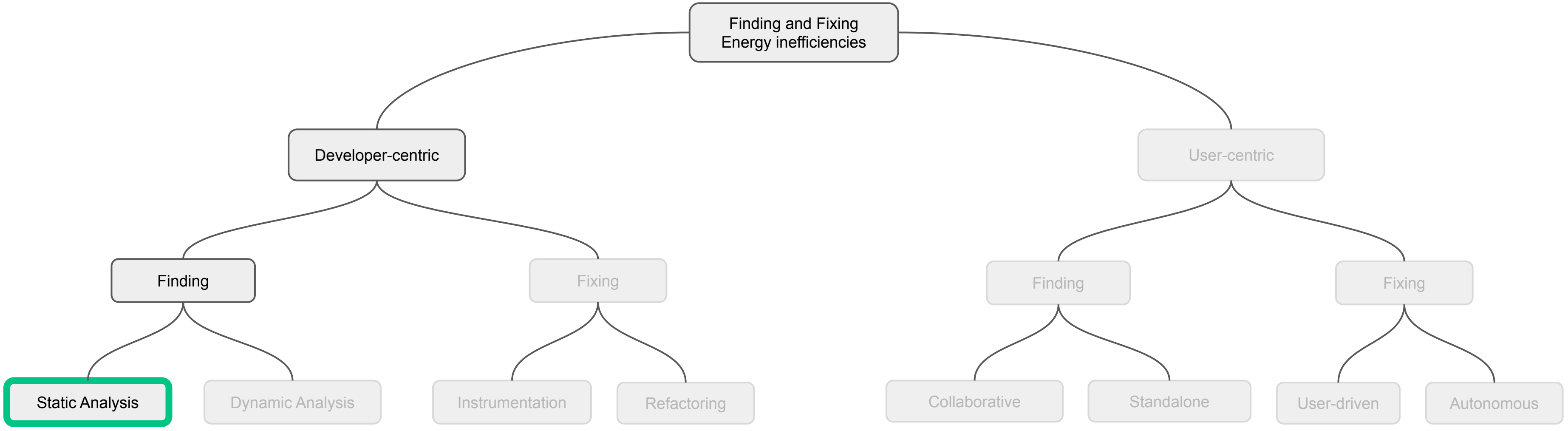
40+ code smells arranged in 9 categories, crosscutting scope 1 & 2

Description based on Java (but Kotlin-ready)

https://github.com/cnumr/best-practices-mobile



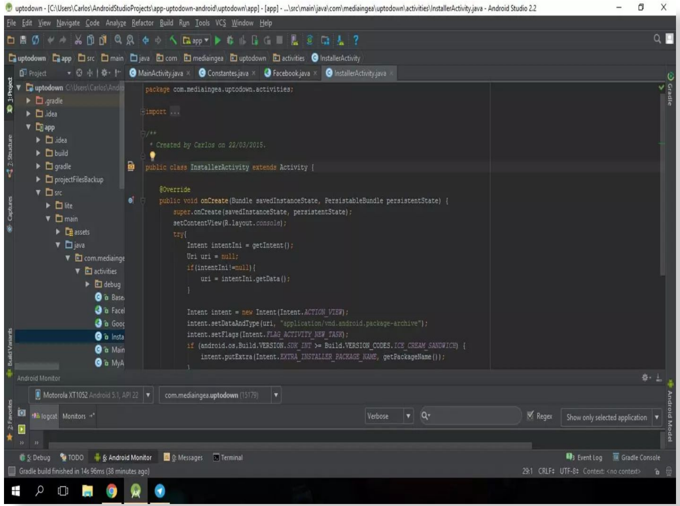
R&D taxonomy

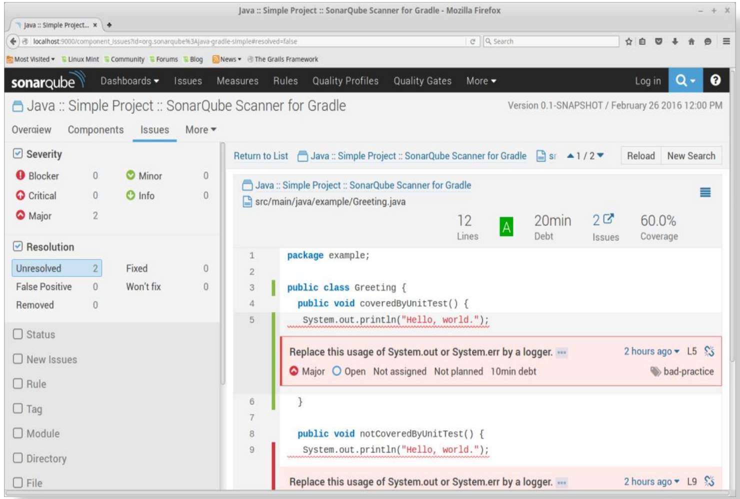


Adapted from Marimuthu C. et al., “Energy Diagnosis of Android Applications: A Thematic Taxonomy and Survey”. *ACM Comput. Surv. 53, 6, Article 117 (February 2021)*

Introducing *ecoCode*

Rationales





android studio

No guidelines on how to write energy-friendly apps

sonarqube

World-class solution to improve code quality

“Green as You Code” sounds good

2-step implementation

It takes 2 simple steps to implement Clean as You Code.

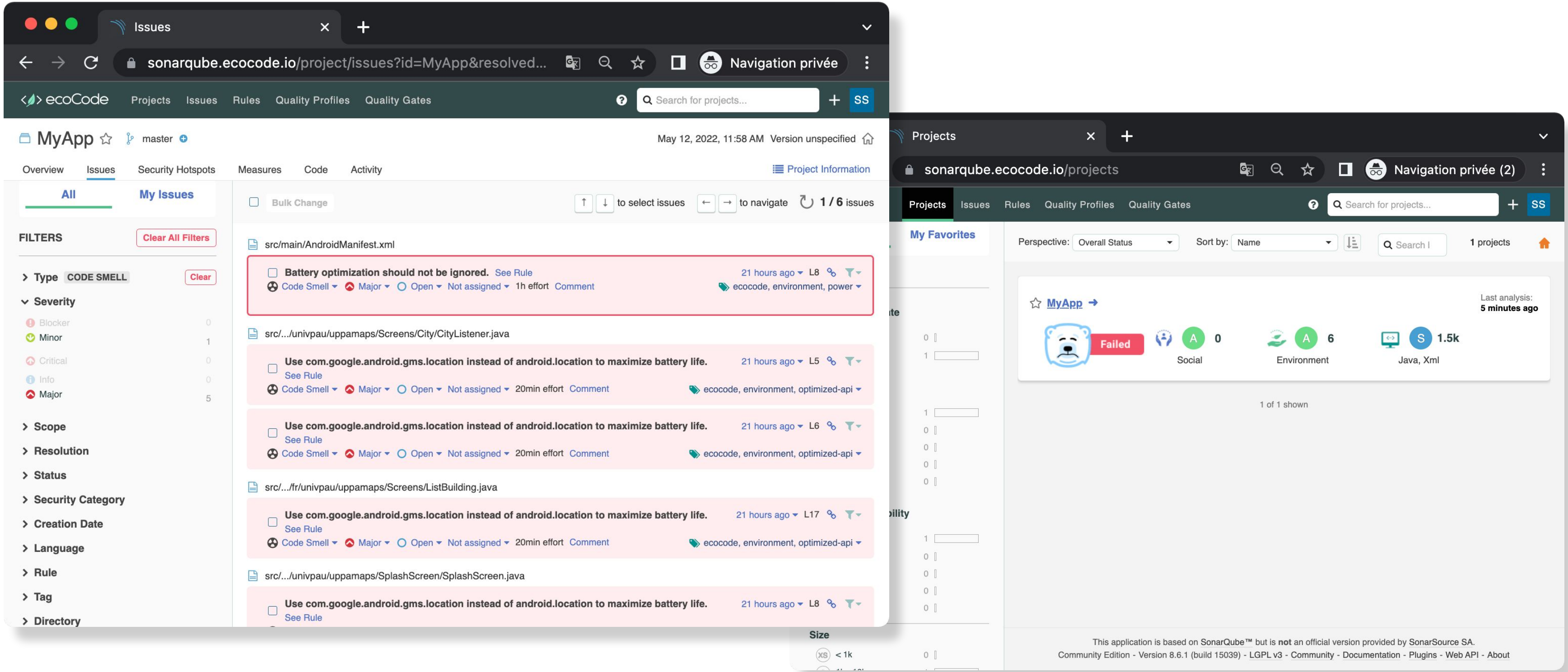
Quality Gate on New Code

A Quality Gate focused only on metrics for New Code – added or changed – prevents new issues from creeping in. Sonar sets this by default and aligns developers across the organization to deliver to that standard.

don’t release unless it’s green

The only rule that needs to be applied is the common organizational understanding that no project will be released to production if it's failing its Quality Gate.

Screenshots*



*self-hosted instance of the ecoCode SonarQube plugin

Technical hurdles

Outdated

Ongoing

Related works

Academic

- EcoAndroid *[Ribeiro et al., 2021]*
- E-Debitum *[Maia et al., 2020]*
- xAL *[Fatimaa et al., 2020]*
- aDoctor *[Iannone et al., 2020]*
- Green Android Lint *[Le Goaer, 2019]*

Non-Academic

- Green Software Insights *[CAST, 2023]*
- EcoSonar *[Accenture, 2022]*
- Greensight Sonar *[Capgemini, 2022]**
- ~~Ecoscan *[Enedis, 2020]*~~

*joined the ecoCode project in 2024

Digital commons

Avoid reinventing the wheel every time

Open Source improves IT sustainability. ecoCode cannot but be OSS

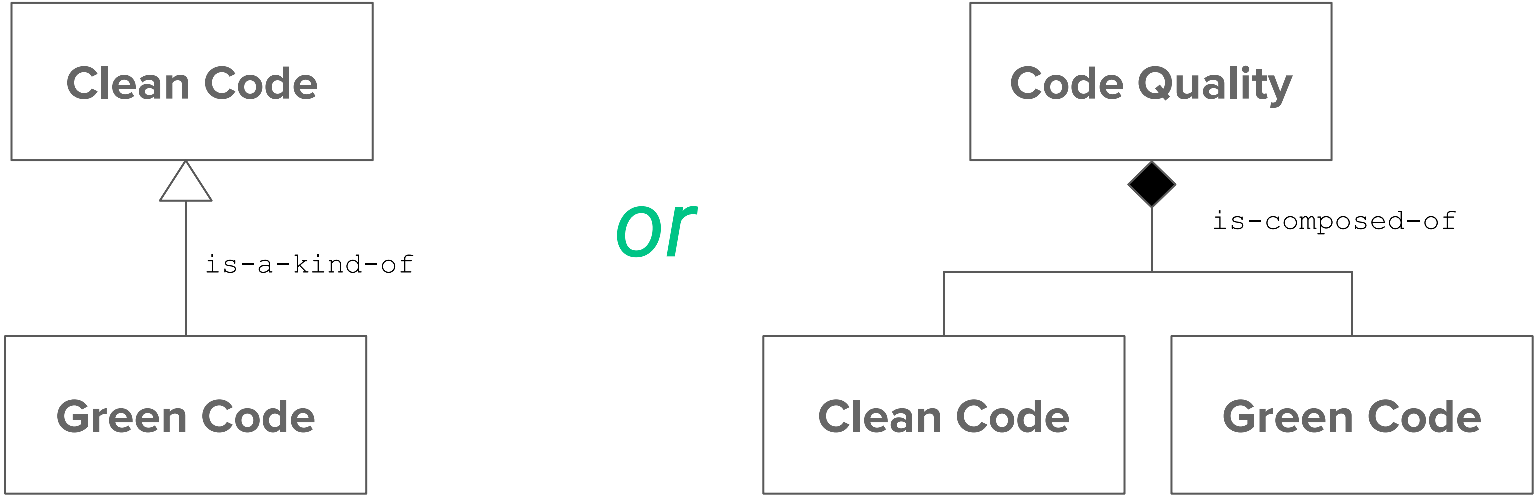
Build a community first (e.g., through hackathons). The lines of code will follow

Many to watch, few to make

Green Code Initiative (GCI)
https://github.com/green-code-initiative

Food for thought

How clean|green code relate?



Green software supply chain

Greening the software is noble, but greening the software supply chain too

Motto: “Wherever there's code, eco-coding is possible !”

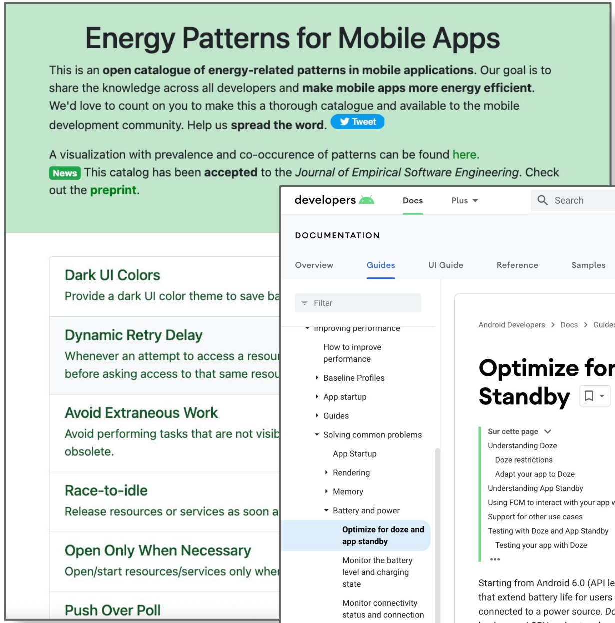
The “Everything-as-code” is a huge potential reservoir of green code smells : infrastructure-as-code, configuration-as-code, platform-as-code, ...

Static analysis: The great filter

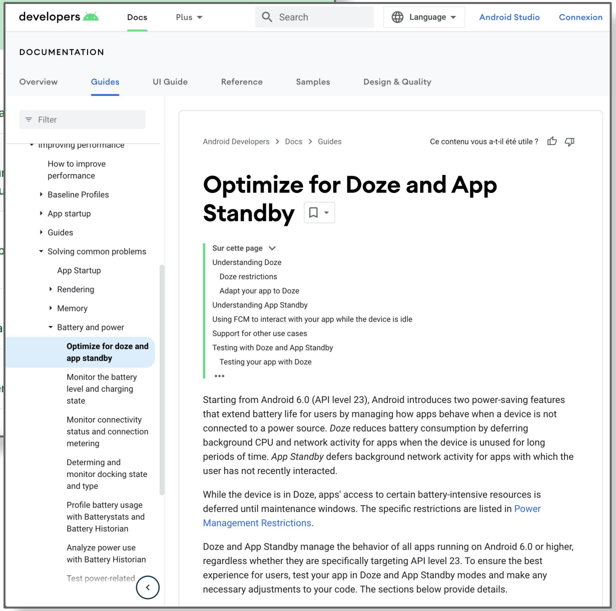
Very few general rules of thumb withstand the filter of static code analysis (“use cache”, “not too much videos”, etc.)

Pro Tips: Must be rooted at syntax-level

Bottom-up approach is the preferred way to find new rules/patterns/best-practices



source



source

Static analysis: A noble art

Challenges

🛑 Post-processing

🔎 Cross-scanning

🚩 False positive/negative

Opportunities

🎨 Taint analysis

🕸 Call Graph/Control Flow Graph

🧠 Machine-Learning

Pain point: the evaluation. But...

Unlike 90’s OOP code smells, green code smells are still in their infancy

Do not expect green code to do what clean code has barely done

Sometimes common sense is enough

Ever-evolving mobile platforms makes things even more challenging

Round-trip engineering

ecoCode

Joular

PowDroid

ETSdiff

design-time

run-time

UX/UI

“Wow effect” is important to attract early-adopters

Our revamped UI was hard-coded. Tailoring the SonarQube UI to green-specific concepts would require diving deep

Developers can find green code burdensome. Gamification can help (to engage and reward)

The end.