Decarbonizing Software with Free and Open Source Software: The ecoCode Project

Olivier Le Goaer

Decarbonizing Software with Free and Open Source Software: The ecoCode Project

Olivier Le Goaër [1]

[1] *Universite de Pau et des Pays de l'Adour, E2S UPPA, LIUPPA, Pau, 64000, France*

Abstract

ecoCode was an applied research project funded for 3 years by the Nouvelle-Aquitaine region (France) and has successfully passed its milestones, from proof of concept to minimum viable product. Today, the project is self-sustaining thanks to a growing community of volunteers from the software development sector. The software business has high expectations for the ecoCode project, which aims to apply the proven principles of code quality to the energy efficiency, and therefore the incidental carbon footprint, of software. As such, ecoCode extends the world-class SonarQube solution to detect a new category of code smell: green code smell. Originally designed to decarbonize mobile software, it now covers a broader spectrum.

Keywords

code smell, carbon footprint, code quality, open source, ecology, sonar

1. Introduction

ecoCode (a contraction of the words "ecology" and "code") is a research project designed to be industry-oriented. Its basis is the assertion that energy efficiency is a quality attribute in the same way as safety or maintainability, for example. Since its inception, there has been a deep conviction that this attribute will become the focus of attention as the issue of climate change arises in all areas of society, including the software engineering [1]. In other words, energy efficiency is a non-functional property that is underestimated, creating a new form of technical and ecological debt that must be solved. But for this to happen, it will be necessary to provide computer-aided code analysis capabilities given the billions of lines of code that make up our modern world, "eaten by software" (sic).

In the age of massively distributed computing, there is obviously no one-size-fits-all solution to cut the carbon footprint of software. So, choices have to be made, mainly driven by the orders of magnitude involved. Among the targets to be decarbonized as a priority, well ahead of programs running in the cloud (server-side), is mobile software (client-side). Indeed, the active user base is huge (close to 7 billion), the panel of available apps seems almost unlimited (close to 9 million), and consequently the number of lines of code under the hood is gigantic. Keep in mind that some popular applications are executed by millions of people for hours every day. It's thus easy to see why saving a few microwatts locally on a smart device makes sense on a global scale. In this area, Android is by far the most popular platform in the world, with a market share of almost 72%. For all these reasons, ecoCode was initially set up to analyze android apps prior to their publication on the Google Play store.

Thus, the rationale behind the research project is that a source code may contain potential issues related to excessive power consumption (OLED display, sensors, network, CPU, I/O, RAM, etc.). If this assumption is correct, it should be possible to automatically detect - and fix - them. This gives rise to the concept of green code smell, which can be defined as follows:

RPE@CAiSE'23: Research Projects Exhibition at the International Conference on Advanced Information Systems Engineering, June 12--16, 2023, Zaragoza, Spain
EMAIL: olivier.legoaer@univ-pau.fr
ORCID: 0000-0001-5405-1688

"Green code smells, also known as green anti-patterns, are poor design or implementation choices that affects the carbon footprint of the program."

The work began in 2019, with an extension to the linter built into Android Studio IDE to detect this new kind of smell (called a check in the jargon) [2]. The number of checks implemented at the time was limited, but the on-the-fly inspections within Android Studio were very promising. Funded by Nouvelle Aquitaine, this proof of concept then evolved into the more advanced solution ecoCode, based on SonarQube [3]. The thread is therefore always organized around 2 work packages (WP):

- WP1: A curated catalogue of green code smells, independent of any program analyzer
- WP2: An operational static program analyzer for highlighting smelly code structures

In the rest of this document, dive into the results obtained, some lessons learned about open source and future research directions.

2. Empirical code smells catalogue (WP1)

Creating a catalogue of green code smells from scratch has been a long-term task since 2019. It is currently available under a CC BY-NC-ND license at https://github.com/cnumr/best-practices-mobile

The sources of knowledge used to compile the list of code smells were varied. By far the most reliable source is the Android API reference documentation itself. Then there are research papers that have attempted to create lists that are unfortunately difficult to detect statically, or that have looked at some very specific aspects related to energy. Finally, the experience of experienced developers has been gathered from popular blogs and forums such as Stack Overflow. The result is a unique catalogue dedicated to native Android developers (the same work is underway for iOS). At the time of writing, the catalogue offers more than 40 green code smells, arranged into 8 categories for easy reference. Each code smell is defined by a short name and a description. This description makes direct references to elements of the Android API and/or SDK. For the sake of brevity, this article provides only a brief overview in Table 1. Interested readers can refer to the GitHub link above. Another originality of this catalogue is that it is based on inspections that go beyond the pure programming code written in Java, because an Android project is more than that.

Table 1
Overview of the green code smell catalogue

Category	Code Smell
Optimized API	Fused Location, Bluetooth Low-Energy
Leakage	Media Leak, Sensor Leak, Everlasting Service
Bottleneck	Internet In The Loop, Wifi Multicast Lock, Uncompressed Data Transmission, Uncached Data Reception
Sobriety	Dark UI, Day Night Mode, Brightness Override, Thrifty Geolocation, Thrifty BLE, Thrifty Motion Sensor, Thrifty Notification, Vibration-free, Torch-free, High Frame Rate, Animation-free
Idleness	Keep Screen On, Keep CPU On, Durable Wake Lock, Rigid Alarm, Continuous Rendering, Keep Voice Awake
Power	Ignore Battery Optimizations, Companion in background, Charge Awareness, Save Mode Awareness, Battery-constrained Work
Batch	Service@Boot-time, Sensor Coalesce, Job Coalesce
Release	Supported Version Range, Same dependencies, Duplicate dependencies, Fat app, Convert to WebP, Clear cache, Shrink Resources, Disable Obfuscation

3. Code smell detection with Sonar (WP2)

The ecoCode project is divided into two sibling components: ecoCode mobile (the main subject of this document) and ecoCode standard to deal with non-mobile software programs. They are now maintained under an organization named Green Code Initiative (GCI) under the GPL-3.0 license and are available at https://github.com/green-code-initiative

As mentioned previously, the project has shifted from Android Studio IDE to SonarQube, which is maintained by SonarSource, a world leader in code quality. SonarQube has historically focused on maintainability and security, both of which are subsumed by the concept of clean code. Therefore, ecoCode brings the missing piece by addressing energy efficiency under the green code heading. As an extension of SonarQube, ecoCode takes advantage of built-in features, including robust code parsers, user-friendly web dashboard and the SQALE quality model (quality gates, remediation costs, etc.). Figure 1 shows the result of a code analysis on a native Android project written in Java.

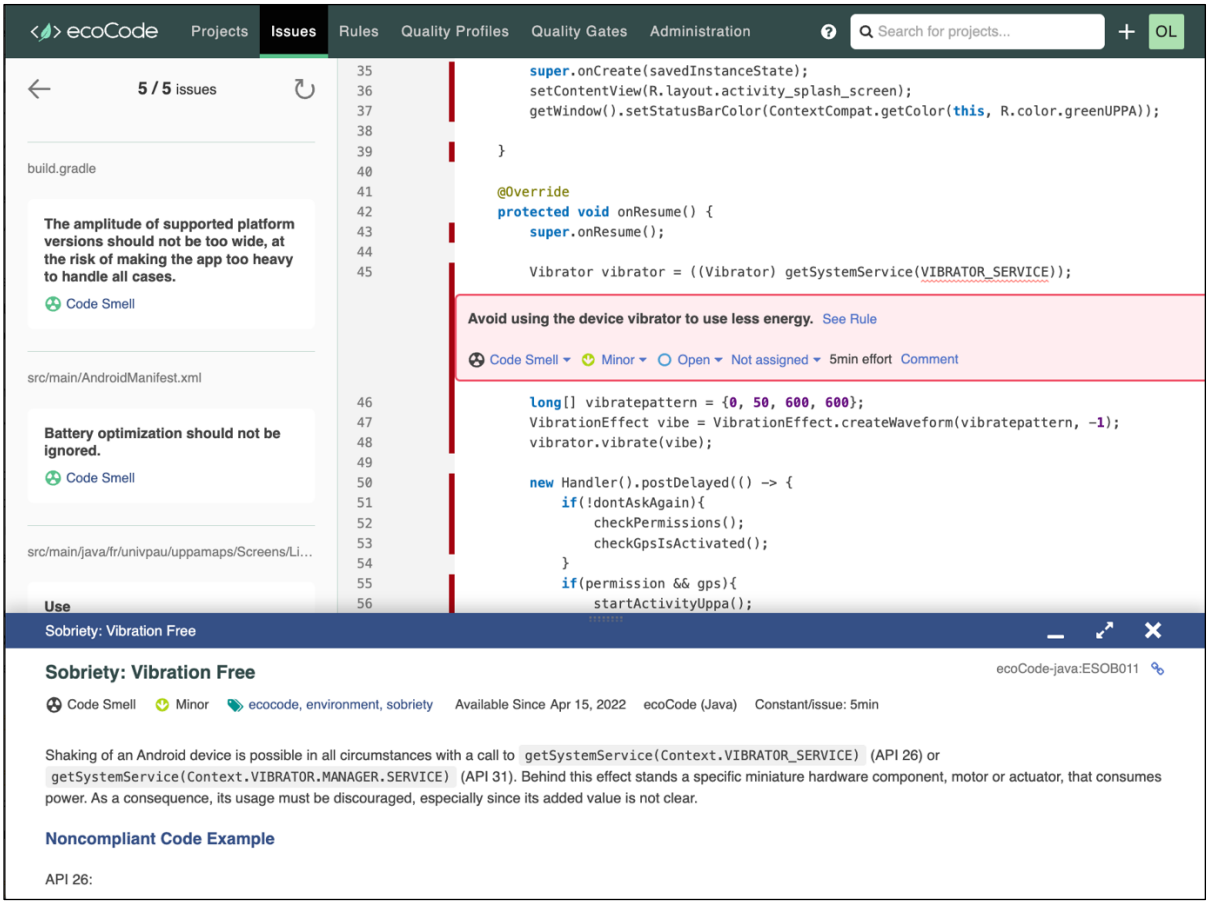

Figure 1: Screenshot of code smells detection

Unlike the basic SonarQube product, which supports around 20 mainstream languages, when building a plugin, only 5 code scanners are available: Java, Xml, Cobol, Php, Python. As far as native Android development is concerned, the lack of Kotlin is problematic (unresolved at present) and support for Gradle scanning (actually groovy) was achieved by a cumbersome technical trick. On this basis, 30 green code smells from the catalogue were successfully detected using the Sonar plugin API, where the detection rules are exclusively written in Java. The remaining smells are only a matter of time.

Careful readers will have noted that we developed a GUI on top of the genuine SonarQube GUI in order to tailor the concepts of clean code to those of green code and thus provide a unique user experience. However, this whimsical overlay has not been released as open source because it breaks compatibility when Sonar is upgraded and also because in practice companies don't care.

4. The power of the open source software

The ecoCode project was destined to be released as open source for two reasons. The first, very pragmatic, is that it is generally accepted that open source is good for the sustainability of software in general. In this context, it would have been contradictory for the ecoCode solution to remain proprietary (in this case, partly owned by the University of Pau). The second is the nature of the project itself. Best practices are evolutionary and require collaboration to stay up to date and meaningful. As far as the static analysis of the code is concerned, this is undoubtedly a complex engineering piece that requires a very large number of man-months and long-term technical support that only an active open-source community can provide once the funding is complete.

4.1. GitHub to the rescue

GitHub has established itself as a first-class infrastructure for hosting research projects. ecoCode is no exception. Its features are adapted to this type of project and have allowed to support 2 hackathons in a row (June 2022 and April 2023) with many diverse contributions. To give an idea at the time of writing, the number of lines of code behind the ecoCode project is shown in table 2. More surprisingly, GitHub turned out to be the ideal tool for growing the catalogue of code smells. In 2020, we built a dedicated web platform in the hope of collecting ideas from experienced Android developers. But it was a flop. In fact, GitHub offers everything you need to put collective intelligence into action: contributors can propose eco-friendly practice drafts (a.k.a Pull Requests), which may or may not be merged into the main catalogue after discussion. Entirely new categories of code smells can even be proposed in a separate branch before being merged into the main branch.

Table 2
Code complexity at glance (external projects excluded)

SonarQube Plugin	Java #LoC	Misc. #LoC
ecoCode Mobile	56 927	36 383
ecoCode Standard	5 637	4 464

4.2. Educational purpose

Green code is still an emerging concept compared to clean code. In this regard, education plays a major role. Education of developers first, who use code quality tools to progress and improve their code continuously. Highlighting green code smells in a dashboard does not necessarily mean they will fix everything in the current project, but that they will be remembered for future projects. Education of students then, as the next generation of developers, to integrate this new concern as of now. This is a clear departure from the generations that preceded them, for whom the question of global limits and sustainability never arose clearly. ecoCode is also a free and open-source software for that reason.

5. The future of ecoCode

The ecoCode project is still at an early stage and several research and engineering hurdles still need to be overcome. Below are four main ones.

5.1. Assessment through measure

According to Sonar and various partners, what hinders the adoption of ecoCode on a large scale is the lack of precise evaluation of the green code smells. In fact, even if stakeholders initially agree on the relevance of the smell catalogue, they want to be able to focus their efforts in the right places in

their code base. Measuring the impact of each smell on energy consumption individually, but also their impact when combined, is a time-consuming task. This task is further complicated by the lack of energy measurement tools (especially on mobile platforms) and associated methodologies. This research will be a turning point for the project.

5.2. Green code smells into the wild

Another practice that stems from Sonar's internal policy is to confront every new smell to real-world projects. Indeed, there is no evidence that this or that green code smell (e.g., Internet In The Loop) occurs frequently in developments teams. The idea behind this is that a large number of open source code repositories can be automatically scanned with ecoCode and the reports can be compiled. A tool like SonarQube allows this kind of job pipeline without involving a human in the loop. We hope to find out whether developers are already applying green coding practices or whether they are poorly educated in that topic. We think this study will provide a very useful insight.

5.3. Extended detection support

ecoCode's reach is constantly challenged. Internally first, when considering a specific domain like mobile platforms, it quickly becomes headache to cover the reality of modern development: Java, Kotlin, Xml, Gradle (now writable in Kotlin), Swift, Objective-C, but also cross-platform frameworks such as React Native (JavaScript), Kotlin Mobile Multiplatform and Flutter (Dart). External then, when targeting all the facets of software where energy savings can be nestled: web front-end (HTML, CSS, JS), web back-end (JS, PHP, Java, SQL), infrastructure-as-code (YAML, Json, Xml), IoT (C/C++), and so on. All this along the fast-paced evolution of languages and APIs. This is a huge undertaking.

5.4. Advanced detection techniques

Static code analysis based on an abstract syntax tree quickly shows its limitations. Even though this is the technique used worldwide for clean code, ecoCode plans to go further for green code. In fact, certain proven energy-saving practices (e.g., implementing a caching mechanism) are not detectable as they are, which is a bit frustrating. In the era of machine learning, this kind of complex code structure could be spotted. Without going that far, SonarQube offers the ability to create a control flow graph (CFG). It is clear that some of the current green smells would be better detected this way.

6. Acknowledgements

The ecoCode project (www.ecocode.io) was funded by a grant from the Nouvelle Aquitaine region (2020-2022), in partnership with the University of Pau and Snapp', a software company. This work would not have been possible without the tireless commitment of the members of the Green Code Initiative. Special thanks to Credit Agricole, Davidson Consulting and Sonar for their kind support.

7. References

[1] Stefano Forti, Uwe Breitenbücher, and Jacopo Soldani. 2022. "Trending Topics in Software Engineering". SIGSOFT Softw. Eng. Notes 47, 3 (July 2022), 20–21.

[2] Olivier Le Goaër. 2021. "Enforcing green code with Android lint". In Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE '20). Association for Computing Machinery, New York, NY, USA, 85–90.

[3] Olivier Le Goaer and Julien Hertout. 2023. "EcoCode: a SonarQube Plugin to Remove Energy Smells from Android Projects". In Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22). Association for Computing Machinery, New York, NY, USA.