Behavioural code analysis

Software Architecture - 2021/2022

Álvaro Rodríguez González
Mario Lombas Calderón
Juan A. Torrente Bermúdez

1. Technical debt? Not really technical.

What we call technical debt occurs whenever we decide to rush in new features or functionalities in order to achieve a quicker deployment, instead of taking the necessary time to plan, design and implement it. Taking this into account, we can see that debt is usually caused in a higher rate by deadlines and restrictions imposed by "business people" than by technical inability. It is, ultimately, a business decision.

However, we do concede that technical debt is often interpreted and used in technical contexts, and can thus become a problem when thrown around without a technical overview.
More often that not, debt will be one more symptom of structural problems in our company or organization: when a developer team can not efficiently assign tasks among themselves or receive new orders "from above", development processess will move more slowly and less profficiently, forcing the individual developer to take shortcuts in order to meet many deadlines. In addition, technical debt will be generated when developers do not understand or agree with other workers' code, as they will need to spend time getting to understand it and working with it themselves.

Now, while all of the above is true, we must take into consideration that the concept of "technical debt" is still useful, for it gives us a exact term to convey information to stakeholders. Our purpose, nonetheless, will be to find an adequate way of reviewing and analyzing code whenever static code analysis just does not meet our expectations.

2. Goal

We, as a company, need to allocate oour resources in the most efficient way possible, so as to save production costs and increase profits and product maintainability. As we will see further on, static code analysis usually does a poor job analyzing codebases taking into consideration the whole context which led that codebase to reach its current state.

Behavioural code analysis is an innovative way of analyzing this context, allowing companies to understand where to allocate said resources.

3. Definition

Verbatim, "behavioral code analysis identifies patterns in how a development team or organization interacts with the codebase they are building". That is, while the code is important, there’s even more value in learning how the code got that way and where it is trending.

This information is used to prioritize technical debt, detect implicit dependencies that are invisible in the code itself, and measure organizational factors like knowledge gaps and support on- and off-boarding.

4. Considerations

There are two main factors which lead us to detect a problem in our codebase, in a sense that static code analysis could never provide:

a) **Change frequency.** In identifying what we call "hotspots" (pieces of code frequently modified) we can understand that a specific part of our codebase needs a whole rewrite, to allow improvements and modifications to be made with less effort. Otherwise, too much of our developers' valuable time is spent in fixing modules which seem to be permanently broken.

b) Single-person dependency. On projects involving more that one person, it is important to always share and swap responsibilities among members, to avoid having a piece of code dependant on a single developer. Otherwise, we are incurring in a huge risk, as the Bus Factor might do a lot of damage to our organization in the case the developer goes missing.

Behavioral code analysis tools can analyze a project's history and identify these hotspots and knowledge islands.

5. Tools

There are two main tools we can make use of to perform behavioural code analysis: Code Maat and CodeScene.

1. Code Maat (https://github.com/adamtornhill/code-maat) was developed as a material for a coursebook titled "Your Code as a Crime Scene". It is a FOSS command line tool which can import, interpret and analyze a project's history through its VCS history.

2. CodeScene is a commercial tool which evolved from Code Maat. It provides a simple and intuitive graphical interface, which can convey information in a simple way to the business people who, in the end, run our company.

   This is a more complex tool than Code Maat, and allows for specific VCS integration since the birth of a repository. That way, it can perform the data mining it requires in order to analyze commit history and ownership to identify the hotspots we reviewed earlier, both in frequency of change and assignment of responsibilities.