Behavioral code analysis

Juan A. Torrente Bermúdez - UO277963

Álvaro Rodríguez González - UO277776

Mario Lombas Calderón - UO275901

Technical debt is not Technical

"Technical" debt occurs when we choose to hack some new feature in a non optimal way in order to deploy faster. It is ultimately a tradeoff.

As debt is more determined by deadlines and other conditions imposed by project directors, it is more a business decision than a technical one.

Reckless debt

- As debt is not technical, but it is used in a technical context, it may become a real problem when used without a technical overview.
- If you walk into a minefield, but say “well, it is not technically a field. There are no crops!” and go running right through the middle, most likely you will end up dead.
- It is possible to incur in this debt in a way that generates neither long-term nor short-term advantages.
- It is important not to confuse Technical debt with legacy code.

Debt = Organizational problems

"What one programmer can do in one month, **two programmers can do in two months.**" - Fred Brooks

Debt is really a symptom of organizational problems.

Although they are not the only kind of problem, a lot of them come from “My merge had a lot of problems” or “This part of the code Vadim wrote is utterly incomprehensible”

So, hold on a sec. Technical Debt is neither technical nor debt, **why** do we call it like that?

It is a good way to convey information to stakeholders. It is not debt, but it works a whole lot like it.

Also, we are technical people, so the name stuck.

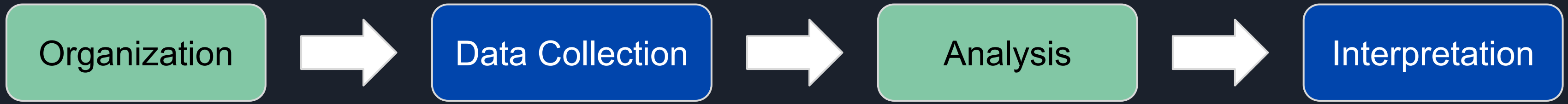
However, it is clear that just static code analysis just does not cut it.

Goal

The development and maintenance of the software cost resources, which must be estimated before an investment. By default, purely static code analysis is used.

This is sufficient for an evaluation of the code quality, but does not provide satisfactory answers to some questions.

By using behavioral code analysis strategies, companies can allocate necessary resources more efficiently.



Definition

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.

How much maintenance is really required?

Old code is bad? No. In fact, code that hasn’t been changed for a long time is the best code.

So, new code is bad? No. Frequent changes demand effort, but the developers know the code and can make adjustments quickly.

Code that is changed occasionally is problematic.

So, what is the best combination for our project?

Do measures need to be taken to improve code quality?

Poor code quality increases maintenance and development efforts.

Code quality is most relevant where there is a lot of ongoing work, and almost irrelevant in places of the code that have not been changed for years.

As part of the code analysis, we evaluate two factors:

- The current code quality in the part that is changed regularly.
- The trend for the entire code base in terms of quality.

How dependent is the code on specific developers?

Often the are parts of the source code which are created and maintained almost exclusively by one person. This may cause some trouble if the developer is absent for a long period.

Also this risk may be increased by poor code quality and insufficient documentation.

As part of behavioral code analysis, we identify knowledge islands, assess the risk and derive measures to counteract it. For example, the targeted development of documentation.

In which places is it worth improving code quality?

There are some parts (hotspots) that demand a lot of effort from developers are made.

So, what is a hotspot?

In the behavioral code analysis we identify hotspots, evaluate the effects and derive countermeasures. In this way, the code quality can be improved at the most important points.

Tools: Code Maat

- Free, open source command line tool able to analyze data from a VCS.
- https://github.com/adamtornhill/code-maat
- Developed as a material for Your Code as a Crime Scene and Software Design X-Rays.
- Evolved into CodeScene.

Tools: CodeScene

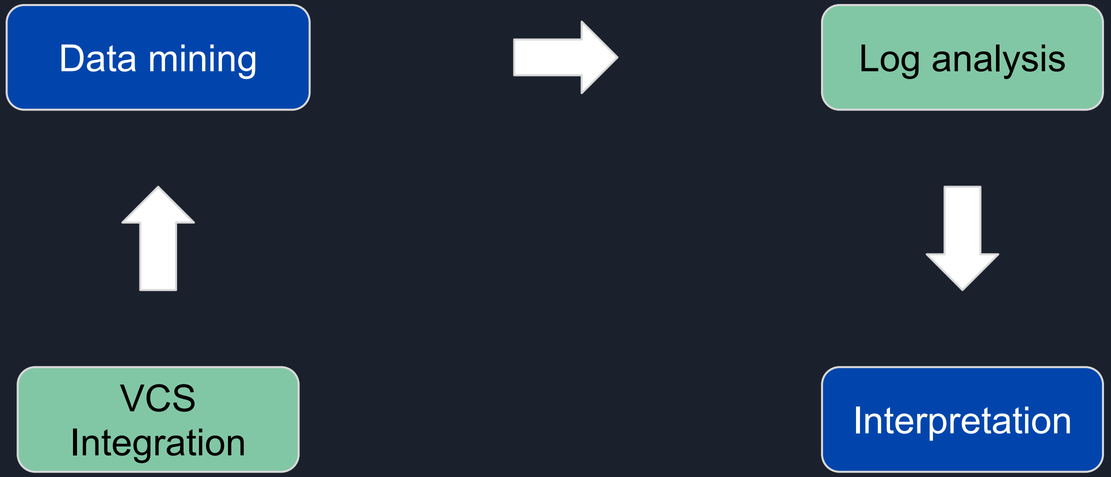
CodeScene claims to perform automatic behavioral code analysis, integrated on your source control platform and free of charge for open source projects.

This tool advertises as having the following capabilities:

- Improving code quality
- Paying off technical debt
- Automatic code reviewing
- Effective team building

Tools: Codescene

Usage



?