BEHAVIOURAL CODE ANALYSIS

Carlos García Pelazas – UO287891

Jaime Loredo Teijeiro-U0287741

Christian Fernández Noriega – UO282650

Ahmet Erdem Yabacı - UO303561

1. INTRODUCTION

Behavioural code analysis is an approach to code analysis from the people's perspective. Code itself is an important piece, but it's more important to understand how the organization and developers behind the code have worked on the codebase to create it. That is where the true information is

2. WHAT IS BEHAVIORAL CODE ANALYSIS?

Before going deeper into the concept of behavioural code analysis, it is necessary to explain the concept of technical debt.

Technical Debt: Refers to the implied cost of additional work caused by choosing an easy but suboptimal solution now instead of using a better approach that would take longer to implement. However, not all technical debt is equal. Some technical debt might exist in stable parts of the codebase where changes are infrequent, and the impact is minimal. On the other hand, technical debt in more volatile or frequently changing parts of the codebase can have a significant impact on the organization's ability to adapt and evolve.

Now that we have clear this concept, what behavioural code analysis does, is not only to identify, but also to prioritize technical debt. Most organizations cannot fix all technical debt at once. So, when you run this analysis, you identify the technical debt that represents the most risk, what parts of the code becomes bottleneck for the team.

Code quality matters in context.

If you pick an average enterprise codebase you could spend years refactoring technical debt. Which would make the company go out of business. Therefore, if you have low-quality code and it's in stable parts of the code, then that's technical debt that you need to be aware of, because it could be a long-term risk, but it's not urgent right now. It is important to find the parts of the code where developers of some organization/company/team spend the most time. And focus on refactoring those parts.

Behavioural code analysis solves these issues.

3. HOW DO WE FIX IT?

The goal here is to identify the behavioural patterns that the developers have interacting with the code, where they spend the most time, and how is the technical debt on those parts of the code.

To analyse behavioural code, we need behavioural data of how the developers interact with the code. We can get the data from version control systems like Git. Git, apart from being a collaboration tool, is also a data source of how the system has evolved, which developers have worked in which part, how often, and what happened to the system after a change has been made. This is the perfect data we are looking for.

4. WHY IS IT IMPORTANT?

Average organizations waste between 23 and 42% of developers time dealing with technical debt and bad code.

The code itself can be very abstract, you can waste hours of your time by extending some low-quality code and not even realizing. So, this can really help you understand the level of technical debt and the consequences of it.

Also, to have data, helps you make informed decisions on what to prioritize, or on what to spend your time. Refactoring code can be expensive, but with this data you can get an estimation of the return on investment of refactoring low-quality code over not doing it. This can be very helpful to justify actions with businesspeople that don't necessarily understand code itself but decide where to invest resources.

5. WHAT IS THE DIFFERENCE BETWEEN STATIC CODE ANALYSIS AND BEHAVIOURAL CODE ANALYSIS?

Static code analysis tools like Sonar cloud, is a good feedback tool when writing code. But it's not good at prioritization of issues. The main difference is that static code analysis can't measure the impact of the code on the development time.

Behavioural code analysis solves the issues of static code analysis, but it adds the necessary context needed to prioritize refactoring of technical debt.

6. WHAT INFORMATION CAN WE FIND?

Behavioural code analysis can answer the following questions:

- Does the current architecture support the way we work with the system?
- Does the current architecture support how the system evolves?

Very often you find that teams are fighting against their own architecture.

Apart from this, it is also very useful to identify knowledge gaps in the terms of code familiarity within the team.

Sometimes a hotspot/bottleneck doesn't happen because of a technical problem, but because a lack of familiarity with the code. The solution in this case might be a better onboarding process of developers to the company, than a code refactoring.

7. WHEN IS A GOOD TIME TO APPLY BEHAVIOURAL CODE ANALYSIS? IN WHICH PROJECTS?

Brown fields systems is the sweet spot for a behavioural code analysis, because there is enough data to perform a proper analysis and to have valid conclusions.

For green-field projects, you can also perform behavioural code analysis, but you need time to build up behavioural data. If you have a team of about 10 people, in three to four weeks, is likely that you have enough data to run this type of analysis, which can identify patterns and high-risk technical debt.

The earlier we can detect problems the better.

Also, is that greenfield projects tend to have strict deadlines, and with deadlines the code quality decreases, so it's good to have a security net with the behavioural code analysis.

8. HOW CAN WE INTEGRATE IT WITH SOFTWARE DEVELOPMENT CYCLE OR CONTINUOUS INTEGRATION?

During the development cycle of a system, you can apply behavioural code analysis in the following:

- Pull requests: To check that you are not degrading the code quality of any hotspot. The problem with this, is that it does not get the full picture of the code, and not enough behavioural data is gathered in a pull request. For this purpose, you can use static code analysis.
- Sprint retrospectives: A very good use case of behavioural code analysis in the development cycle is at the end of sprints. Use it as a communication tool in team retrospectives. This is very useful to make informed decisions on what to prioritize in the following sprints.

9. WHAT TOOLS CAN WE USE FOR BEHAVIOURAL CODE ANALYSIS

The guest of the podcast is Adam Tornhill, CTO of CodeScene, which is a startup on behavioural code analysis.

This tool does the following:

- Gathers data about code changes, commits and branches
- Analyses the history of the code to identify patterns of how the codebase evolves
- Calculates code health
- Assess the risk associated with different parts of the codebase
- It provides recommendations for improving code quality, reducing technical debt, and managing software processes more efficiently.

This tool costs 20$ or 30$ per month depending on the size of your organization.

10. BIBLIOGRAPHY

[SE Radio 554: Adam Tornhill on Behavioural code analysis]

[CodeScene]

Behavioural code analysis 101