Next-Level of Static Code Analysis Tool

Anti-Patterns and Design Issue, Is that All?
Firdaus Harun

Client Solution Lead | Software Quality Researcher at Embold Technologies, GmbH (embold.io)

Help Enterprise to Identify and Refactor Bad Smells and Anti-Patterns in Code

Enthusiast in Software Quality & Technical Debt
Agenda

Static Code Analysis:
- Background
- Its impact
- Current challenges

Generations of static code analysis

What’s next?
- WOW features
- Technical Debt*
I’m NOT Talking About

Static Analysis in-depth

Details on Anti-Patterns and Design Issues
Agenda

Static Code Analysis:
- Background
- Its impact
- Current challenges

Generations of Static Code Analysis

What’s Next?
- WOW features
What is a Static Code Analysis?
Code Quality in General

- Source code
- Build script

Developers

Static code analysis

Evaluators

Quality score and metrics

Structural representation

Issues reported
Why We Need the Static Code Analysis Tool?

To ensure the codebase conform with high quality software design such as:
- Open-Closed Principle
- Single Responsibilities
- Design Pattern
...

To produce high quality software that make customer happy 😊 (value)
- Less bugs and defects
- Easy to maintain and less cost to fix or refactor
What If We Ignore all These Issues and Warning? Code Smells and It’s Impact 1

God Class / Brain Method

- Flaw design in architectural design (J. Santos et al., 2018)
- Merge conflict that leads to more bugginess (Ahmed et al., 2017)
- Positive correlation with defects (Zazworka et al., 2014)

Data Class

- Make an editing effort takes more longer (Soh et al., 2016)
- Too many dependencies with this data class (Waog, 2014)

Feature Envy

- Make a searching effort is quite harder (Soh et al., 2016)
- Hard to implement unit test on functionality (bojanv91, 2017)
What If We Ignore all These Issues and Warning? Code Smells and It’s Impact 2

**Shotgun surgery**

- Lots of duplicate code ([Dzone, 2019](#))
- Takes more time to develop a small feature because the functionalities are splitted into several classes ([Ndepend, 2019](#))
- Hard to understand the EXACT feature responsibilities ([Ndepend, 2019](#))

**Message Chain**

1. Transitive dependencies between classes which hard to fix and maintain ([Embold, 2019](#))
2. They are easily fragile when the dependencies between classes or objects are break in the middle ([Embold, 2019](#))

many more ...
State-of-the-Art: Static Code Analysis Tool

- Code issues / smells / anti-patterns detection
- Structural metrics
- Code quality models
  Style guide enforcements
- Dependency view
  City view / hotspot area
- Clone management
- Dashboard
- Aggregated Metrics
Huge Challenge for Static Code Analysis Tools

<table>
<thead>
<tr>
<th>Large Scale Application and Bug Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Loss information while analyzing a large scale code base</td>
</tr>
<tr>
<td>• Hard to assess the impact of code smells or anti-patterns</td>
</tr>
<tr>
<td>• Difficult to predict a likelihood of bug</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>False-Positive Reduction and No Standard Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>• High rate of false positive code smells / anti-patterns class</td>
</tr>
<tr>
<td>• Even rules are defined but each tool detects differently the smelly classes</td>
</tr>
<tr>
<td>• Threshold settings for each metrics are quite challenging based on size of system</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lack of IDEs, DevOps / Infrastructure of Code Integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Not integrated with developer’s workflow</td>
</tr>
<tr>
<td>• Too expensive to run in a large-scale deployment</td>
</tr>
<tr>
<td>• Takes too long to run</td>
</tr>
</tbody>
</table>

including interpreting output from the tool and organizing/layouting information for large code base.
What are the Mistakes if We Only Rely On Static Code Analysis Tool?

- **Just rely on metrics, that’s it.**
  - Too focus on metrics and other software measurement
  - At the end, it does not reflect with architecture at all
  - Therefore, how can it evolves?
  - To evolve and mitigate a risk, we must understand thoroughly in architecture in general and require in-depth analysis

- **Target metric value as a main goal**
  - This will lead developers try to find ways to achieve
  - May lead an opposite steps to what was planned

Agenda

Static Code Analysis:
State-of-the-Art

Background
Its impact
Current challenges

Generations of Static Code Analysis

What’s Next?
WOW features
Generations of Static Code Analysis

1st Generation
- Lint tools
- Limited analysis capabilities
- Limited code metrics

2nd Generation
- A few code smells
- Bug catching
- Prioritization and rank issues
- Dependency view between files
- More structural metrics

3rd Generation
- Anti-patterns on code and architecture level
- Prioritization and rank with different criteria
- Dependency view on file/package level and cyclic
- Support a various type of languages
- Integrate with IDEs and CI/CD pipeline
- More aggregated metrics

4th Generation
- AI / ML?

What's next?
Agenda

Static Code Analysis:
State-of-the-Art
- Background
- Its impact
- Current challenges

Generations of Static Code Analysis

What’s Next?
- WOW features
What’s Next?
Automated Refactoring

- Code level
- Functional level
- Architecture level

- Search-based optimization techniques
- Refactoring steps on field, method and class level
- Metrics

Execute

Refactored code
Recommendation Engine on Finding Likely Buggy Code

- Embold RE learns from historical data and matches new code with patterns previously learned.
- In addition to Embold’s Design and Code Quality analysis, it builds a correlation between code changes and Bugs.
- Embold RE achieves this by using deep learning techniques.
- A neural network (LSTM) is trained using an encoder/decoder architecture to associate code change patterns with bug descriptions.
- This is used to predict if a new code change is likely to introduce a bug.
Embold score rates the quality of the application. It ranges from -5 to 5. -5 indicates bad and 5 indicates good quality software.

The overall rating is derived from design, metrics, duplication and code quality ratings.

These hotspots require attention, as they affect the overall quality the most.

Automatic prioritization of issues.
Strong Visualisation to Rapidly understand complex software
There are Few More that can be Considered ...

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic analysis</td>
<td>Examine in-depth functional behavior, memory usage etc.</td>
</tr>
<tr>
<td>Security flaws detection</td>
<td>Understand the context and can detect any loop holes or misused APIs</td>
</tr>
<tr>
<td>DevOps and IaC integration</td>
<td>Ensure consistency of deployment of DevOps and cloud-based technologies</td>
</tr>
<tr>
<td>Self-learning anti-patterns detection</td>
<td>Reduce false positive rates which based on code changes by developers</td>
</tr>
</tbody>
</table>
Concluding Remarks

• Code quality can be **ensured** with static code analysis tool

• Even though there are few limitation of current static code analysis tool, **expert review is still needed** to reduce false positive smelly classes

• There are **a lot of on-going research** on code smells / anti-patterns domain which **influences the development** of static code analysis tools over time

• **Integrating an AI** with static code analysis tool **can improve analysis and produce a better result**, thus **giving a confidence** to developers to use it

Thanks!

Connect me @mfirdausharun at LinkedIn and Twitter