Static testing

TVVS 2019/20

André Baptista - up201505375
Francisco Machado Santos - up201607928

Definition

Static testing are a set of techniques for **manually examining** software development **documents** and **code** without the need of executing it.

Generally, static testing can be divided into two main processes:

- **Reviews**
- **Static analysis using tools**

Reviews

The purpose of these meetings is to find **defects** before they turn into failures and to increase the overall **understanding** of the project.

Any document of a software project can be reviewed, including code. Examples:

- System requirements, architecture and design documents;
- Policy, strategy and business plans;
- Acceptance test plans;
- User manuals.

Reviews

There are different **types** of reviews with varying levels of formality:

Informal review
Walkthrough
Technical review
Inspection

less formal

more formal

Steps

Planning ➡ Kick-off ➡ Individual preparation ➡ Review meeting ➡ Rework ➡ Follow-up

Roles

Manager, Moderator, Author, Reviewers, Scribe

Reviews

Comparison to Dynamic testing

Reviews can take place much sooner than dynamic testing.

Defects found during reviews are simpler and less costly to remove.

Unlike dynamic testing, reviews are capable of:

finding **omissions** (for example missing requirements)

Identifying **deviations** from the project's norms

Reviews

Pros

Reduction in testing and maintenance costs
Decreased number of faults
Improved schedules

Cons

They require roughly 5-15% of development effort.

Possible failure reasons

Lack of training in formal techniques
Poor documentation quantity and quality
Inappropriate schedules that don’t accommodate time for reviews
Inability to improve processes based on meeting results

Static analysis using tools

The goal of static analysis tools is to **automate** the finding of **defects** in a software's **source code**.

These tools extend the knowledge gained from the compiler's **lexical**, **syntax** and **semantic** analysis. Common found defects:

Unreachable code
Unused variables
Uncalled functions
Array bound violations
Referencing undefined variables
Coding standards and guidelines violations

Static analysis using tools

The analysis performed by these kind of tools make use of many techniques, such as:

Data flow analysis

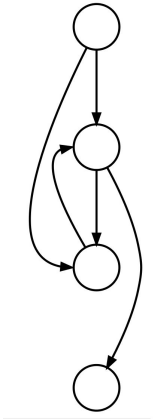
Control flow analysis

Static analysis using tools

Control flow analysis

Makes use of the Control Flow Graph to determine the **paths** and **order** of program statements. This analysis is capable of finding:

- Unaccessible nodes
- Infinite loops
- Infeasible paths
- Reducible graphs
- Information about loops:
  - Loop invariants
  - Nested loops
  - Multiple paths to entry



Static analysis using tools

Data flow analysis

Studies the **usage of variables** and their **values** throughout the flow of the program by analyzing the program's Control Flow Graph.

The status of variables can be categorized into:

(d) Defined, Created, Initialized
(k) Killed, Undefined, Released
(u) Used:
    (c) in a calculation
    (p) in a predicate

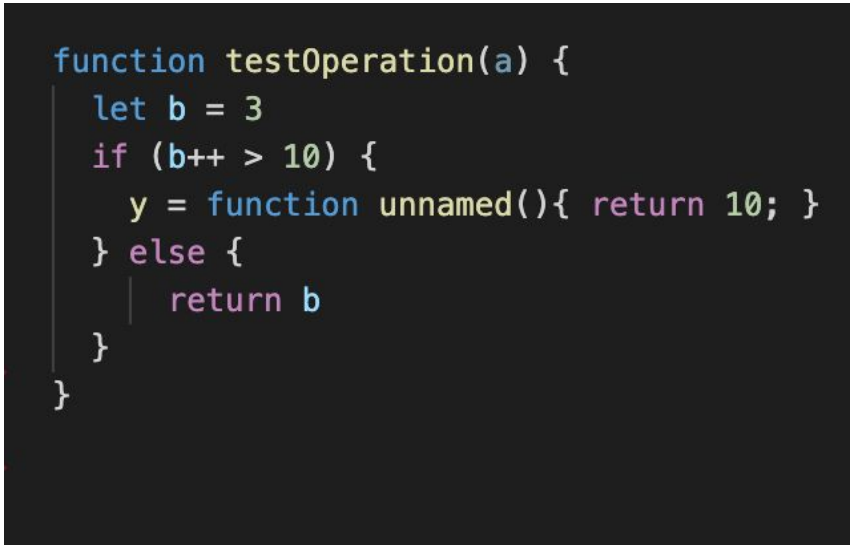
Static Analysis Tools

Static Analysis Tools

Why are these so important?

Challenge

Can you guess how many errors does this piece trigger?



```
function testOperation(a) {
  let b = 3
  if (b++ > 10) {
    y = function unnamed(){ return 10; }
  } else {
      return b
  }
}
```

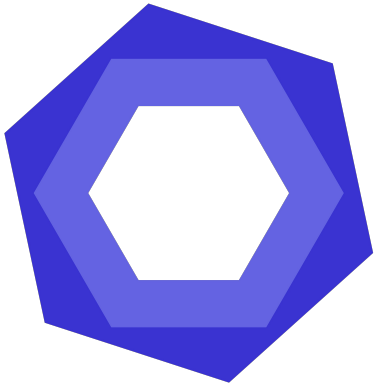
Challenge

```
function testOperation(a) {
  let b = 3
  if (b++ > 10) {
    y = function unnamed(){ return 10; }
  } else {
      return b
  }
}
```

JS index.js 23

- Expected to return a value at the end of function 'testOperation'. eslint(consistent-return) [1, 10]
- 'testOperation' is defined but never used. eslint(no-unused-vars) [1, 10]
- 'a' is defined but never used. eslint(no-unused-vars) [1, 24]
- Expected linebreaks to be 'CRLF' but found 'LF'. eslint(linebreak-style) [1, 28]
- Expected indentation of 1 tab but found 2 spaces. eslint(indent) [2, 1]
- Expected linebreaks to be 'CRLF' but found 'LF'. eslint(linebreak-style) [2, 12]
- Missing semicolon. eslint(semi) [2, 12]
- Expected indentation of 1 tab but found 2 spaces. eslint(indent) [3, 1]
- Unary operator '++' used. eslint(no-plusplus) [3, 7]
- Expected linebreaks to be 'CRLF' but found 'LF'. eslint(linebreak-style) [3, 18]
- Expected indentation of 2 tabs but found 1. eslint(indent) [4, 1]
- 'y' is not defined. eslint(no-undef) [4, 2]
- Missing space before opening brace. eslint(space-before-blocks) [4, 24]
- Expected linebreaks to be 'CRLF' but found 'LF'. eslint(linebreak-style) [4, 38]
- Missing semicolon. eslint(semi) [4, 38]
- Expected indentation of 1 tab but found 2 spaces. eslint(indent) [5, 1]
- Expected linebreaks to be 'CRLF' but found 'LF'. eslint(linebreak-style) [5, 11]
- Mixed spaces and tabs. eslint(no-mixed-spaces-and-tabs) [6, 2]
- Expected linebreaks to be 'CRLF' but found 'LF'. eslint(linebreak-style) [6, 12]
- Missing semicolon. eslint(semi) [6, 12]
- Expected indentation of 1 tab but found 2 spaces. eslint(indent) [7, 1]
- Expected linebreaks to be 'CRLF' but found 'LF'. eslint(linebreak-style) [7, 4]
- Expected linebreaks to be 'CRLF' but found 'LF'. eslint(linebreak-style) [8, 2]

Static Analysis Tools



ESLint (open-source)

Language: JavaScript

Pros:

- Rules are configurable and can be customized
- Covers both code quality and coding style issues
- Available as a plugin in numerous code editors/IDEs (Visual Studio Code, JetBrains' WebStorm, Eclipse, Atom...)

Cons:

- Initial configuration is required
- Not the fastest linter

Static Analysis Tools

Astrée

Astrée (created and developed by)

Real-time embedded software static analyzer

Language: C

Pros:

- Targets safety-critical embedded code
- Suitable for critical applications such as flight control software.

Cons:

- Lack of updates

Static Analysis Tools

CODACY

Codacy (created and developed by Codacy - open source)

Language: Java, JavaScript, PHP, JSP, C#

Pros:

- Github, Bitbucket and Gitlab integration
- Highly detailed and clear description of the issues
- Interface to manipulate and configure the rules used to analyse source code

Cons:

- Analysis tends to be complex in large projects
- Small community
- Limit the access to the source code in the UI

Static Analysis Tools

RIPS (created and developed by RIPS Technology)

Pros:

- Great product support (online and business hours)
- Easy integration in continuous integration and delivery tools
- One of the best products in static application security testing

Cons:

- Paid Product

Static Analysis Tools

Coverity (created and maintained by Synopsys)

Languages: C, C++, Java, Python, JavaScript, PHP, etc. (Multi-Language)

Pros:

- Great number of supported frameworks
- Integration with GitHub
- Fast. It quickly analyzes large projects in the order of hundreds of millions lines of code
- Highly security-centred (resource leaks, dereferences of NULL pointers, control flow issues, etc.)

Cons:

- “Heavy” interface

Static Analysis Tools

SonarLint

Languages: JavaScript, TypeScript, HTML, PHP, Python, Apex (Multi-Language)

Pros:

- Open Source
- Rich documentation and rule descriptions on the spot
- No configuration required, ready to use immediately
- Available as a plugin in numerous IDEs (Eclipse, IteliJ IDEA, Visual Studio and VSCode)

Cons:

- Difficult to manually configure the rules we want to use

Static Analysis Tools

Cppcheck (open-source)

Language: C++

Pros:

- Fast tool
- Regularly updated
- Existence of plug-ins in the most used IDEs

Cons:

- Solely focuses on bugs and not styling issues
- Limited bugs discovery

Static Analysis Tools

Pylint (open-source)

Language: Python

Pros:

- Very-descriptive errors
- Creates statistics about the number of warnings and errors found in different files
- Provides a code quality metric (out of 10) based on the linting errors

Cons:

- Does not highlight the code where errors are located

Static Analysis Tools Hyperlinks

ESLint : https://eslint.org

Astrée: http://www.astree.ens.fr

Codacy : https://www.codacy.com

RIPStech : https://www.ripstech.com

Coverity : https://scan.coverity.com

SonarLint : https://www.sonarlint.org/vscode/

Cppcheck : http://cppcheck.sourceforge.net

PyLint : https://www.pylint.org

References

https://www.cs.drexel.edu/~spiros/teaching/SE320/slides/dataflow-testing.pdf

https://www.cs.cmu.edu/~aldrich/courses/654-sp08/slides/11-dataflow.pdf

http://web.cs.iastate.edu/~weile/cs513x/2018spring/lecture2-controlflow.pdf

https://www.template.net/business/tools/static-analysis-tool/ (mostly paid tools)