

Seminar Software Quality

Preliminary meeting

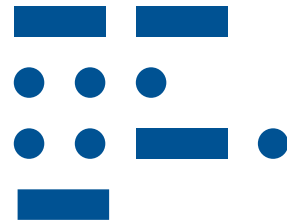
We will start at 11:02

Fabian Leinen (Orga)

Jakob Rott
Roland Würsching
Dr. Markus Schnappinger
Maximilian Jungwirth
Dr. Andreas Stahlbauer
Martin Gruber



Software Quality



Code



Tests

Participating

1

Apply via matching tool

2

Application with us: Online form

- Letter of motivation
- Optional: CV + grade report
- Your 3+ favorite topics

<http://go.tum.de/070420>



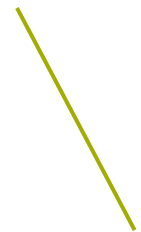
February 19th, 23:59

Schedule

Application



Matching



Matching results and
topic assignment



Kickoff



Individual working
phase



Block seminar



Until 19th of February

April

July

- Literature research
- How to thesis?
- Effective presentations



Grading

Thesis

- Seminar paper: max. 15 pages
- Content: Theory + **application** of the topic
(results, experiences, problems and limitations)
- Initial submission
- Final submission: 1 week after presentation

Presentation

- 20 min + 10 min discussion
- Mandatory dry run (1 week before seminar)

50/50



Questions about the organization?

Clone Detection:

"Where can identical (copied) parts be found in source code?"

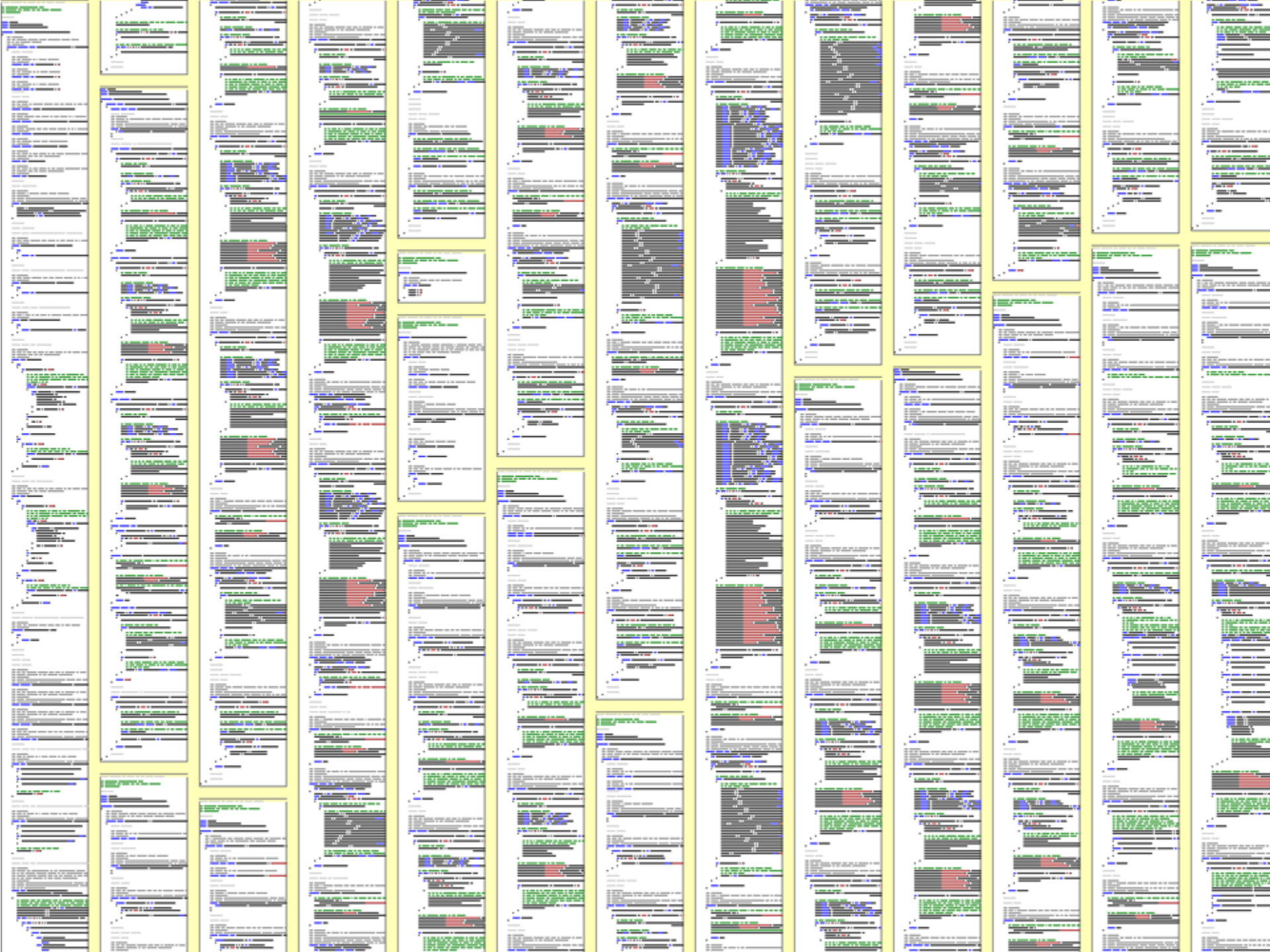
```
// Utilities for arrays of elements
public String showElements(ModelElement[] elements, String nomsg) {
    boolean found = false;
    StringBuffer res = new StringBuffer();
    if (elements != null) {
        Index.getInstance().setCurrentRenderer(
            FlatReferenceRenderer.getInstance());
        for (int i = 0; i < elements.length; i++) {
            ModelElement el = elements[i];
            res.append(showElementLink(el)).append(HTML.LINE_BREAK);
            found = true;
        }
        Index.getInstance().resetCurrentRenderer();
    }
    if (!found && nomsg != null && nomsg.length() > 0) {
        res.append(HTML.italics(nomsg));
    }
    return res.toString();
}
```

```
// Utilities for arrays of elements
public String showElements(ModelElement[] elements, String nomsg) {
    boolean found = false;
    StringBuffer res = new StringBuffer();
    if (elements != null) {
        Index.getInstance().setCurrentRenderer(
            FlatReferenceRenderer.getInstance());
        for (int i = 0; i < elements.length; i++) {
            ModelElement el = elements[i];
            res.append(showElementLink(el)).append(HTML.LINE_BREAK);
            found = true;
        }
        Index.getInstance().resetCurrentRenderer();
    }
    if (!found && nomsg.length() > 0) {
        res.append(HTML.italics(nomsg));
    }
    return res.toString();
}
```



```
// Utilities for arrays of elements
public String showElements(ModelElement[] elements, String nomsg) {
    boolean found = false;
    StringBuffer res = new StringBuffer();
    if (elements != null) {
        Index.getInstance().setCurrentRenderer(
            FlatReferenceRenderer.getInstance());
        for (int i = 0; i < elements.length; i++) {
            ModelElement el = elements[i];
            res.append(showElementLink(el)).append(HTML.LINE_BREAK);
            found = true;
        }
        Index.getInstance().resetCurrentRenderer();
    }
    if (!found && nomsg != null && nomsg.length() > 0) {
        res.append(HTML.italics(nomsg));
    }
    return res.toString();
}
```





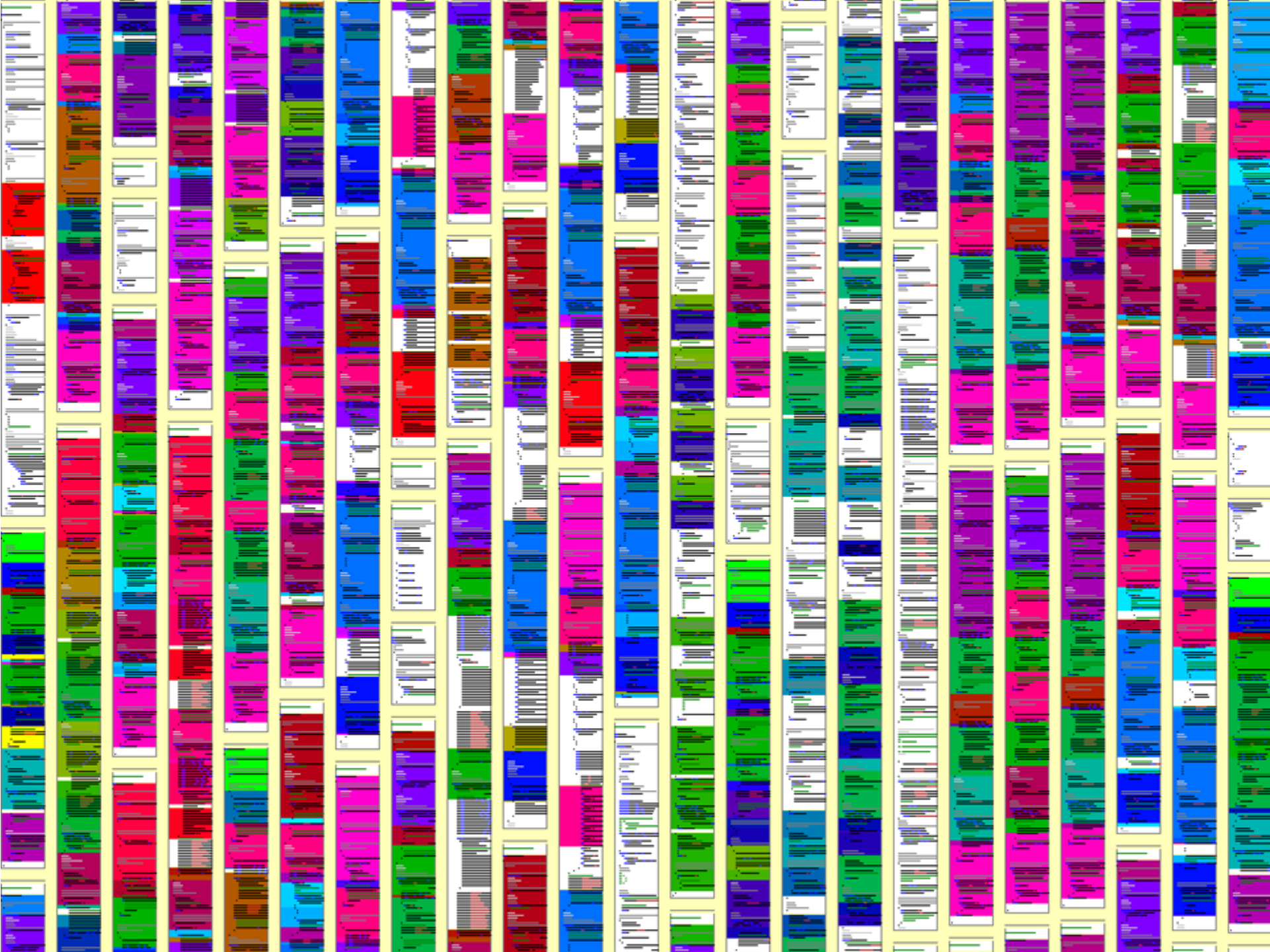


TUM

CQSE

msg
RESEARCH



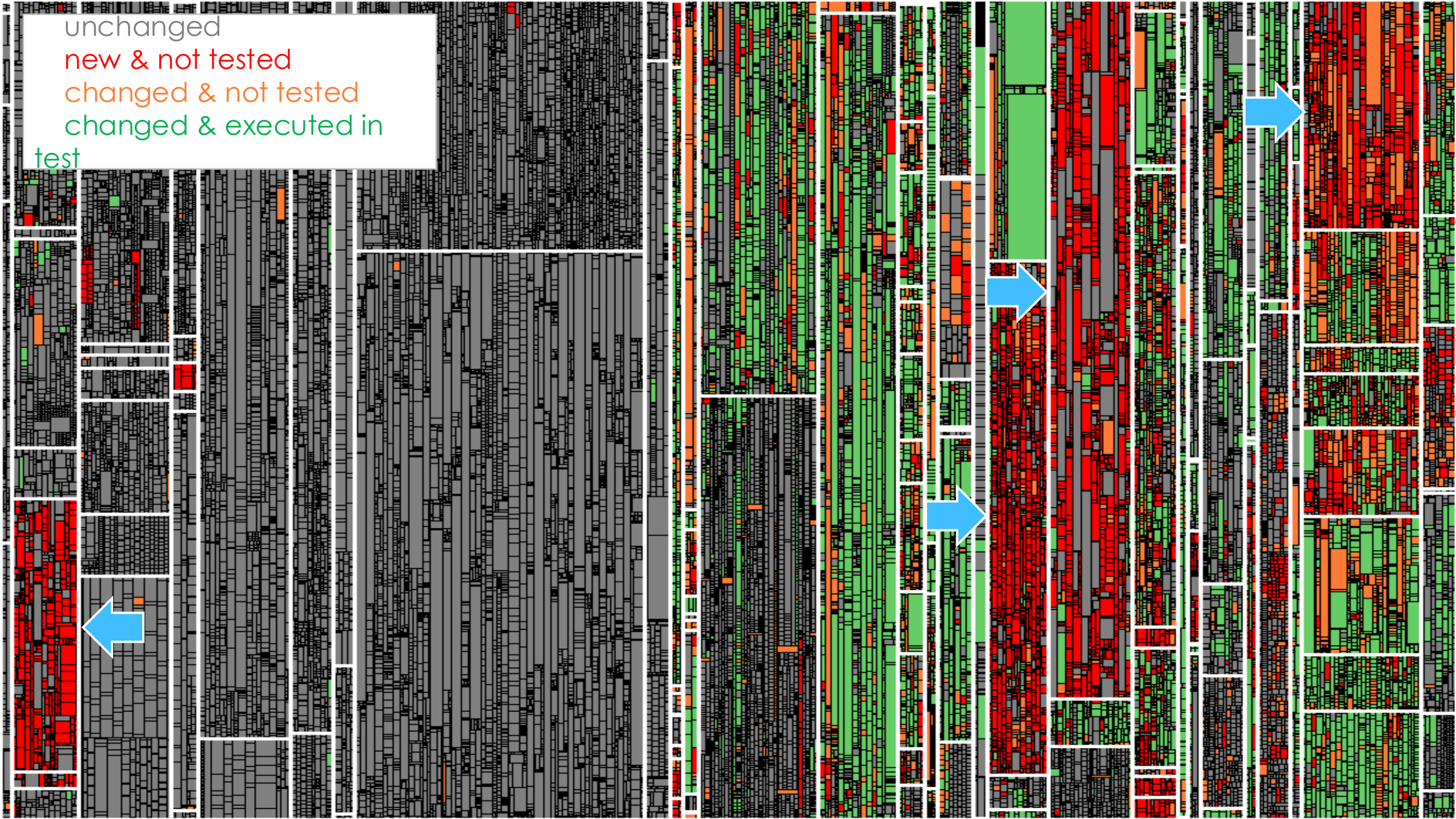




Test Gap Analysis

"Have all changes since the last release been tested?"

unchanged
new & not tested
changed & not tested
changed & executed in test



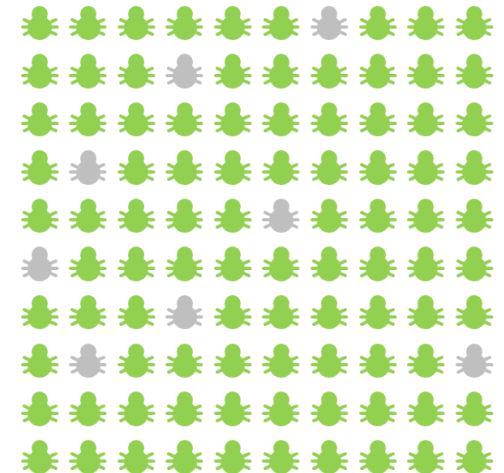
Test Selection And Prioritization Without Coverage, With Information Retrieval Techniques

Problem. Selecting a **subset of tests** written to cover only changes in a merge request can **save computational resources, time**, and speed-up the feedback cycles from the CI/CD pipelines. Many approaches use **per-test code coverage** to identify relevant tests. This coverage information is rather hard to get in industry contexts.

Approach. Use approaches based on **IR techniques** for selecting and prioritizing related tests.

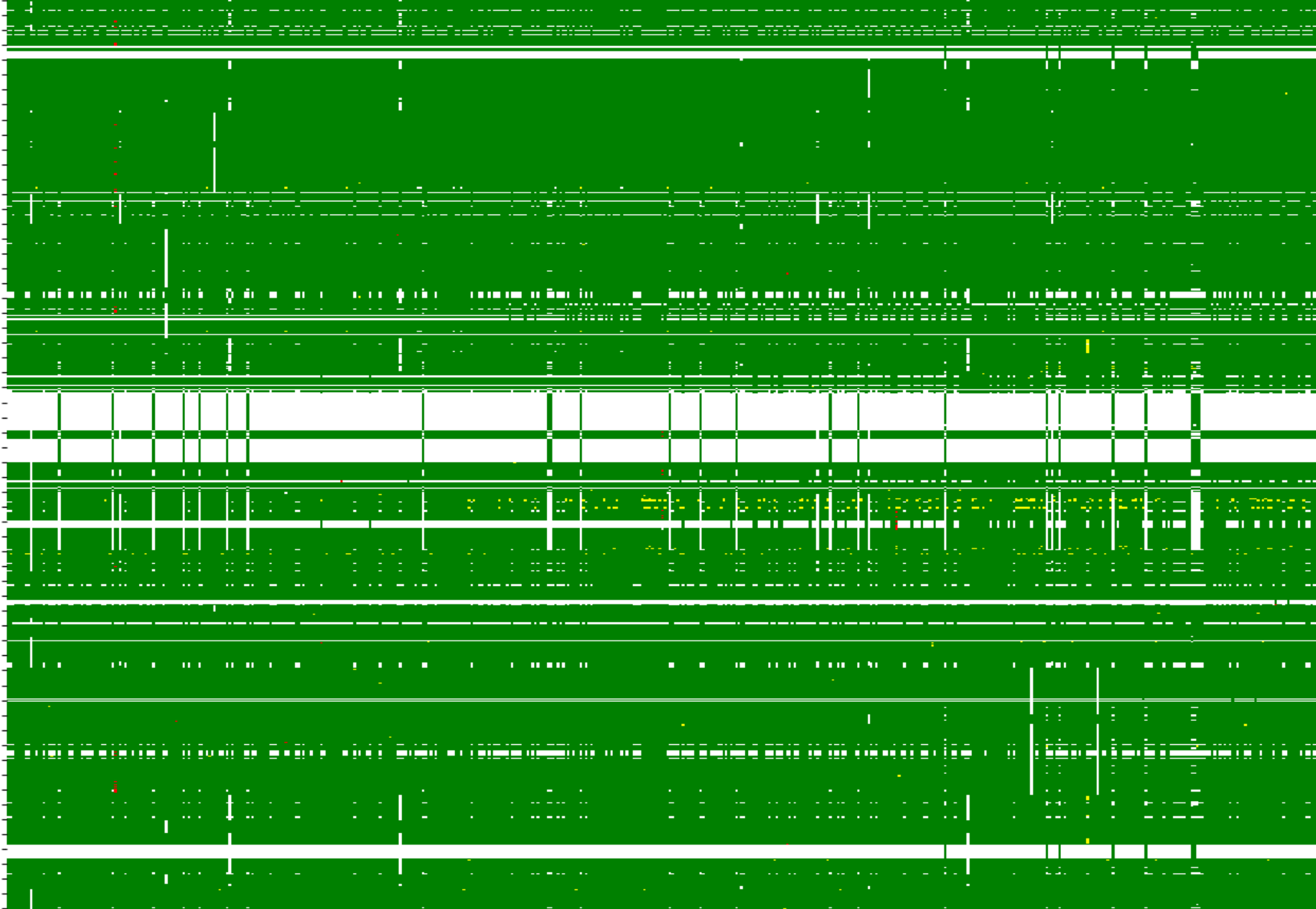
Task

1. **Evaluate** the IR-based test selection approach implemented in *Teamscale*, compare different scoring strategies.
2. Do a brief **literature review** on techniques for test-selection and prioritization without test coverage.
3. Outline **potential enhancements** of the existing IR-approach to enhance the retrieval performance.

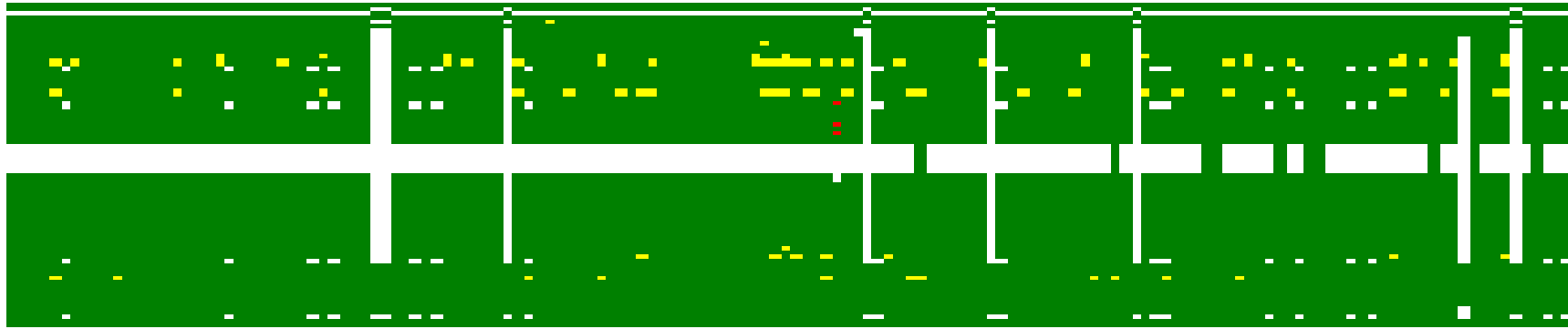


Flaky Tests, Solid Solutions: Finding Common Root Causes of Flaky Tests

	44a5562	c2b92d1	ce8bf3c	
testLogin()				
testRegister()				Retry



Flaky Tests, Solid Solutions: Finding Common Root Causes of Flaky Tests



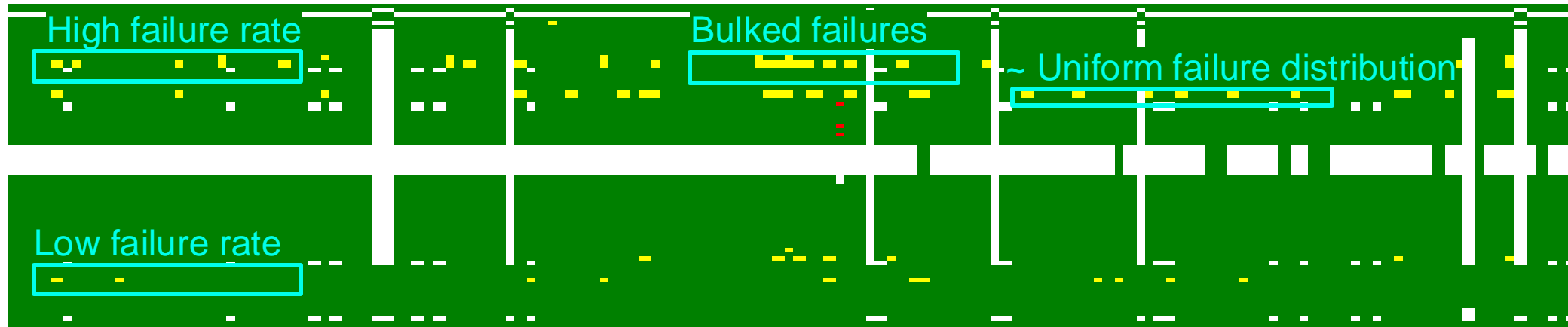
There are test cases failing flakiness together

→ Finding and understanding these clusters helps repairing flaky tests

Skills:

- Python
- Data science
- Efficient algorithms

Debugging Flaky Tests: Horizontal Test Result History Analysis



Root Causing

- Concurrency
- Randomness
- Networking


Event Detection

- Tests becoming flaky
- Changes in failure rate
- Tests becoming stable

Skills:

- Python
- Data science
- Efficient algorithms

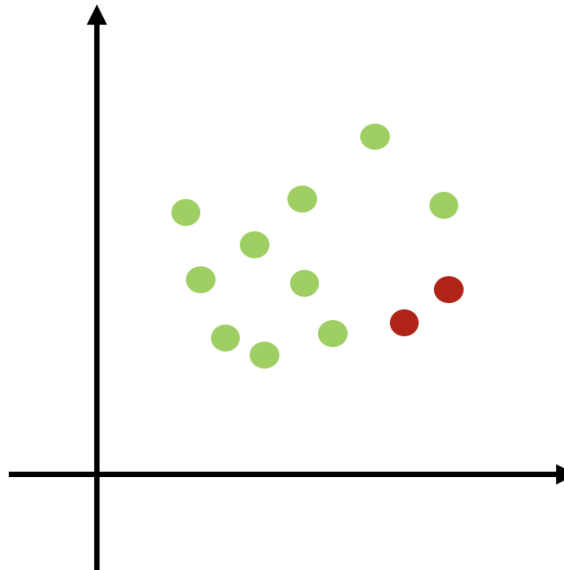
Using Pre-Trained Embedding Models for Diversity-Based Test Prioritization



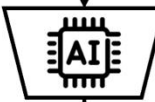
```
def test_90_kmh_to_ms() -> None:
    assert kmh_to_ms(90) == 25
```

JaCoCo

Element	Missed Instructions	Cov.	Missed Branches	Cov.
org.jacoco.core		97%		90%
org.jacoco.examples		58%		64%
org.jacoco.agent.rt		75%		83%
jacoco-maven-plugin		90%		82%
org.jacoco.cli		97%		100%
org.jacoco.report		99%		99%
org.jacoco.ant		98%		99%
org.jacoco.agent		86%		75%
Total	1,454 of 29,386	95%	206 of 2,442	91%



```
def test_90_kmh_to_ms() -> None:
    assert kmh_to_ms(90) == 25
```



-3,14 0,42 ... 1,23

Using AI to generate Test Cases: **Reality** or **Illusion**

GitHub Copilot and other Coding Assistants claim to generate useful unit tests, thus relieving developers.



The quality of the generated tests is actually **pretty bad**.

Anonymous

SWQ Student 2024

I will **improve** the quality of the generated tests!



You

SWQ Student 2025

Nice!



Me

Seminar Advisor

Quality Issues in Natural Language Tests

Step	Description	Expected Result
1	Launch the browser.	Browser is started.
2	Click menu → select 'Customize'.	The 'Customize' window is opened.
3	Drag 3 new items from the palette or menu panel and drop them onto the Navigation toolbar.	All items are added onto the Navigation toolbar.
4	Exit 'Customize'.	The changes are applied.
5	Wait at least 15 seconds, after exiting 'Customize', then restart the browser.	Browser is restarted and the previously made customizations are in place.

(1) **Identification** of quality issues:

- Ambiguous descriptions
- Long test steps
- Misplaced actions
- Inconsistent wording
- etc.

(2) Automated **improvement**

Participating

1

Apply via matching tool

2

Application with us: Online form

- Letter of motivation
- Optional: CV + grade report
- Your 3+ favorite topics

<http://go.tum.de/070420>



February 19th, 23:59