Examination of Performance Changes (at Code Level)

David Georg Reichelt$^{1,2}$

$^1$School of Computing and Communications, Lancaster University Leipzig,
$^2$Universitätsrechenzentrum, Universität Leipzig

15.4.2023
Why measure performance? [SSP19]

Which is faster?

```java
1  StringBuilder buf
2    = new StringBuilder(16);
3  buf.append("Hello\nWorld");
4
5  return buf.toString();
```

```java
1  StringBuilder buf
2    = new StringBuilder(16);
3  buf.append("Hello")
4     .append("\n")
5     .append("World");
6  return buf.toString();
```
Why measure performance? \[\text{[SSP19]}\]

Which is faster?

```java
1  StringBuilder buf
2       = new StringBuilder(16);
3  buf.append("Hello\World");
4
5  return buf.toString();
```

```java
1  StringBuilder buf
2       = new StringBuilder(16);
3  buf.append("Hello")
4       .append("\World")
5  .append("World");
6  return buf.toString();
```
Why measure performance? [SSP19]

Which is faster?

```java
StringBuilder buf = new StringBuilder(16);
buf.append("Hello/uni2423World");
return buf.toString();
```

Complex behaviour

⇒ impact of changes needs to be measured
Why Unit Test Measurement?

Software

- Component 1
  - Unit Test 1
- Component 2
  - Unit Test 2
- Component 3
  - Unit Test 3
Why Unit Test Measurement?

Software
- Component 1
- Component 2
- Component 3

Unit Test
- Component 1 → Unit Test 1
- Component 2 → Unit Test 2
- Component 3 → Unit Test 3

Load Test

Benchmark

D.G. Reichelt  Examination of Performance Changes (at Code Level)  15.4.2023
Why Unit Test Measurement?

~4% of all projects

~33% of all projects
Why Unit Test Measurement?

Software

Component 1

Component 2

Component 3

Unit Test 1

Unit Test 2

Unit Test 3

Load Test

Benchmark

~4% of all projects

⇒ unit tests are transformed and used for measurement

~33% of all projects
Performance analysis of software systems

CLI-Tool: https://github.com/DaGeRe/peass

Jenkins-Plugin: https://github.com/jenkinsci/peass-ci-plugin
Process of Peass

Repository

<table>
<thead>
<tr>
<th>Commit 1</th>
<th>Commit 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test 1</td>
<td>Test 1</td>
</tr>
<tr>
<td>Test 2</td>
<td>Test 2</td>
</tr>
</tbody>
</table>

Examination of Performance Changes (at Code Level) 15.4.2023
Process of Peass

Repository

<table>
<thead>
<tr>
<th>Commit 1</th>
<th>Commit 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test 1</td>
<td>Test 1</td>
</tr>
<tr>
<td>Test 2</td>
<td>Test 2</td>
</tr>
</tbody>
</table>

Measurement

<table>
<thead>
<tr>
<th>Commit 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test 1</td>
</tr>
<tr>
<td>Test 2</td>
</tr>
</tbody>
</table>

Examination of Performance Changes (at Code Level) 15.4.2023
Process of Peass

Regression Test Selection

- Test 1
- Test 2
- Test 3
- Test 4
- Test 5
Process of Peass

Regression Test Selection

Repository

Test 1 → Class 1 → Change
Test 2 → Class 2
Test 3 → Class 3
Test 4 → Class 4
Test 5 → Class 5
Process of Peass

**Repository**

- **Commit 1**
  - Test 1
  - Test 2
- **Commit 2**
  - Test 1
  - Test 2

**Regression Test Selection**

- **Commit 1**
  - Test 1
  - Test 2
- **Commit 2**
  - Test 1
  - **Test 2** crossed out

**Measurement**

- **Commit 2**
  - Test 1
  - Test 2

Examination of Performance Changes (at Code Level) 15.4.2023
Peass Method Performance Change Detection Application to GraalVM Data

**Process of Peass**

**Repository**
- Commit 1
  - Test 1
  - Test 2
- Commit 2
  - Test 1
  - Test 2

**Regression Test Selection**
- Commit 1
  - Test 1
  - Test 2
- Commit 2
  - Test 1
  - Test 2

**Root Cause Analysis**

**Measurement**
- Commit 2
  - Test 1
  - Test 2

Examination of Performance Changes (at Code Level) 15.4.2023
Process of Peass

Repository
- Commit 1
  - Test 1
  - Test 2
- Commit 2
  - Test 1
  - Test 2

Regression Test Selection
- Commit 1
  - Test 1
  - Test 2
- Commit 2
  - Test 1
  - Test 2

Root Cause Analysis

Measurement
- Commit 2
  - Test 1
  - Test 2
Regression Test Selection - Approach [LTB18]

<table>
<thead>
<tr>
<th>TestA</th>
<th>TestB</th>
</tr>
</thead>
<tbody>
<tr>
<td>void testA1()</td>
<td>void testB1()</td>
</tr>
<tr>
<td>void testA2()</td>
<td>void testB2()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class α</th>
<th>Class β</th>
</tr>
</thead>
<tbody>
<tr>
<td>void methodα1()</td>
<td>void methodβ1()</td>
</tr>
<tr>
<td>void methodα2()</td>
<td>void methodβ2()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class γ</th>
</tr>
</thead>
<tbody>
<tr>
<td>void methodγ1()</td>
</tr>
<tr>
<td>void methodγ2()</td>
</tr>
</tbody>
</table>

**Trace:**
TestA.testA1
Classα.methodα1
Classγ.methodγ1
Classα.methodα1
Classγ.methodγ1

**Change:**
...

Examination of Performance Changes (at Code Level) 15.4.2023
Regression Test Selection - Approach [LTB18]

**Trace:**
TestA.testA1
Classα.methodα1
Classγ.methodγ1
Classα.methodα1
Classγ.methodγ1

**Change:**
Classα.methodα1

---

TestA

```java
void testA1()
void testA2()
```

Class α

```java
void methodα1()
void methodα2()
```

Class γ

```java
void methodγ1()
void methodγ2()
```

TestB

```java
void testB1()
void testB2()
```

Class β

```java
void methodβ1()
void methodβ2()
```
Regression Test Selection - Approach [LTB18]

**Trace:**
- TestA.testA1
- Class α.method α1
- Class γ.method γ1
- Class α.method α1
- Class γ.method γ1

**Change:**
- Class α.method α1
Regression Test Selection - Approach [LTB18]

**Trace:**
- TestA.testA1
- Class\(\alpha\).method\(\alpha\)1
- Class\(\gamma\).method\(\gamma\)1
- Change:
  - Class\(\alpha\).method\(\alpha\)2
Peass Method Performance Change Detection Application to GraalVM Data

Repository

Commit 1

Test 1  Test 2

Commit 2

Test 1  Test 2

Regression Test Selection

Commit 1

Test 1  Test 2

Commit 2

Test 1  Test 2

Root Cause Analysis

Measurement

Commit 2

Test 1  Test 2

9/29 D.G. Reichelt Examination of Performance Changes (at Code Level) 15.4.2023
Root Cause Analysis

- Root cause analysis for performance changes (Heger et al., 2013)
  - Getting the call tree

```
testA1()
methodα1()  methodγ1()
```
Root Cause Analysis

- Root cause analysis for performance changes (Heger et al., 2013)
  - Getting the call tree
  - Levelwise measurement of call tree nodes

\[
\begin{align*}
time() & \quad testA1() \\
time() & \quad time() \\
method_\alpha 1() & \quad time() \\
time() & \quad method_\gamma 1() \\
time() & \quad time()
\end{align*}
\]
Root Cause Analysis

- Root cause analysis for performance changes (Heger et al., 2013)
  - Getting the call tree
  - Levelwise measurement of call tree nodes
  - \texttt{time()-calls} if method$\alpha_1()$ and method$\gamma_1()$ influence measurement of testA1()
Root Cause Analysis

- Root cause analysis for performance changes (Heger et al., 2013)
  - Getting the call tree
  - Levelwise measurement of call tree nodes
  - $\text{time()}$-calls if method\(\alpha_1()\) and method\(\gamma_1()\) influence measurement of test\(A_1()\)
Root Cause Analysis

- Root cause analysis for performance changes (Heger et al., 2013)
  - Getting the call tree
  - Levelwise measurement of call tree nodes
  - time()-calls if method$_{\alpha1}()$ and method$_{\gamma1}()$ influence measurement of testA1()
Root Cause Analysis

- Root cause analysis for performance changes (Heger et al., 2013)
  - Getting the call tree
  - Levelwise measurement of call tree nodes
  - \texttt{time()-calls if method}^{\alpha} \texttt{1()} and \texttt{method}^{\gamma} \texttt{1()} influence measurement of \texttt{testA1()}

![Diagram of call tree showing time() and method calls]

- Challenges for unit test measurement
  - Overhead influences measurement quality and duration
  - Low overhead necessary
  - Configuration and node selection strategy a priori unknown
Root Cause Analysis

- Root cause analysis for performance changes (Heger et al., 2013)
  - Getting the call tree
  - Levelwise measurement of call tree nodes
  - \texttt{time()-calls if method}_α\texttt{1()} and \texttt{method}_γ\texttt{1()} influence measurement of \texttt{testA1()}

\begin{center}
\begin{tikzpicture}
  \node (root) {\texttt{time()}};
  \node (testA1) [below left of=root] {\texttt{testA1()}};
  \node (time) [below of=root] {\texttt{time()}};
  \node (methodA1) [below of=testA1] {\texttt{method}_α\texttt{1()}};
  \node (methodGamma1) [below of=time] {\texttt{method}_γ\texttt{1()}};
  \node (time2) [below of=methodGamma1] {\texttt{time()}};

  \draw[->] (root) -- (testA1);
  \draw[->] (root) -- (time);
  \draw[->] (testA1) -- (methodA1);
  \draw[->] (time) -- (methodA1);
  \draw[->] (testA1) -- (methodGamma1);
  \draw[->] (time) -- (methodGamma1);
  \draw[->] (methodGamma1) -- (time2);
\end{tikzpicture}
\end{center}
Root Cause Analysis

- Root cause analysis for performance changes (Heger et al., 2013)
  - Getting the call tree
  - Levelwise measurement of call tree nodes
  - `time()`-calls if method $\alpha_1()$ and method $\gamma_1()$ influence measurement of testA1()

```
1
  2
 / \  /
3   3
```

- Challenges for unit test measurement
  - Overhead influences measurement quality and duration
    ⇒ Low overhead necessary
  - Configuration and node selection strategy a priori unknown
Root Cause Analysis

- Root cause analysis for performance changes (Heger et al., 2013)
  - Getting the call tree
  - Levelwise measurement of call tree nodes
  - `time()`-calls if `method_α1()` and `method_γ1()` influence measurement of `testA1()`

- Challenges for unit test measurement
  - Overhead influences measurement quality and duration
    ⇒ Low overhead necessary
  - Configuration and node selection strategy a priory unknown
Kieker Monitoring Process [WOSPC23]

**Figure:** Monitoring Process in Kieker

- **AspectJ** → **OperationExecutionRecord**
- **Instrumentation** → **Original source** → **Probe**
- **LinkedBlockingQueue** → **Queue**
- **FileWriter** → **Writer** → **Hard Disk**
Idea 1: Optimize each step

Idea 2: Only process necessary data

Figure: Monitoring Process in Kieker
Optimization Options [WOSPC23]

**Figure:** Possible Monitoring Optimizations
Optimization Options [WOSPC23]

Figure: Possible Monitoring Optimizations

- Source Code Instrumentation
- DurationRecord
- Instrumentation
- Original source
- Probe
- CircularFifoQueue
- AggregatedWriter
- Hard Disk

Optimizations all reduced overhead
Details in Paper
Performance Change Detection
Process of Peass

Repository

Commit 1
Test 1  Test 2

Commit 2
Test 1  Test 2

Regression Test Selection

Commit 1
Test 1  Test 2

Commit 2
Test 1  Test 2

Root Cause Analysis

Measurement

Commit 2
Test 1  Test 2
General Measurement Process (Adaption of Georges et al., 2007)

Measurement Start → Execution Workload → Measurement End
General Measurement Process (Adaption of Georges et al., 2007)

- Measurement Start
- Execution Workload
- Measurement End

Repetition
General Measurement Process (Adaption of Georges et al., 2007)
General Measurement Process (Adaption of Georges et al., 2007)
Example Workloads [QRS22]

- **Goal:** Determine statistical boundaries of performance change detection
- **Statistical boundaries depend on**
  - Type I error
  - Type II error
  - Standard deviation
  - Effect size (change size in relation to standard deviation)
  - VM count
Example Workloads [QRS22]

- **Goal:** Determine statistical boundaries of performance change detection
- **Statistical boundaries depend on**
  - Type I error
  - Type II error
  - Standard deviation
  - Effect size (change size in relation to standard deviation)
  - VM count
- **For examination of performance measurements:** Usage of example workloads with workload size $s$

- **Used workloads**
  - Add-workload: Adding $s$ random numbers
  - RAM-workload: Reservation of $s$ arrays consisting of 3 ints
  - Sysout-workload: Generating $s$ random numbers and printing them to System.out
Example Workloads [QRS22]

- **Goal:** Determine statistical boundaries of performance change detection
- **Statistical boundaries depend on**
  - Type I error Decided: 1%
  - Type II error
  - Standard deviation Empirical: 1%
  - Effect size (change size in relation to standard deviation)
  - VM count
- **For examination of performance measurements:** Usage of example workloads with workload size $s$
- **Used workloads**
  - Add-workload: Adding $s$ random numbers
  - RAM-workload: Reservation of $s$ arrays consisting of 3 ints
  - Sysout-workload: Generating $s$ random numbers and printing them to System.out
**Statistical Boundaries of Detecting Changes** [QRS22]

*Figure:* Type II error in relation to VM count and effect size \( \gamma \)
Example: Workload size 1,000 will take 97 seconds for 10,000,000 repetitions ⟹ at maximum 891 VMs are possible when using 12 hours measurement time

Boundaries depend on measurement duration; \( \gamma = 0.1 \) would require 4.808 VMs and will hardly be measurable
Example: Workload size 1,000 will take 97 seconds for 10,000,000 repetitions
⇒ at maximum 891 VMs are possible when using 12 hours measurement time

Boundaries depend on measurement duration; $\gamma = 0.1$ would require 4.808 VMs and will hardly be measurable

Measurements via monitoring (e.g. with OpenTelemetry, Kieker) or load tests have higher deviation and will only detect more coarse-grained performance changes
Approach [QRS22]

Unit Test Pair
(Workload Size $n, n \cdot (1 + d)$)

- Configuration 1
  - Sampling $n - n$
    - True Negatives
    - False Positives
  - Sampling $n - n \cdot (1 + d)$
    - True Positives
    - False Negatives

- Configuration ...

- Configuration $n$
  - Sampling $n - n$
    - True Negatives
    - False Positives
  - Sampling $n - n \cdot (1 + d)$
    - True Positives
    - False Negatives

Sampling

True Negatives

False Positives
Workload Size [QRS22]

- Examination of unit test sized workloads

$\Rightarrow$ What is a unit test size?

(What is $s$?)
Examination of unit test sized workloads
⇒ What is a unit test size?
(What is $s$?)

Approach: Analysis of call counts of open source projects
Workload Size [QRS22]

- Examination of unit test sized workloads
  ⇒ What is a unit test size?
  (What is $s$?)

- Approach: Analysis of call counts of open source projects
Example: Comparison of Statistical Tests [QRS22]

Figure: Average $F_1$-Score With T-Test
Example: Comparison of Statistical Tests [QRS22]

Figure: Average $F_1$-Score With T-Test

Figure: Average $F_1$-Score With Mann-Whitney-Test
Example: Comparison of Statistical Tests [QRS22]

30 VMs, 49 iterations, 100,000 repetitions and Mann-Whitney-Test are an efficient configuration for typical unit tests

Figure: Average $F_1$-Score With Mann-Whitney-Test
Example Usage: Peass-CI
Example Usage: Peass-CI

Detection of performance changes in CI: https://plugins.jenkins.io/peass-ci/

Steps of the approach are all implemented
<table>
<thead>
<tr>
<th>Peass Method</th>
<th>Performance Change Detection</th>
<th>Application to GraalVM Data</th>
<th>Summary and Outlook</th>
</tr>
</thead>
</table>

Application to GraalVM Data
Problem: GraalVM benchmarks might be executed to often

Goal: Determination of suitable VM count and iteration / run count

Problem: Artificial benchmark pairs and option to repeat them more often than necessary not available → Approach: Sample from existing data
Application to GraalVM Data – Approach

Measurement Data Pair

Changed (T-Test)

Configuration 1
- Sampling
  - True Positives
  - False Negatives

Configuration n
- Sampling
  - True Positives
  - False Negatives

Unchanged (T-Test)
Application to GraalVM Data – Approach

Measurement Data Pair

Changed (T-Test)

Configuration 1
- Sampling
  - True Positives
  - False Negatives

Configuration n
- Sampling
  - True Positives
  - False Negatives

Unchanged (T-Test)

Configuration 1
- Sampling
  - True Negatives
  - False Positives

Configuration n
- Sampling
  - True Negatives
  - False Positives
Figure: Difference 48994
Figure: Difference 49001
Figure: Difference 50180
Figure: Difference 50405
Summary and Outlook
Summary and Outlook

- Previous research
  - Definition of a method for performance change detection using unit tests:
    https://github.com/DaGeRe/peass
  - Regression test selection by trace analysis
  - Root cause analysis implementation using Kieker

- Performance change detection
  - Basic idea: Sample from artificial / known performance changes and derive measurement configuration
  - Current goal: Check configuration generation on GraalVM data
Thanks for your attention!

- David Georg Reichelt
- Assistant Professor
- Lancaster University Leipzig
- Contact: d.greichelt[a t]lancaster.ac.uk