Lancaster University Leipzig

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

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung

15.4.2023

Why measure performance? [SSP19]

Which is faster?

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

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

Why measure performance? [SSP19]

Which is faster?

```java
StringBuilder buf
  = new StringBuilder(16);
buf.append("Hello World");


return buf.toString();
```

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

Why measure performance? [SSP19]

Which is faster?

```
StringBuilder bu
  = new StringBu
buf.append("Hell


return buf.toStr
```

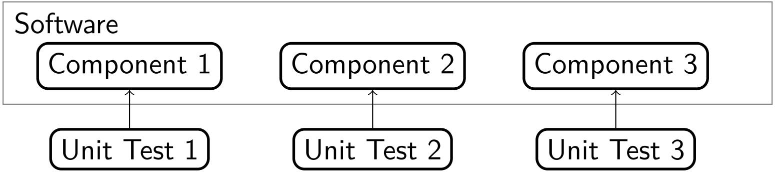
```
er buf
ingBuilder(16);
"Hello")
␣")
World");
toString();
```

Complex behaviour

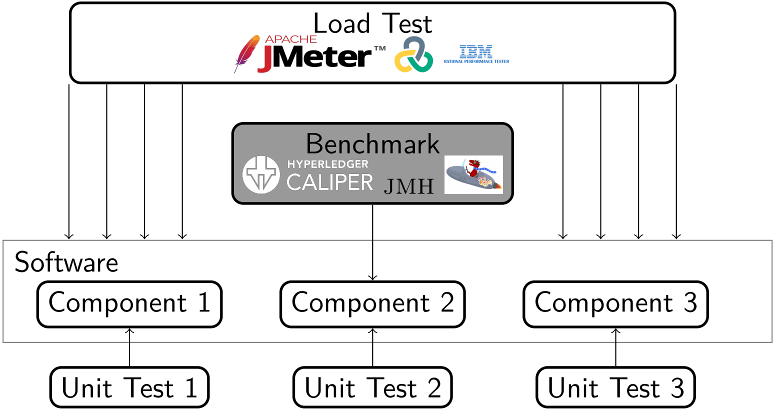
⇒ impact of changes needs to be measured

Peass Method

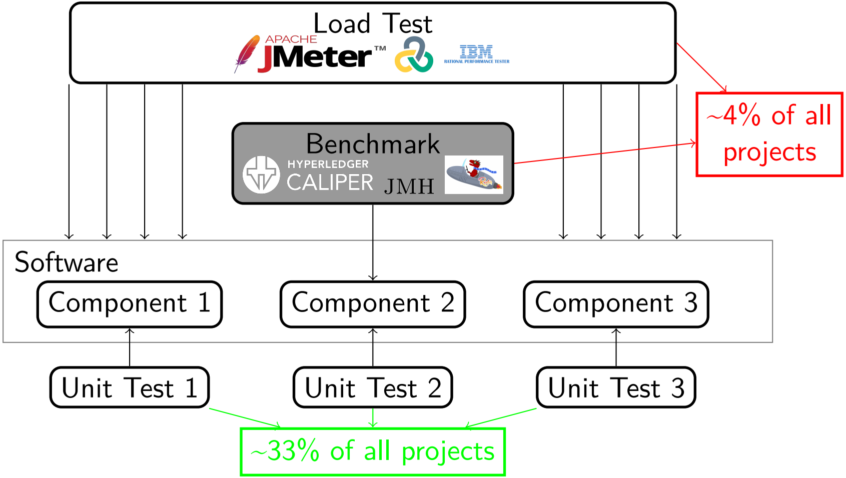
Why Unit Test Measurement?



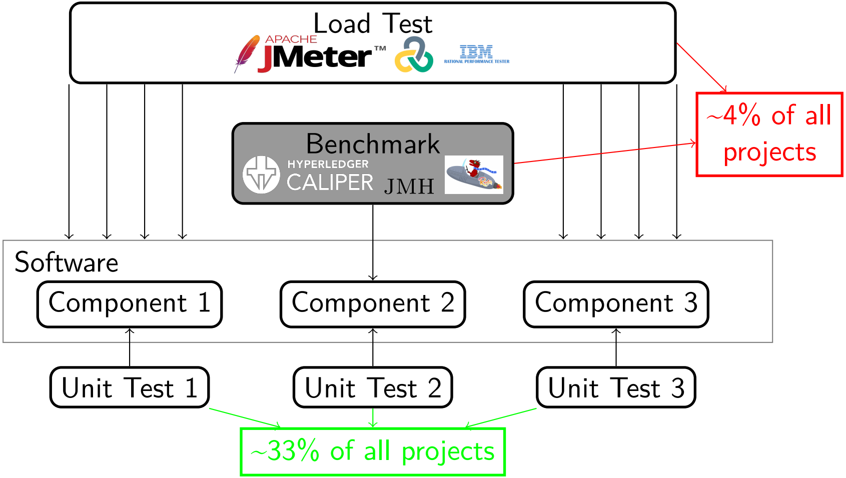
Why Unit Test Measurement?



Why Unit Test Measurement?



Why Unit Test Measurement?

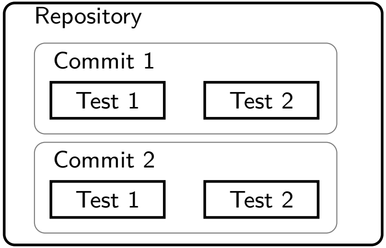


$\Rightarrow$ unit tests are transformed and used for measurement

Peass

- **P**erformance **a**nalysis of **s**oftware **s**ystems
- CLI-Tool: `https://github.com/DaGeRe/peass`
- Jenkins-Plugin: `https://github.com/jenkinsci/peass-ci-plugin`

Process of Peass



Process of Peass

Repository

Commit 1

Test 1 Test 2

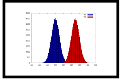
Commit 2

Test 1 Test 2

Measurement

Commit 2

Test 1 Test 2

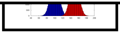


Process of Peass

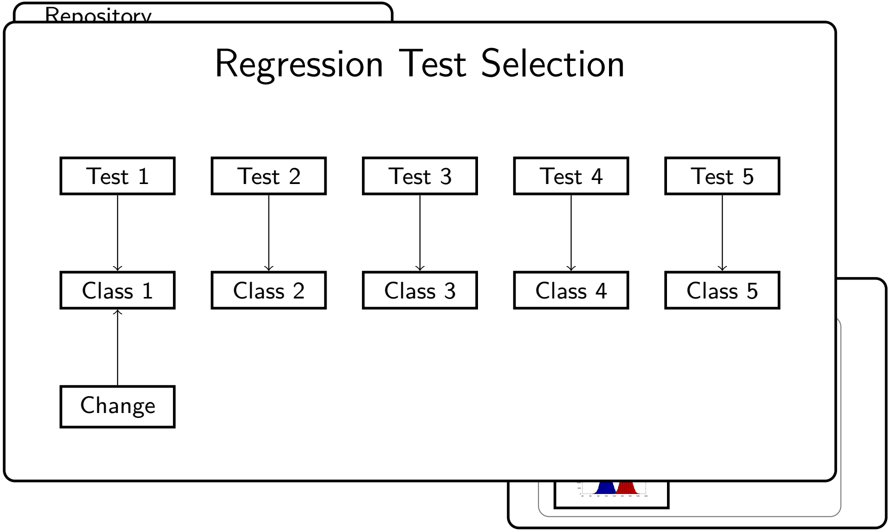
Repository

Regression Test Selection

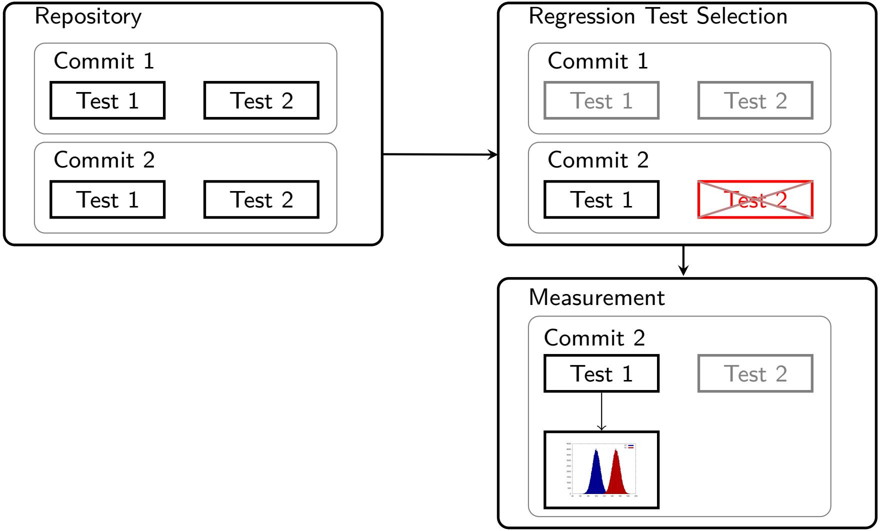
Test 1 | Test 2 | Test 3 | Test 4 | Test 5



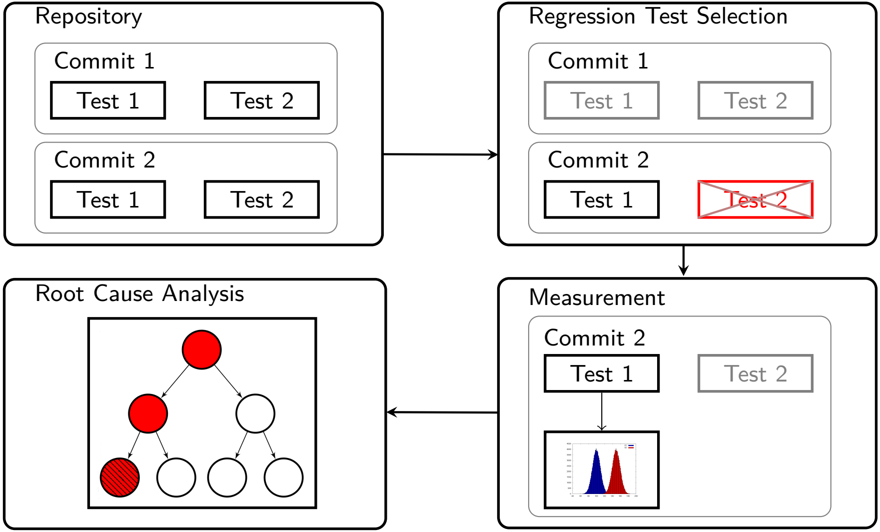
Process of Peass



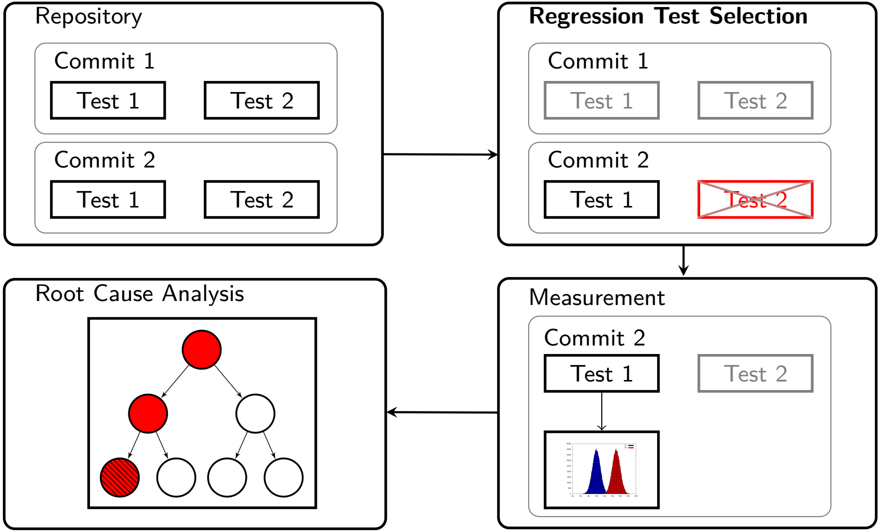
Process of Peass



Process of Peass



Process of Peass



Regression Test Selection - Approach [LTB18]

TestA

void testA1()
void testA2()

TestB

void testB1()
void testB2()

Class α

void methodα1()
void methodα2()

Class β

void methodβ1()
void methodβ2()

Class γ

void methodγ1()
void methodγ2()

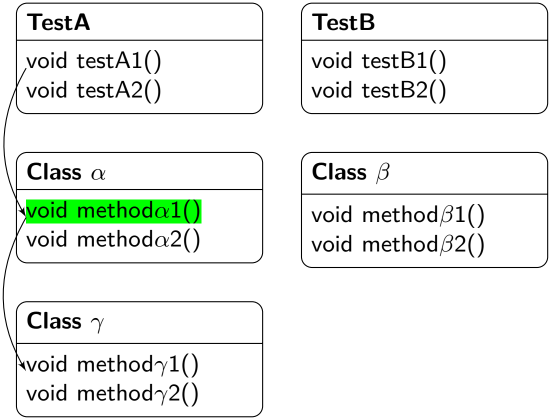
Trace:

TestA.testA1
Classα.methodα1
Classγ.methodγ1
Classα.methodα1
Classγ.methodγ1

Change:

...

Regression Test Selection - Approach [LTB18]



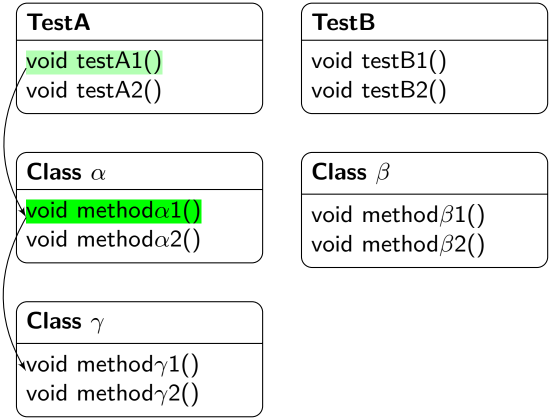
Trace:

TestA.testA1
Classα.methodα1
Classγ.methodγ1
Classα.methodα1
Classγ.methodγ1

Change:

Classα.methodα1

Regression Test Selection - Approach [LTB18]



Trace:

TestA.testA1

Classα.methodα1

Classγ.methodγ1

Classα.methodα1

Classγ.methodγ1

Change:

Classα.methodα1

Regression Test Selection - Approach [LTB18]

TestA
- void testA1()
- void testA2()

TestB
- void testB1()
- void testB2()

Class α
- void methodα1()
- void methodα2()

Class β
- void methodβ1()
- void methodβ2()

Class γ
- void methodγ1()
- void methodγ2()

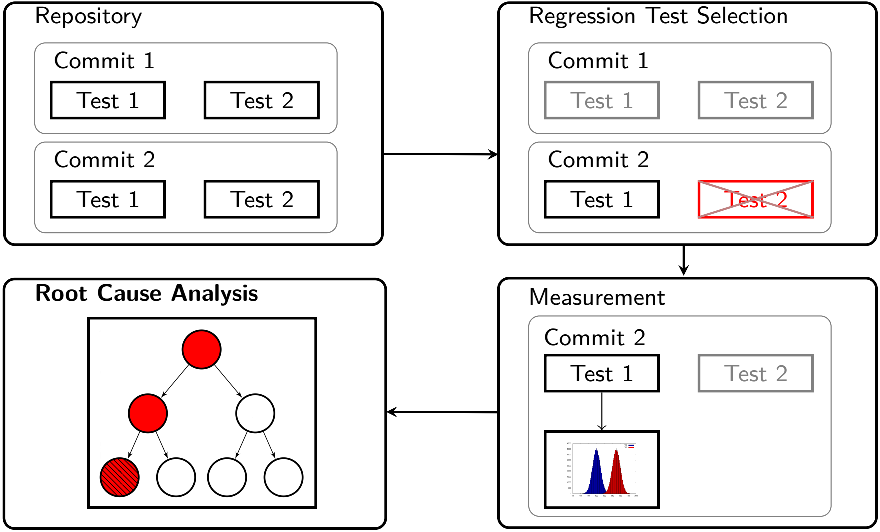
Trace:

TestA.testA1
Classα.methodα1
Classγ.methodγ1
Classα.methodα1
Classγ.methodγ1

Change:

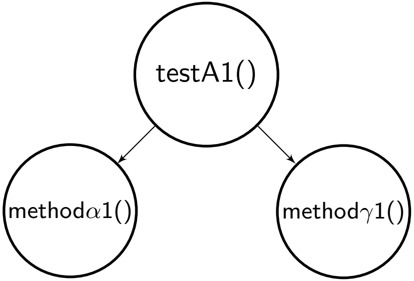
Classα.methodα2

Process of Peass



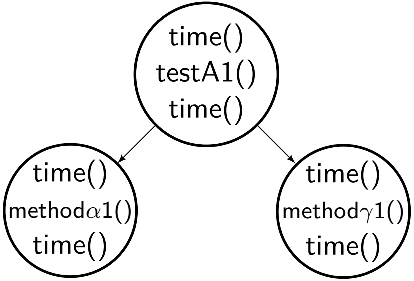
Root Cause Analysis

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



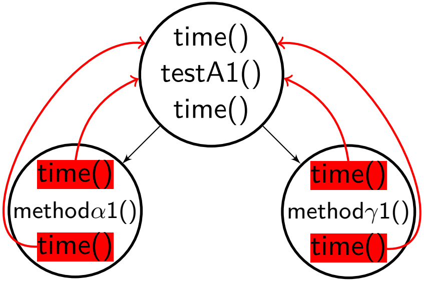
Root Cause Analysis

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



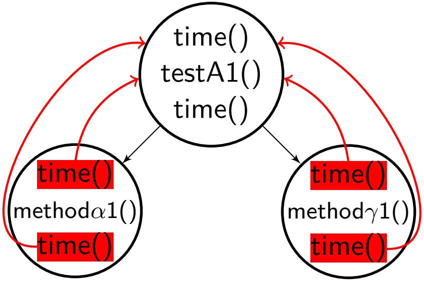
Root Cause Analysis

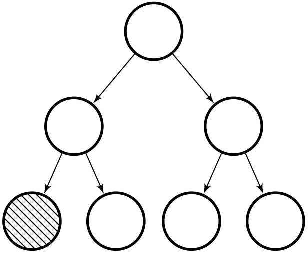
- Root cause analysis for performance changes (Heger et al., 2013)
  - Getting the call tree
  - Levelwise measurement of call tree nodes
  - `time()`-calls if methodα1() and methodγ1() influence measurement of `testA1()`



Root Cause Analysis

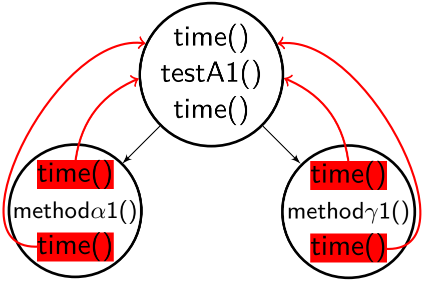
- Root cause analysis for performance changes (Heger et al., 2013)
  - Getting the call tree
  - Levelwise measurement of call tree nodes
  - `time()`-calls if methodα1() and methodγ1() influence measurement of `testA1()`

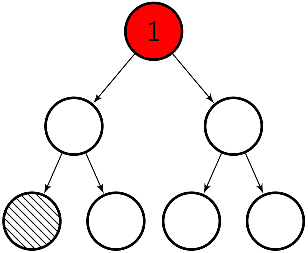




Root Cause Analysis

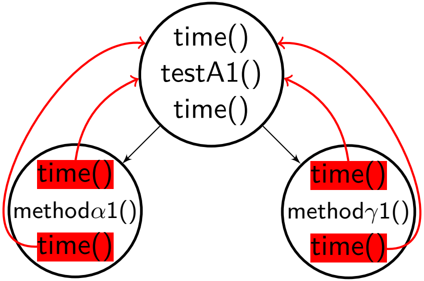
- Root cause analysis for performance changes (Heger et al., 2013)
  - Getting the call tree
  - Levelwise measurement of call tree nodes
  - `time()`-calls if methodα1() and methodγ1() influence measurement of `testA1()`

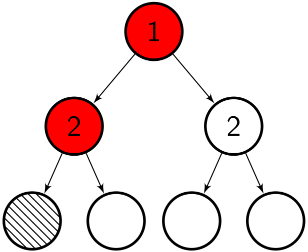




Root Cause Analysis

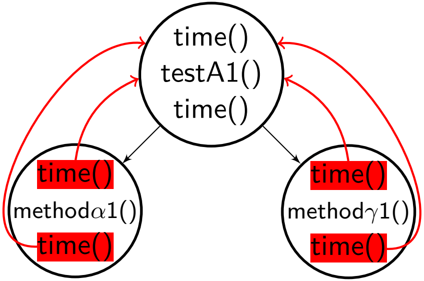
- Root cause analysis for performance changes (Heger et al., 2013)
  - Getting the call tree
  - Levelwise measurement of call tree nodes
  - `time()`-calls if methodα1() and methodγ1() influence measurement of `testA1()`

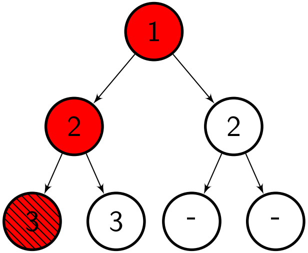




Root Cause Analysis

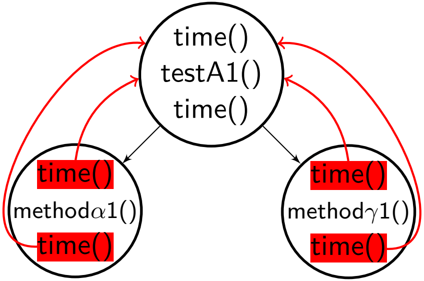
- Root cause analysis for performance changes (Heger et al., 2013)
  - Getting the call tree
  - Levelwise measurement of call tree nodes
  - `time()`-calls if methodα1() and methodγ1() influence measurement of `testA1()`

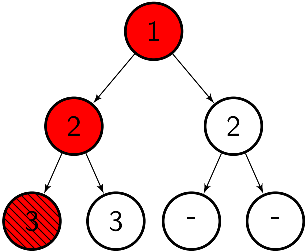




Root Cause Analysis

- Root cause analysis for performance changes (Heger et al., 2013)
  - Getting the call tree
  - Levelwise measurement of call tree nodes
  - `time()`-calls if methodα1() and methodγ1() influence measurement of `testA1()`

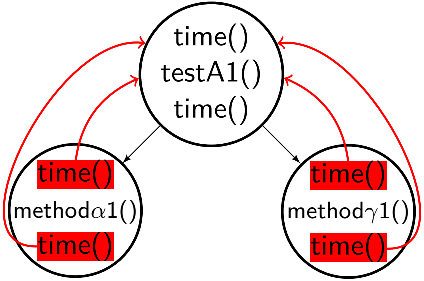


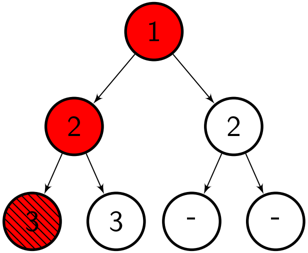


- Challenges for unit test measurement
  - Overhead influences measurement quality and duration
    $\Rightarrow$ Low overhead necessary
  - Configuration and node selection strategy a priory unknown

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 methodα1() and methodγ1() influence measurement of `testA1()`





- Challenges for unit test measurement
  - **Overhead influences measurement quality and duration**
    $\Rightarrow$ **Low overhead necessary**
  - Configuration and node selection strategy a priory unknown

Kieker Monitoring Process [WOSPC23]

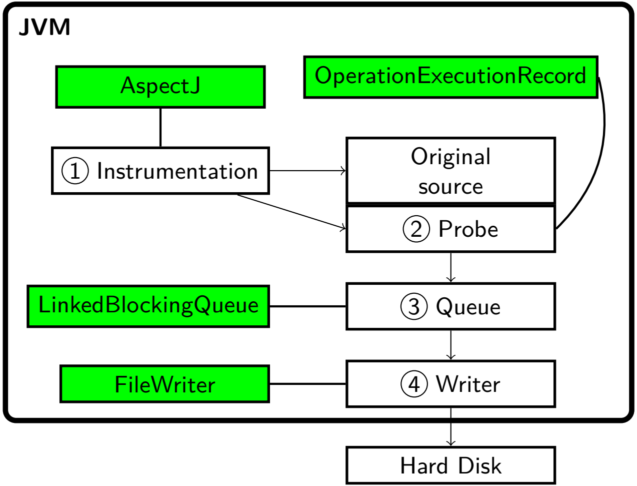


Figure: Monitoring Process in Kieker

Kieker Monitoring Process [WOSPC23]

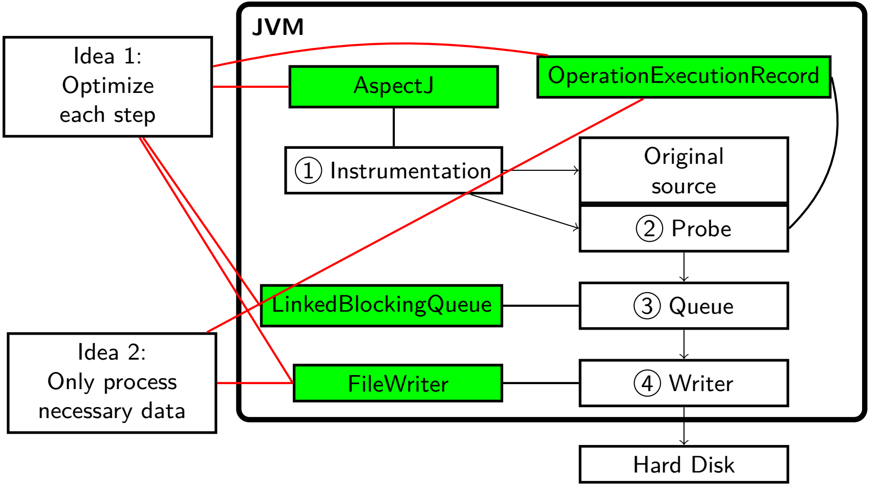


Figure: Monitoring Process in Kieker

Optimization Options [WOSPC23]

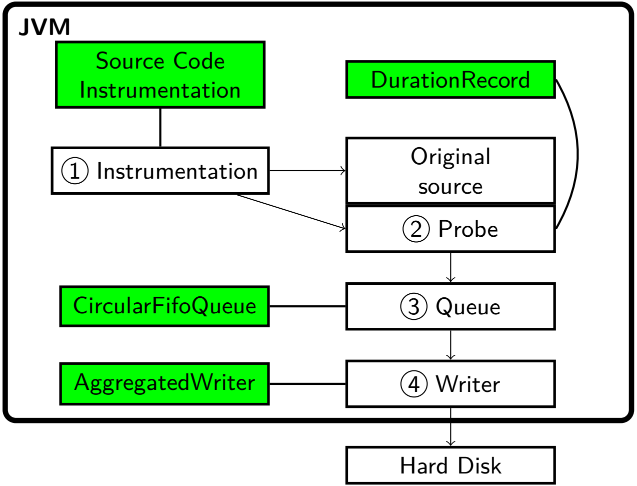
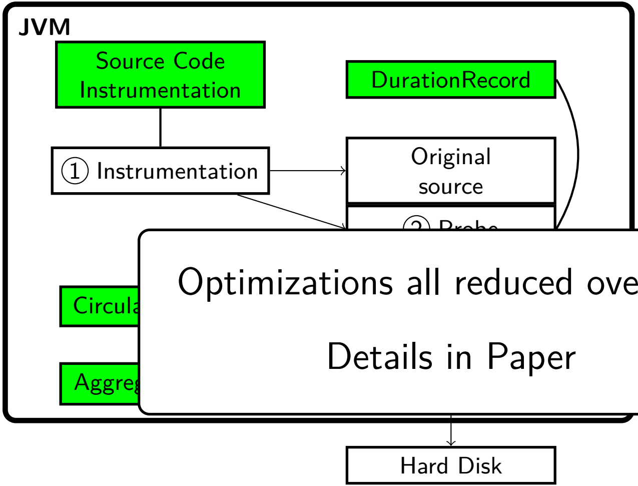


Figure: Possible Monitoring Optimizations

Optimization Options [WOSPC23]



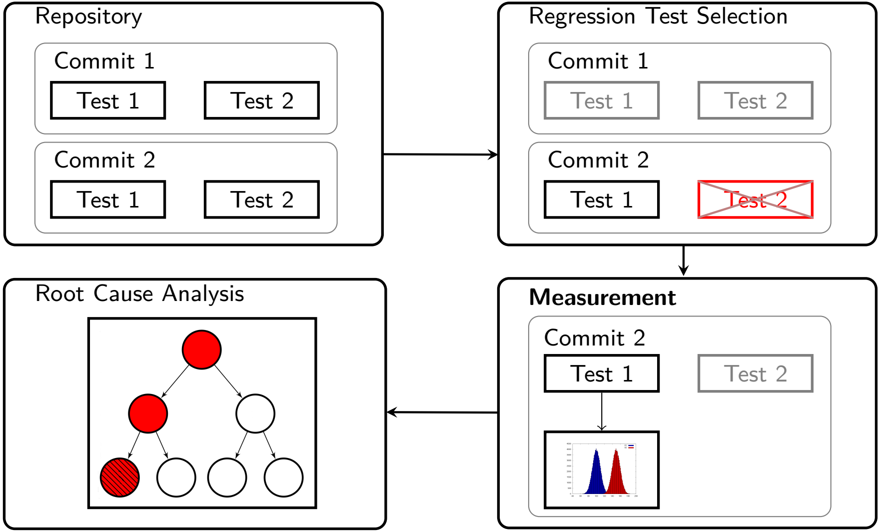
Optimizations all reduced overhead

Details in Paper

Figure: Possible Monitoring Optimizations

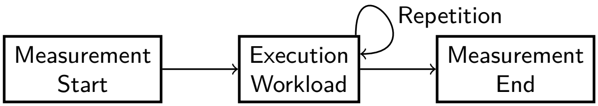
Performance Change Detection

Process of Peass

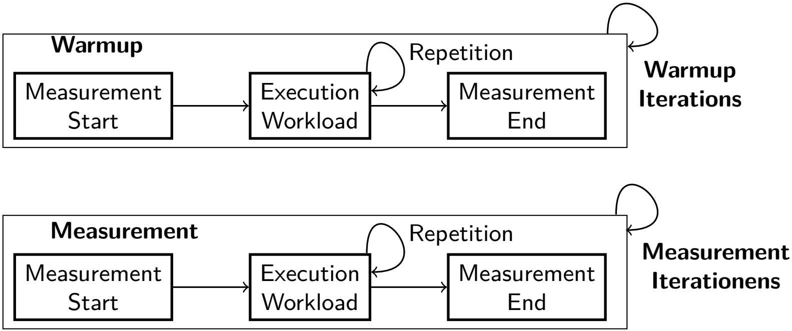


General Measurement Process (Adaption of Georges et al., 2007)

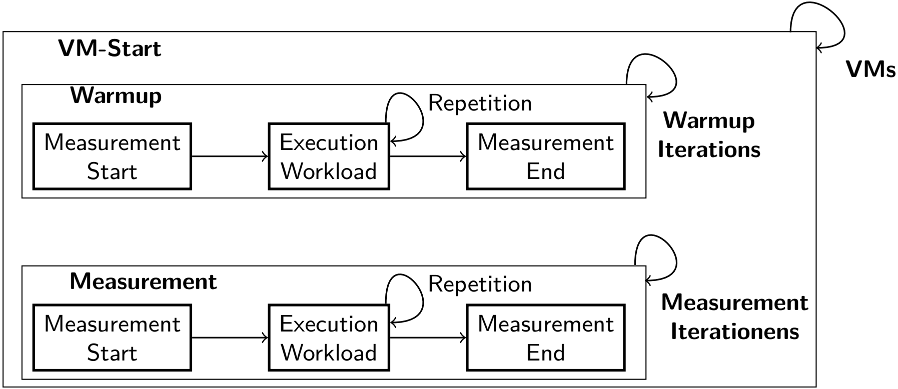
General Measurement Process (Adaption of Georges et al., 2007)



General Measurement Process (Adaption of Georges et al., 2007)



General Measurement Process (Adaption of Georges et al., 2007)



Example Worklods [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 Worklods [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 Worklods [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]

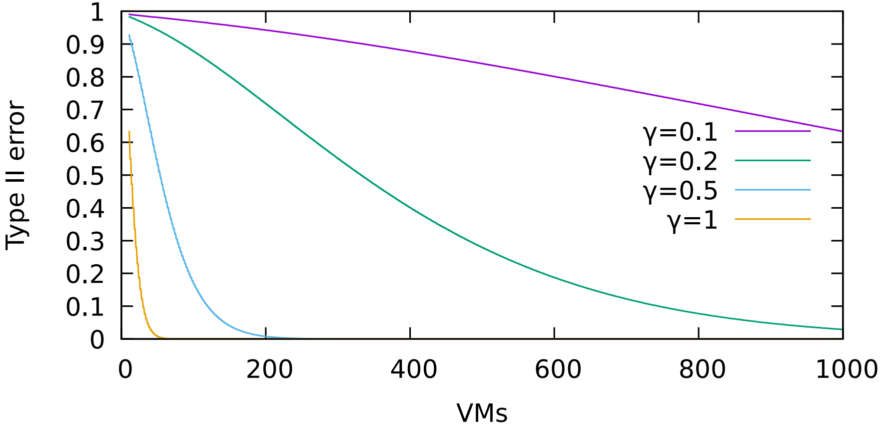


Figure: Type II error in relation to VM count and effect size γ

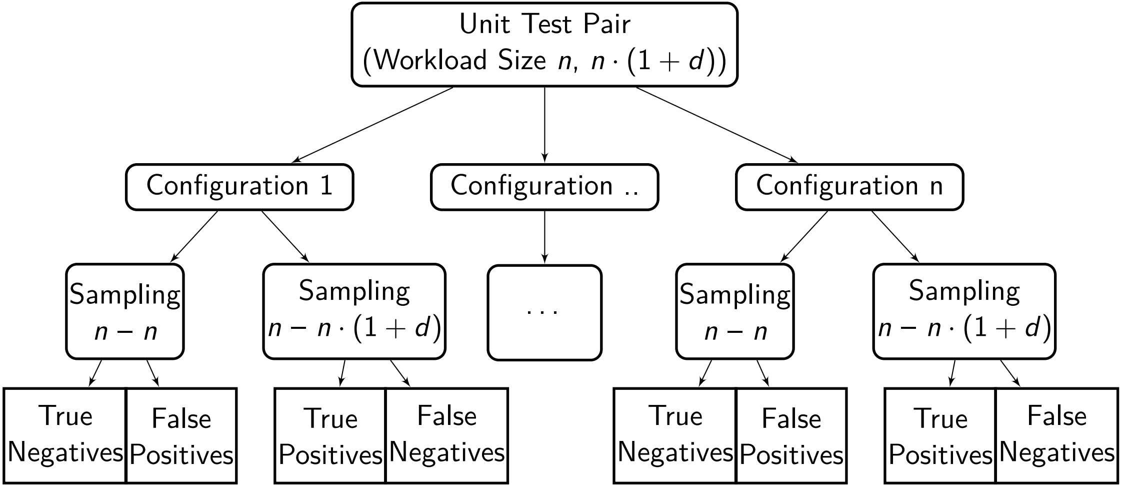
Statistical Boundaries of Detecting Changes [QRS22]

- Example: Workload size 1,000 will take 97 seconds for 10,000,000 repetitions $\Rightarrow$ 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

Statistical Boundaries of Detecting Changes [QRS22]

- Example: Workload size 1,000 will take 97 seconds for 10,000,000 repetitions $\Rightarrow$ 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 corse-grained performance changes

Approach [QRS22]



Workload Size [QRS22]

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

Workload Size [QRS22]

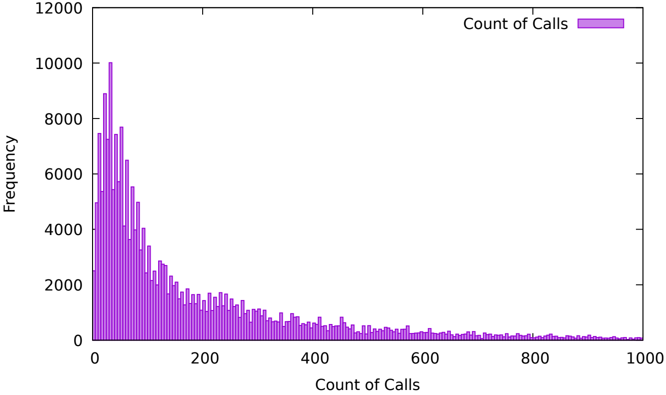
- Examination of unit test sized workloads
  $\Rightarrow$ What is a unit test size?
  (What is s?)

- 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]

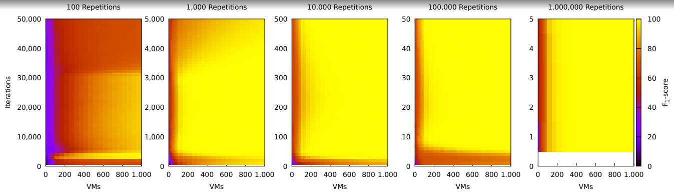
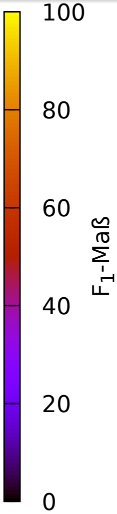


Figure: Average F_1-Score With T-Test



Example: Comparison of Statistical Tests [QRS22]

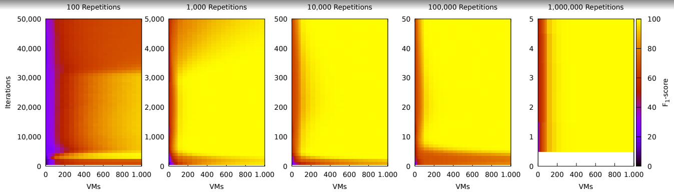


Figure: Average F_1-Score With T-Test

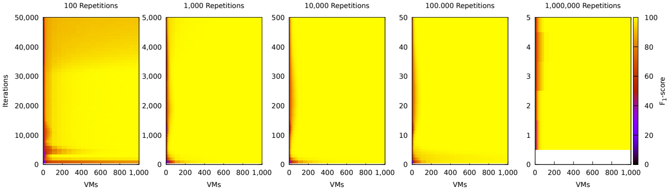
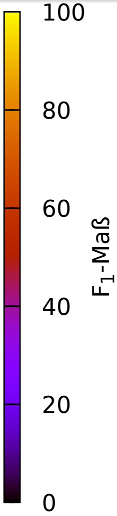
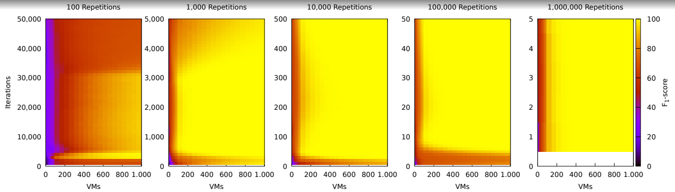


Figure: Average F_1-Score With Mann-Whitney-Test

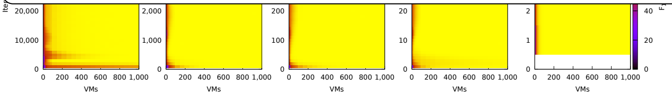


Example: Comparison of Statistical Tests [QRS22]



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

for typical unit tests



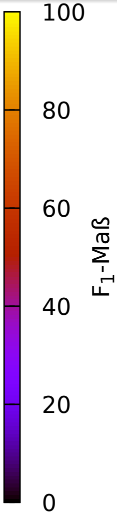
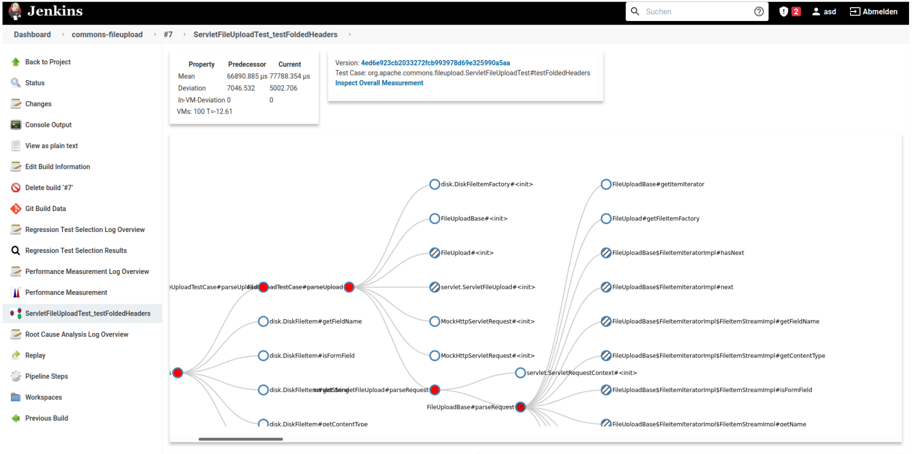
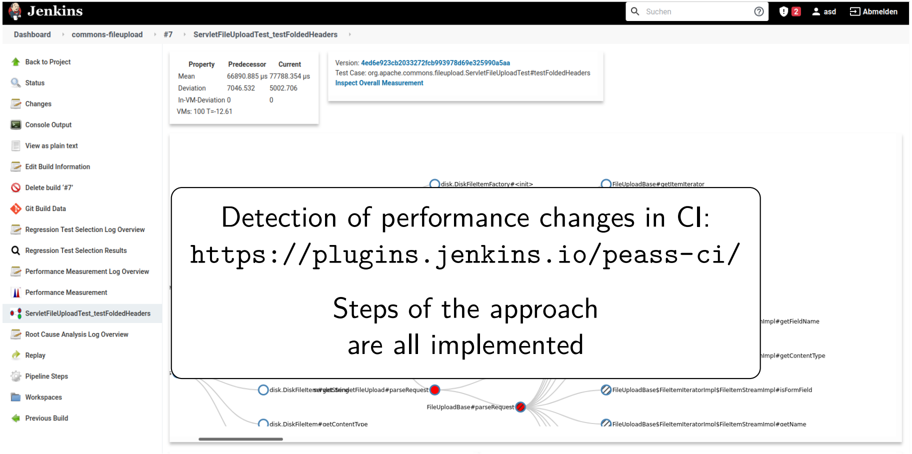


Figure: Average F_1-Score With Mann-Whitney-Test

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

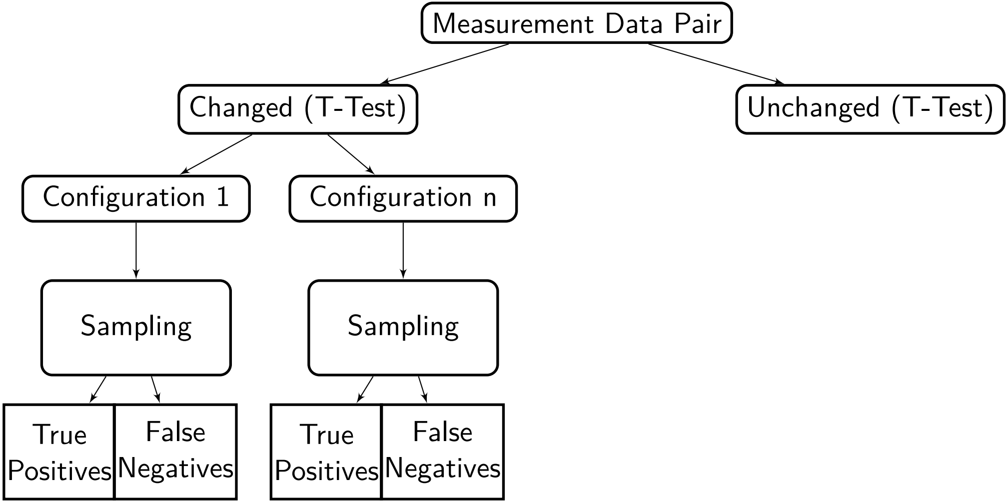
Application to GraalVM Data

Application to GraalVM Data – Idea

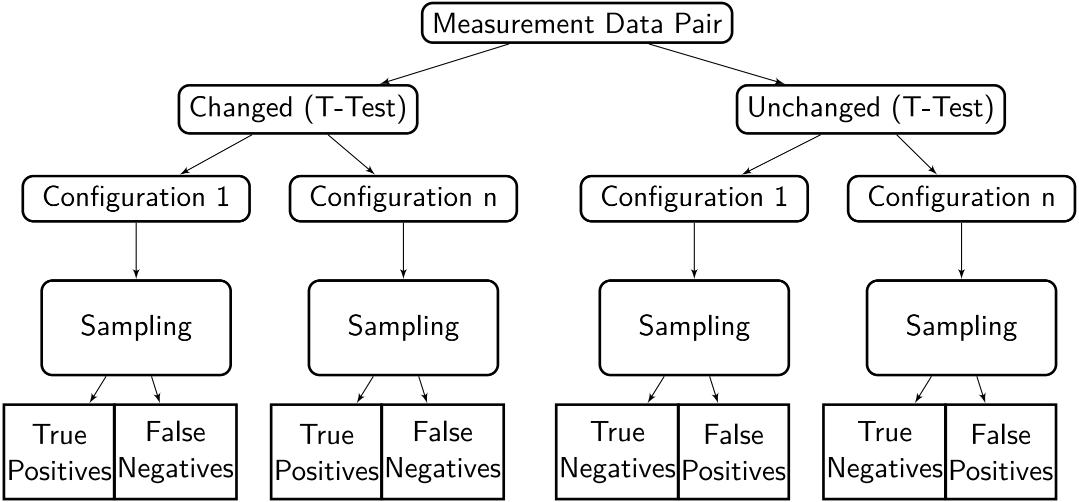
- Problem: GraalVM benchmarks might be executed to often
- Goal: Determination of suitable VM count and iteration / run count

- Problem: Artificial benchmark pairs and option to repeat them more often than necessary not available $\Rightarrow$ Approach: Sample from existing data

Application to GraalVM Data – Approach



Application to GraalVM Data – Approach



Application to GraalVM Data – Preleminary Results

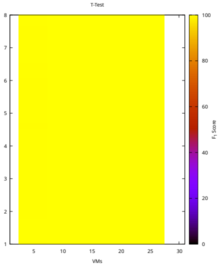


Figure: Difference 48994

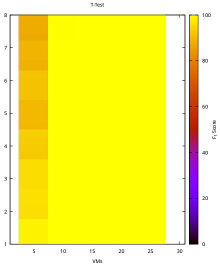


Figure: Difference 49001

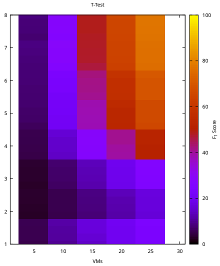


Figure: Difference 50180

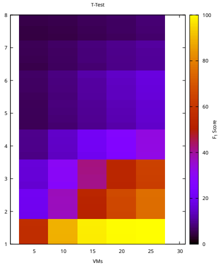


Figure: Difference 50405

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.g.reichelt [a t] lancaster.ac.uk