Continuous Integration in Kieker
(Experience Report)

Nils Christian Ehmke, Christian Wulf, and Wilhelm Hasselbring
Software Engineering Group, Kiel University, Germany
A Short Outline

1. An Introduction to Continuous Integration

2. Continuous Integration in Kieker

3. Conclusion
A Short Outline

1. An Introduction to Continuous Integration

2. Continuous Integration in Kieker

3. Conclusion
The Meaning of CI

• “Continuous Integration is a software development practice where members of a team integrate their work frequently [...]” (Martin Fowler)

• Origins in Extreme Programming (XP)

• Goals:
  – Increase and monitor software quality
  – Reduce effort for developers
  – Increase communication

• Realization:
  – Automate as much as possible (build, tests, ...)

The diagram illustrates the cycle of writing code, performing automated builds, performing automated tests, and publishing reports.
Some Practices of CI [Based on Fowler and Foemmel 2006] => CI Server (e.g., Jenkins)
- Maintain a single source repository => VCS (e.g., Git)
- Automate the build => Build tools (e.g., Ant, Gradle)
- Make your build self-testing => Testing (e.g., JUnit)
- Keep the build fast
- Make it easy [...] to get the latest executable => Snapshots
- Everyone can see what's happening => Reports

Task Management (e.g., Trac, Jira)
The Impact of CI

• „Given all of the above, teams moving to a CI driven process can expect to achieve at least a 40% reduction in check-in overhead when compared to a check-in process that maintains the same level of code base and product quality.” (Ade Miller, 2008)

• 2013: 68% use CI¹
• 2014: 78.5% use CI²

A Short Outline

1. An Introduction to Continuous Integration

2. Continuous Integration in Kieker

3. Conclusion
CI in Kieker

"Maintain a single source repository"
CI in Kieker

“Automate the build”
CI in Kieker

“Automate the build (cont’d)”
CI in Kieker

“Make your build self-testing”

- Unit- and Integration-Tests (JUnit)
- Static Code Analysis (PMD, Checkstyle, Findbugs)
- Regression Benchmarking (MooBench)

Jan Waller, Florian Fittkau and Wilhelm Hasselbring

Application Performance Monitoring: Trade-Off between Overhead Reduction and Maintainability
“Make your build self-testing (cont’d)”

### All Tests

<table>
<thead>
<tr>
<th>Package</th>
<th>Duration</th>
<th>Failures</th>
<th>Skips</th>
<th>Passes</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>kiekerTest:analysis:unit:configuration</td>
<td>0.37 s</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>kiekerTest:analysis:unit:plugin</td>
<td>8.6 ms</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>kiekerTest:analysis:unit:plugin:filterflow</td>
<td>3.6 s</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>kiekerTest:analysis:unit:plugin:filtertemplate</td>
<td>55 s</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>kiekerTest:analysis:unit:plugin:filteraest</td>
<td>11 ms</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>kiekerTest:analysis:unit:plugin:filtertrage</td>
<td>68 ms</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>kiekerTest:analysis:unit:plugin:filtervisualization</td>
<td>31 ms</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>kiekerTest:analysis:unit:plugin:reader</td>
<td>29 ms</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>kiekerTest:analysis:unit:plugin:reader:filteraestsystem</td>
<td>2 s</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>kiekerTest:analysis:unit:plugin:plugin:namedRecordflow</td>
<td>63 ms</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>kiekerTest:common:unit:configuration</td>
<td>3.6 s</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>kiekerTest:common:unit:configuration</td>
<td>7 ms</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>kiekerTest:common:unit:record</td>
<td>21 ms</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>kiekerTest:common:unit:record:controlflow</td>
<td>25 ms</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>kiekerTest:common:unit:record:flow:trace</td>
<td>33 ms</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>kiekerTest:common:unit:record:flow:trace:concurrency</td>
<td>4 ms</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>kiekerTest:common:unit:record:flow:trace:concurrency:monitor</td>
<td>0.1 s</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>kiekerTest:common:unit:record:flow:trace:operation</td>
<td>0.13 s</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>kiekerTest:common:unit:record:flow:trace:operation:constructor</td>
<td>81 ms</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>kiekerTest:common:unit:record:flow:trace:operation:object</td>
<td>74 ms</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>kiekerTest:common:unit:util</td>
<td>16 ms</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

*Tests (42)*  
Duration: 3 minutes 42 seconds  
Description:  

---


Nils Christian Ehmke, Christian Wulf, and Wilhelm Hasselbring — Continuous Integration in Kieker
CI in Kieker

“Make your build self-testing (cont’d)”
CI in Kieker

“Make your build self-testing (cont’d)”

Mean Overhead of Kieker

- No Probe
- Deactivated Probe
- Collecting Data
- Writer (ASCII)
- Writer (Bin)
- Writer (TCP)
CI in Kieker

“Make your build self-testing (cont’d)”

[Bar chart showing mean response times over time with specific annotations and data points.]
CI in Kieker

“Everyone can see what’s happening”
### Projekt kieker-nightly-release

#### Java doc

#### Arbeitsbereich

**Arbeitsblätter des letzten erfolgreichen Builds**

- kieker:1.10-SNAPSHOT_binaries.tar.gz
- kieker:1.10-SNAPSHOT_binaries.zip
- kieker:1.10-SNAPSHOT_javadoc.tar.gz
- kieker:1.10-SNAPSHOT_javadoc.zip
- kieker:1.10-SNAPSHOT_sources.tar.gz
- kieker:1.10-SNAPSHOT_sources.zip
- kieker:1.10-SNAPSHOT_userguide.pdf

#### Letzte Änderungen

- Letzter Build (#2141), vor 5 Stunden 7 Minuten
- Letzter erfolgreicher Build (#2138), vor 6 Tage 5 Stunden
- Letzter erfolgreicher Build (#2144), vor 5 Stunden 7 Minuten

#### Permalinks

- Letzter Build (#2141), vor 5 Stunden 7 Minuten
- Letzter erfolgreicher Build (#2138), vor 6 Tage 5 Stunden
- Letzter erfolgreicher Build (#2144), vor 5 Stunden 7 Minuten
CI in Kieker

Task Management

#1126 new feature

Adaptive monitoring based on dynamic instrumentation

Ersteller: avh
Priorität: major
Komponente: software-monitoring
Stichworte: thesis

Beschreibung

Adaptive monitoring in Kieker is currently not based on dynamic instrumentation. However, this would be desirable, e.g., based on JavaAssist and/or runtime class redefinitions. @inspectIT has support for this.

A student from Stuttgart will write a Bachelor's thesis on this topic (currently at the beginning of the proposal phase). For the evaluation, which is essentially a comparison of the overhead of the current implementation with the new one, we were thinking about using/adopting MoosBench.

@Jan: Maybe we can briefly talk about this on Monday.
A Short Outline

1. An Introduction to Continuous Integration

2. Continuous Integration in Kieker

3. Conclusion
Conclusion

• Benefits:
  – Early warnings and feedbacks for the developers
  – Avoid corruption of anything without noticing
  – Automated a lot of jobs
  – More relaxing releases

• Challenges:
  – CI does need some initial effort
  – Keep CI in synchronisation with the project
  – Complex system for new developers
Bibliography

• Martin Fowler and Matthew Foemmel; Continuous Integration; 2006
• RebelLabs; Developer Productivity Report 2013; 2013
• RebelLabs; Java Tools and Technologies Landscape for 2014; 2014
• Ade Miller; A Hundred Days of Continuous Integration; 2008; AGILE ‘08 Conference; Pages 289-293