

Application Performance Monitoring & Architecture Discovery with Kieker

Prof. Dr. Wilhelm (Willi) Hasselbring

Competence Cluster Software Systems Engineering

<http://www.kosse-sh.de/>

Software Engineering Group, Kiel University

<http://se.informatik.uni-kiel.de/>

WAIS Seminar

Southampton, April, 23rd 2014



Thanks to Contributors

Several persons contributed to Kieker and the contents of this presentation in the past eight years:

*Tillmann (Till) Bielefeld, Peer Brauer, Philipp Döhring,
Jens Ehlers, Nils Ehmke, Florian Fittkau, Thilo Focke,
Sören Frey, Tom Frotscher, Henry Grow, André van Hoorn,
Reiner Jung, Benjamin Kiel, Dennis Kieselhorst, Holger Knoche,
Arnd Lange, Marius Löwe, Marco Lübcke, Felix Magedanz,
Nina Marwede, Robert von Massow, Jasminka Matevska,
Oliver Preikszas, Sönke Reimer, Bettual Richter, Matthias Rohr,
Nils Sommer, Lena Stöver, Jan Waller, Robin Weiß,
Björn Weißenfels, Matthias Westphal, Christian Wulf*

— Alphabetic list of people who contributed in various forms and intensity

- 1 Dynamic Software Analysis
- 2 Kieker Monitoring & Analysis Framework
- 3 Application Performance Monitoring
- 4 Reverse Engineering
- 5 Summary and Future Work

Static Analysis of Software Systems

To distinguish it from dynamic analysis

Dynamic Software Analysis



Static analysis may be applied to the program code to check the programs quality for various concerns, some examples:

Clone detection Bauhaus

Code metrics JavaNCSS

Coding conventions Checkstyle

Fault patterns FindBugs

Architecture analysis Sotograph



Dynamic analysis,
or the *analysis of data gathered from
a running program.*

has the potential to
provide an **accurate picture**
of a software^{system} **system's actual behavior.**
because it exposes the

This picture can range
from **class-level details**

up to
**high-level
architectural views** [...]



(Cornelissen et al. 2009)

Among the **benefits over static analysis** are the **availability of runtime information** and, in the context of object-oriented software, the **exposure of object identities** and the **actual resolution of late binding**.

A **drawback** is

that **dynamic analysis** can only provide a **partial picture of the system**, i.e., the results obtained are valid for the **scenarios that were exercised during the analysis**.

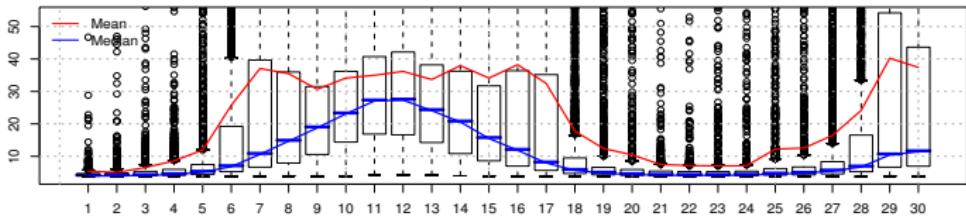
(Cornelissen et al. 2009)

- Static analysis alone is not sufficient to study the internal behavior of software systems comprehensively
- Continuous monitoring allows to gather a system's **actual** runtime behavior resulting from production usage profiles
- So, how to collect runtime data from running systems? — Instrumentation
 - Profiling — employed in development environments
 - Monitoring — employed in production environments
- An comprehension of the requirements of system **operation** is required to do the right things at **development** time
 - Instrument upfront!
 - Integrate with continuous integration and delivery.
 - Have you heard of DevOps ?

How does Monitoring work?

Example instrumentation: Measuring response times

Dynamic Software Analysis



Instrumentation

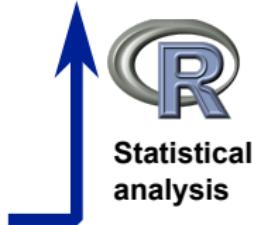
```
84 public static void searchBook() {  
85     long tin = System.currentTimeMillis();  
86     Catalog.getBook(false);  
87     long tout = System.currentTimeMillis();  
88     System.out.printf("Catalog;getBook;%s;%s", tin, tout);  
89 }  
90 }  
91 }
```

Visualization of results

Program execution

```
Catalog;getBook;1273333822196;1273333822202  
Catalog;getBook;1273333823197;1273333823199  
Catalog;getBook;1273333824198;1273333824200  
Catalog;getBook;1273333825199;1273333825201  
Catalog;getBook;1273333826200;1273333826202
```

Monitoring log



Challenges

Flexibility (instrumentation, changing log formats), runtime overhead, data volume, data distribution, online analysis, legacy systems, ...

Type of Monitoring Probes

Depends on analysis goals

Number and Position of Monitoring Points

Trade-off between performance overhead and information quality

Consideration of the Induced Monitoring Overhead

During continuous operation: only small overhead acceptable

Physical Location of the Monitoring Log/Stream

E.g., database, file system, queue, distributed deployment

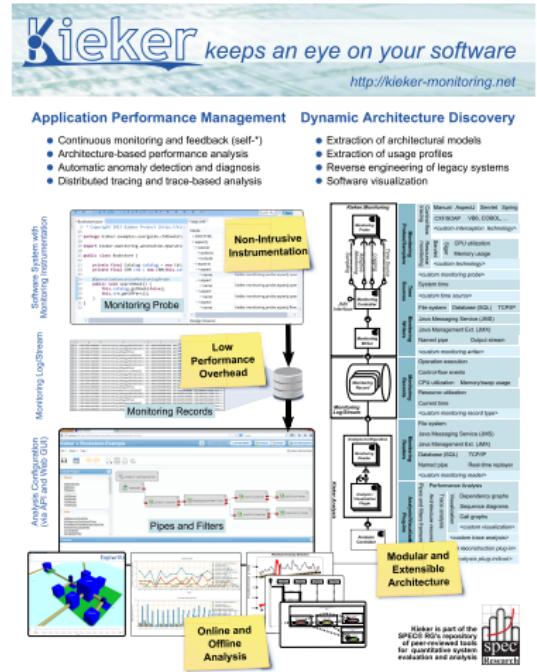
Intrusiveness of Instrumentation

Don't mix monitoring logic with business functionality → AOP

Agenda

Kieker Monitoring & Analysis Framework

- 1 Dynamic Software Analysis
- 2 Kieker Monitoring & Analysis Framework
- 3 Application Performance Monitoring
- 4 Reverse Engineering
 - Reverse Engineering of Java EE
 - Reverse Engineering of C#
 - Reverse Engineering of Visual Basic
 - Reverse Engineering of COBOL
 - Reverse Engineering of C / C++
 - Reverse Engineering of Perl
 - Kieker in Space
- 5 Summary and Future Work



Framework Features & Extension Points



Essential Characteristics [Rohr et al. 2008, van Hoorn et al. 2009b; 2012]

Kieker Monitoring & Analysis Framework



- Modular, flexible, and extensible architecture
(Probes, records, readers, writers, filters etc.)
 - Pipes-and-filters framework for analysis configuration
 - Distributed tracing (logging, reconstruction, visualization)
 - Low overhead (designed for continuous operation)
 - Evaluated in lab and industrial case studies



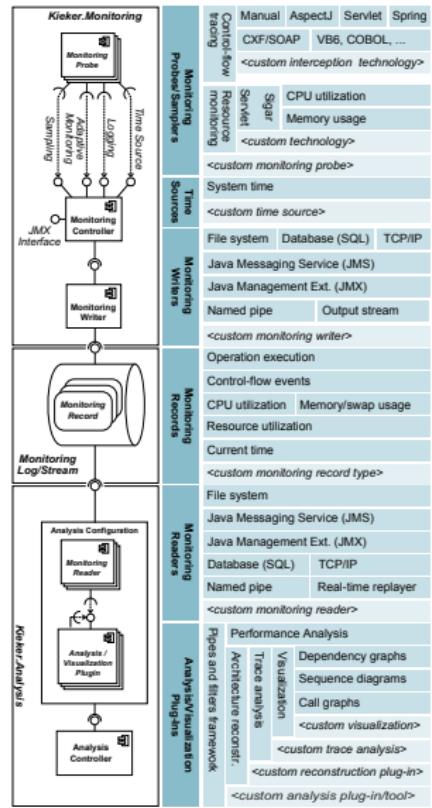
Kieker is open-source software (Apache License, V. 2.0)

<http://kieker-monitoring.net>

Recommended Tool of the SPEC Research Group

Kieker is distributed as part of SPEC RG's repository of peer-reviewed tools for quantitative system evaluation and analysis,

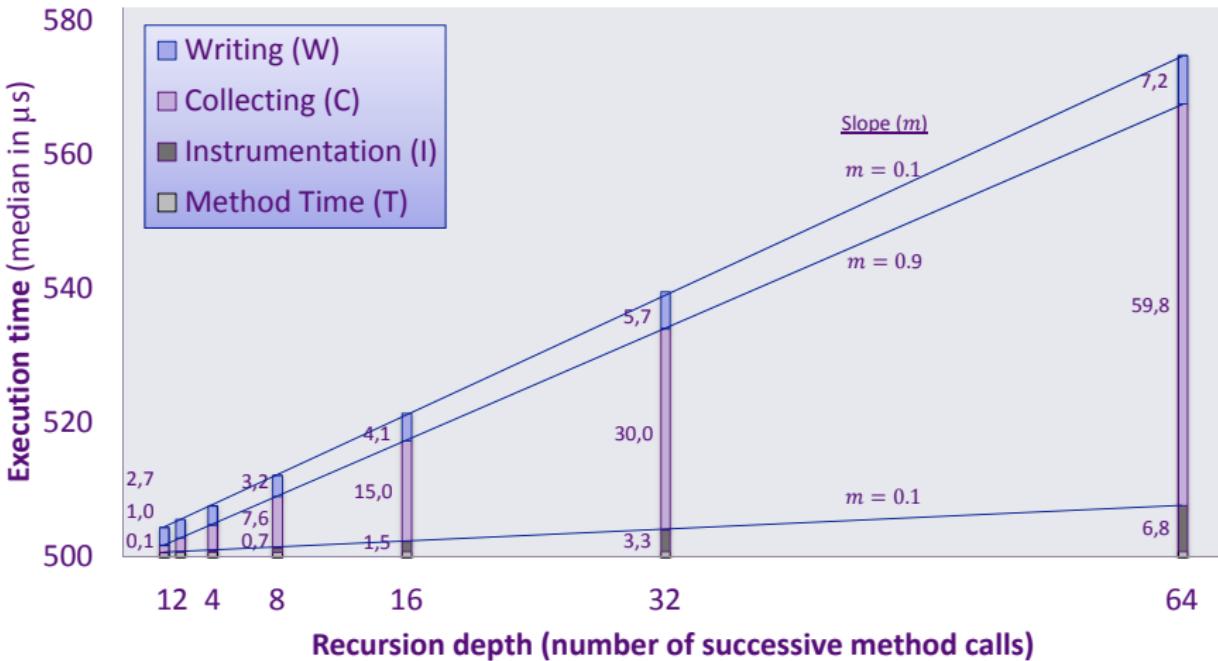
<http://research.spec.org/projects/tools.html>



Overhead Evaluation

[Waller and Hasselbring 2012]

Kieker Monitoring & Analysis Framework



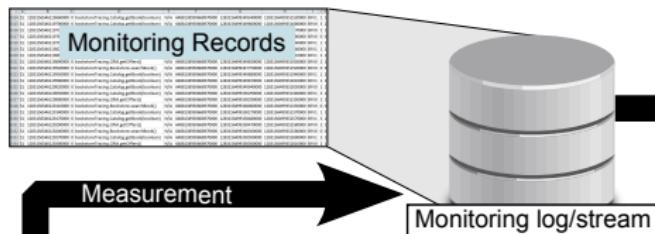
Experiment similar to:

A5 AsyncFS writer 16 cores whole system is available

Kieker: Dynamic Analysis Workflow

[van Hoorn et al. 2012]

Kieker Monitoring & Analysis Framework



This section shows the 'Monitoring Probe' component. It includes a Java code snippet for a 'Bookstore' class and its associated XML configuration file ('aop.xml'). The code defines a probe that searches for books in a catalog and checks if offers are available. The configuration file specifies various aspects for monitoring operations.

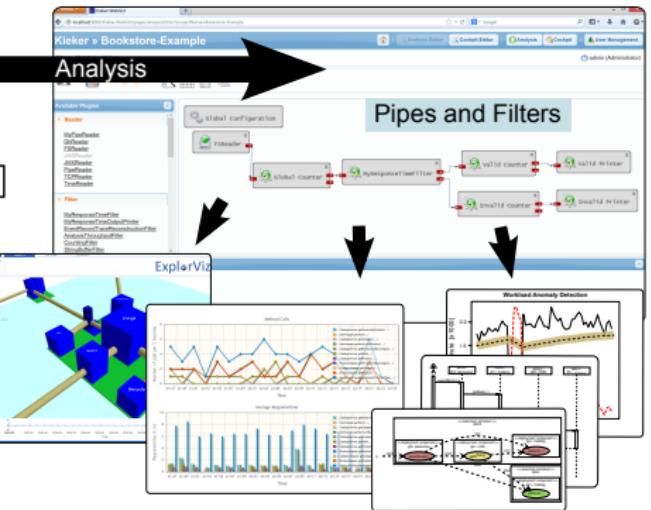
```
#Bookstore.java
1 // Copyright 2013 Kieker Project (http://kieker.org)
2 package kieker.examples.userguide.chBookstore;
3
4 import kieker.monitoring.annotation.Operation;
5
6 public class Bookstore {
7     private final Catalog catalog = new Catalog();
8     private final CRM CRM = new CRM(this.catalog);
9
10    @OperationExecutionMonitoringProbe
11    public void searchBook() {
12        this.catalog.getBook(false);
13        this.CRM.getOffers();
14    }
15 }
16
17 package kieker.monitoring.annotation.Operation;
18
19 import kieker.monitoring.annotation.Operation;
20
21 public class Bookstore {
22     private final Catalog catalog = new Catalog();
23     private final CRM CRM = new CRM(this.catalog);
24
25    @OperationExecutionMonitoringProbe
26    public void searchBook() {
27        this.catalog.getBook(false);
28        this.CRM.getOffers();
29    }
30 }
31
32 
```

Monitoring Probe

```
<aop.xml>
<?xml version="1.0" encoding="UTF-8"?>
<AspectJ</AspectJ>
<aspects>
<aspect name="kieker.monitoring.probe.aspectj.operation">
<operations>
<operation name="searchBook" pointcut="execution(* kieker.examples.userguide.chBookstore.Bookstore.searchBook())"/>
</operations>
</aspect>
<aspect name="kieker.monitoring.probe.aspectj.operation">
<operations>
<operation name="getOffers" pointcut="execution(* kieker.examples.userguide.chBookstore.Bookstore.getOffers())"/>
</operations>
</aspect>
<aspect name="kieker.monitoring.probe.aspectj.operation">
<operations>
<operation name="getBook" pointcut="execution(* kieker.examples.userguide.chBookstore.Bookstore.getBook(boolean))"/>
</operations>
</aspect>
<aspect name="kieker.monitoring.probe.aspectj.flow">
<operations>
<operation name="getBook" pointcut="execution(* kieker.examples.userguide.chBookstore.Bookstore.getBook(boolean))"/>
</operations>
</aspect>
</aspects>
<weaver>
<include name="kieker.monitoring.probe.aspectj.operation" />
</weaver>
<options>
<option name="kieker-monitoring-probe-aspectj-operation" value="true" />
</options>
</AspectJ>
```

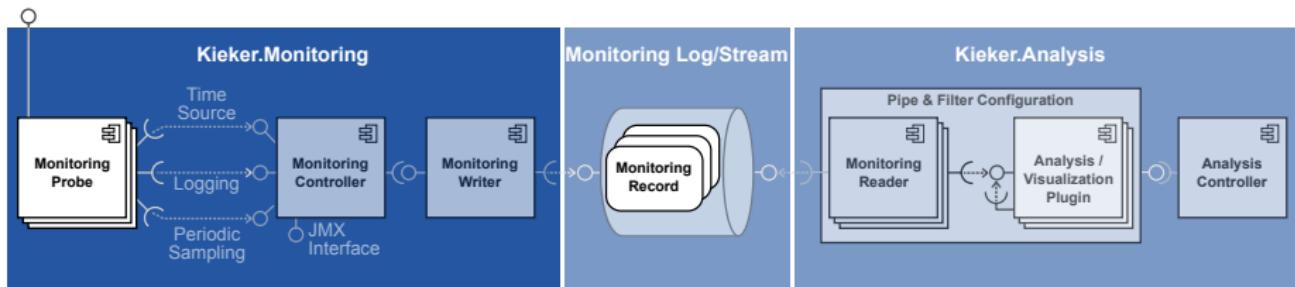
Software System with Monitoring Instrumentation

Analysis Configuration (via API and WebGUI)



Online and Offline Visualization

Core Kieker Framework Components



Java probes/samplers:

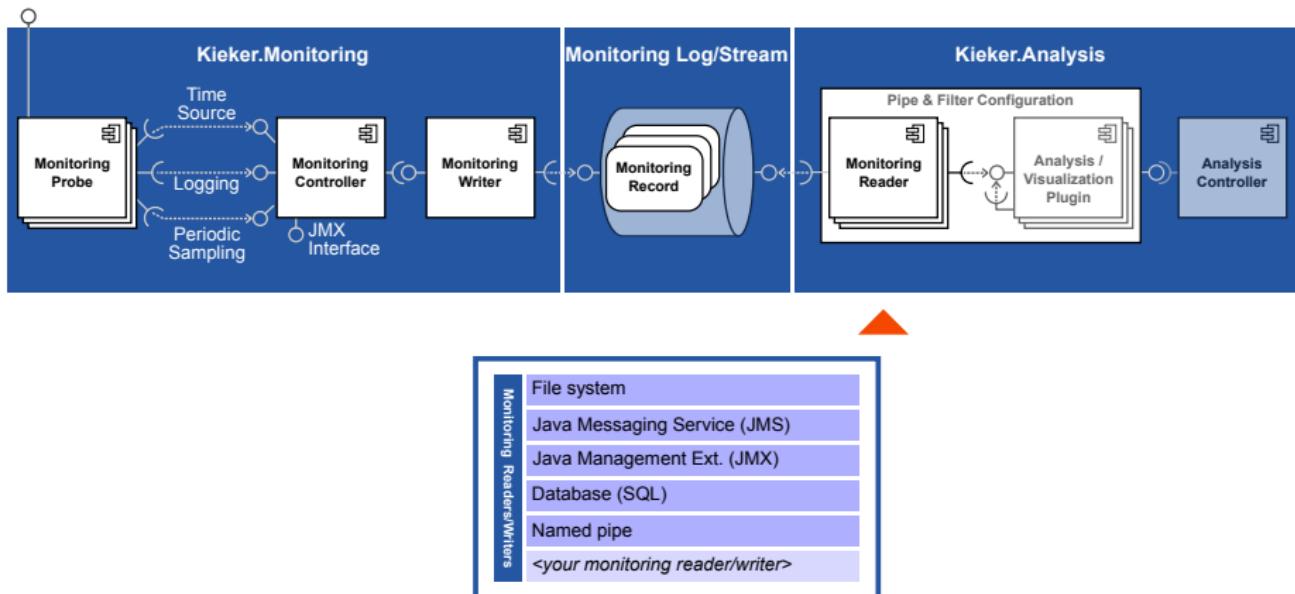
Monitoring Probes/Samplers	Manual instrumentation	
	Control-flow tracing	AspectJ
Resource monitoring	Servlet	Spring
	<your interception technology>	
Performance monitoring	Servlet	CPU utilization
	Sigar	Memory usage
Network monitoring	<your technology>	
	<your monitoring probe>	

+ basic adapters for

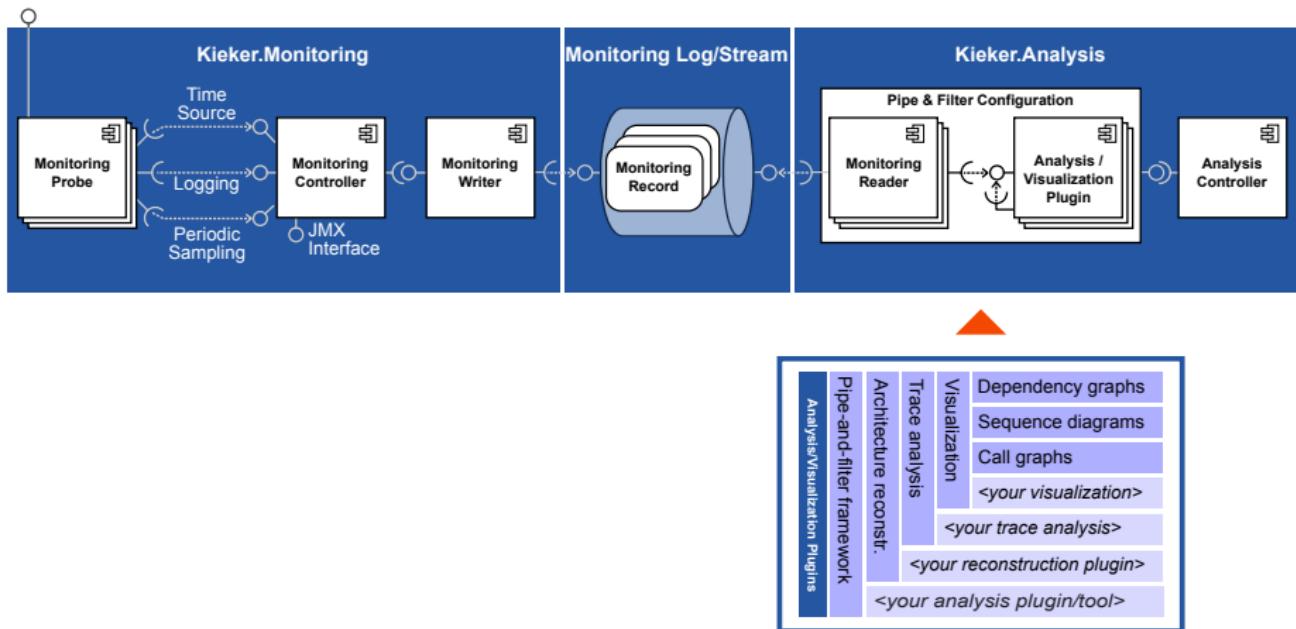
- C#/.NET
- Visual Basic 6/COM
- COBOL

Monitoring Records	
Operation execution	
Control-flow events	
CPU utilization	Memory/swapping usage
Resource utilization	
Current time	
<your monitoring record type>	

Core Kieker Framework Components



Core Kieker Framework Components



Program Instrumentation (Here: Manual)

Example: Monitoring Operation Executions

Kieker Monitoring & Analysis Framework



Christian-Albrechts-Universität zu Kiel

Application code:

```
public void getOffers() {  
    // EXECUTION to be monitored:  
    catalog.getbook(false);  
}
```

Monitoring probe code (schematic):

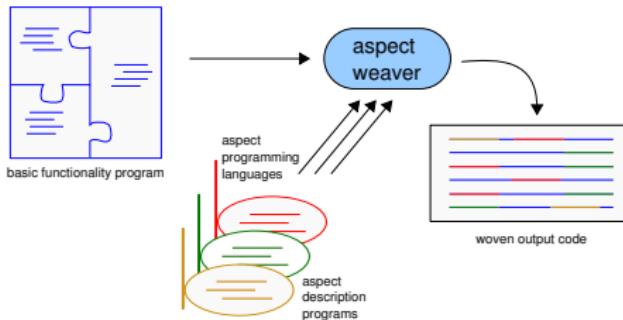
```
// BEFORE execution to be monitored  
if (!isMonitoringEnabled()) {  
    collectDataBefore();  
}
```

```
// AFTER execution to be monitored  
if (!isMonitoringEnabled()) {  
    collectDataAfter();  
    writeMonitoringData();  
}
```

Instrumentation — Getting the *monitoring probe* into the *code*

- ① Manual instrumentation
- ② Aspect-oriented programming (AOP), middleware interception, ...

AOP-Based Instrumentation by Annotation



```
11  
12     @OperationExecutionMonitoringProbe  
13     public void getOffers() {  
14         catalog.getBook(false);  
15     }  
16 }
```

Annotation-based (AOP) instrumentation for monitoring trace information

Listing 1: META-INF/aop.xml

```
<!DOCTYPE aspectj PUBLIC "-//AspectJ//DTD//EN" "http://www.aspectj.org←
    /dtd/aspectj_1_5_0.dtd">
<aspectj>
    <weaver options="">
        <include within="*"/>
    </weaver>
    <aspects>
        <aspect name="kieker.monitoring.probe.aspectj.operationExecution.←
            OperationExecutionAspectFull"/>
    </aspects>
</aspectj>
```

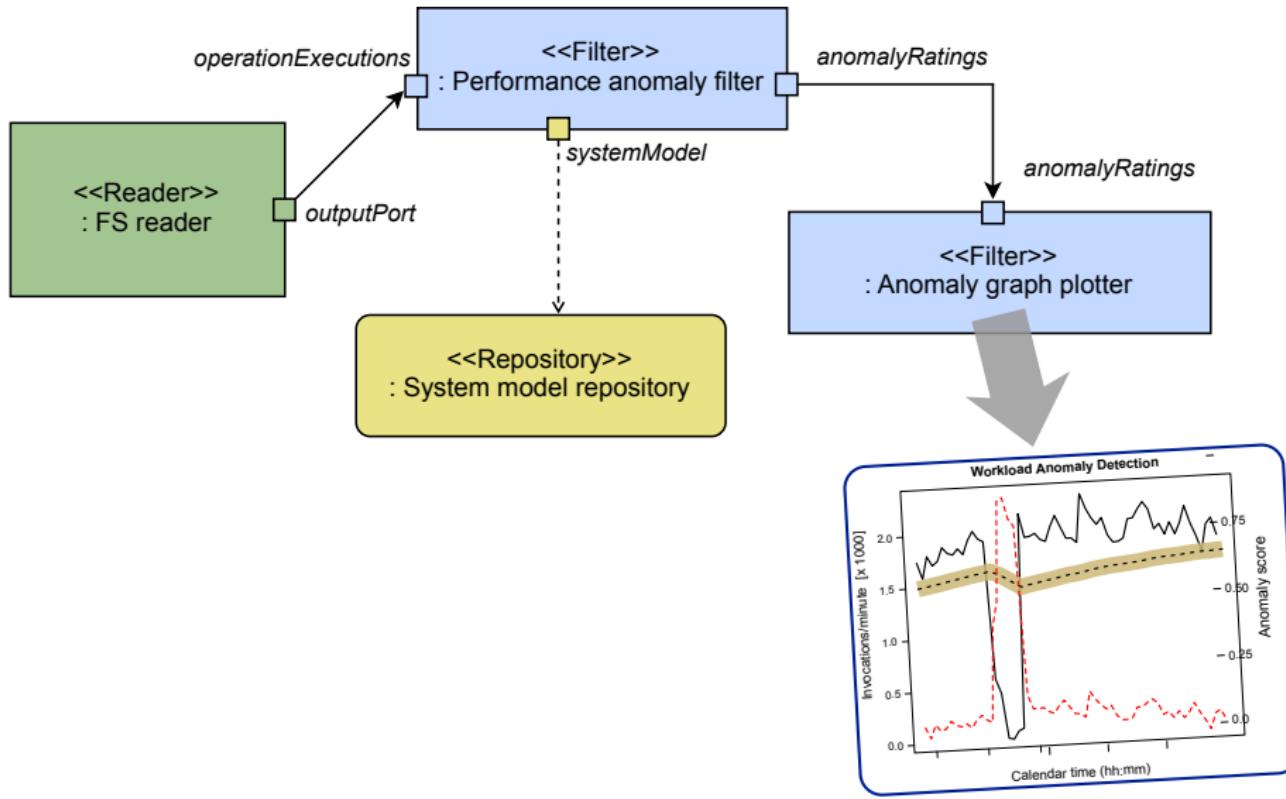
Start the monitored application:

```
$ java -javaagent:lib/kieker-1.8_aspectj.jar BookstoreStarter
```

Analysis Configuration with Kieker

Example Pipe-and-Filter Configuration

Kieker Monitoring & Analysis Framework



Textual Analysis Configuration



```
@Plugin(outputPorts = {  
    @OutputPort(name = "eventCount", eventTypes = { Long.class })})  
public final class CountingFilter extends AbstractFilterPlugin {  
  
    ...  
  
    @InputPort(name = "inputEvents", eventTypes = { Object.class })  
    public final void inputEvent(final Object event) {  
        final Long count = this.counter.incrementAndGet();  
  
        super.deliver("eventCount", count);  
    }  
}
```

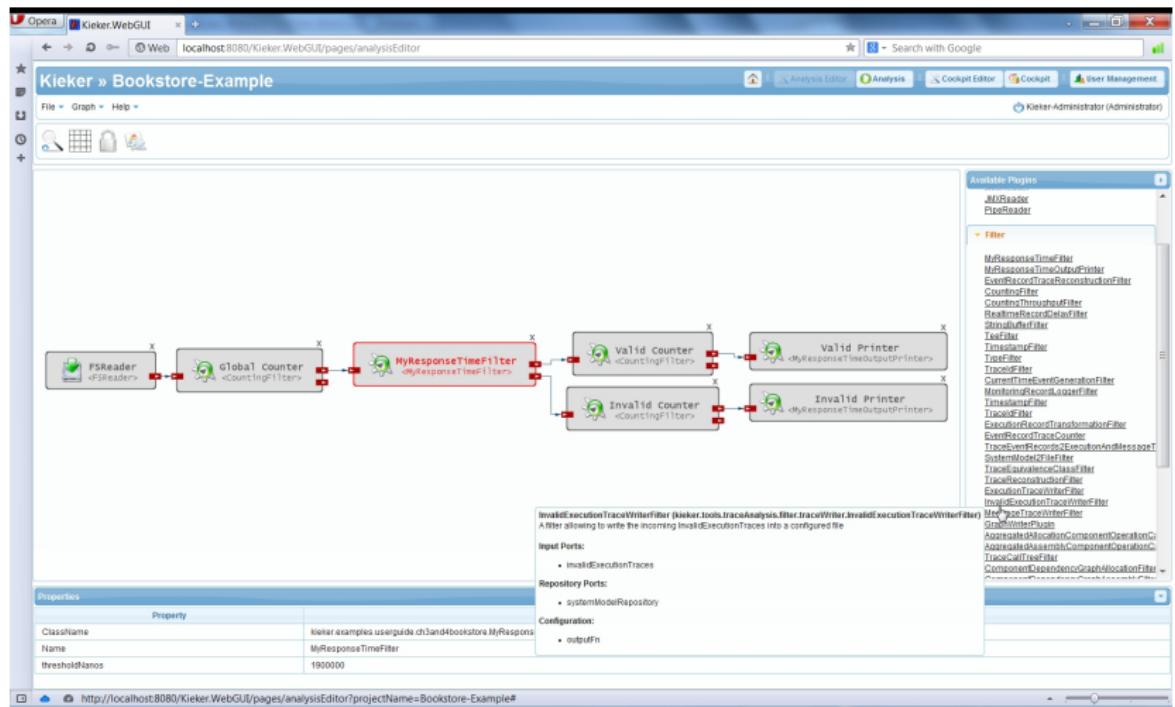
Graphical Analysis Configuration

Web User Interface [Ehmke 2013]

Kieker Monitoring & Analysis Framework



Christian-Albrechts-Universität zu Kiel



<http://localhost:8080/Kieker.WebGUI/pages/analysisEditor?projectId=Bookstore-Example#>

Web Control Center

Cockpit [Ehmke 2013]

Kieker Monitoring & Analysis Framework

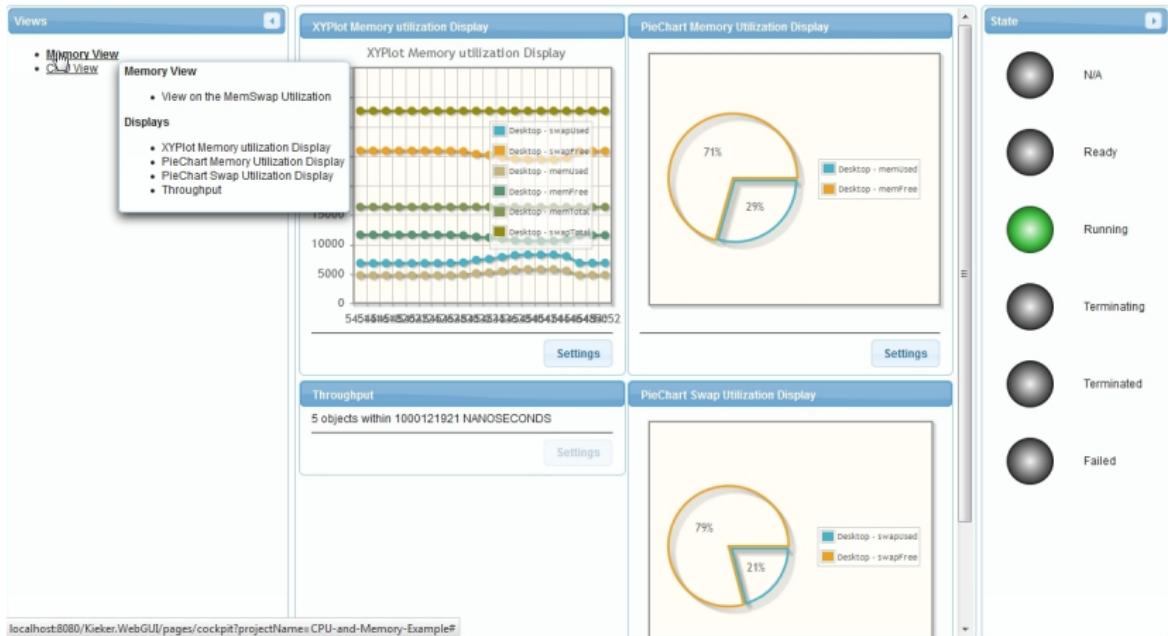


Christian-Albrechts-Universität zu Kiel

Kieker » CPU-and-Memory-Example

File Help

admin (Administrator)

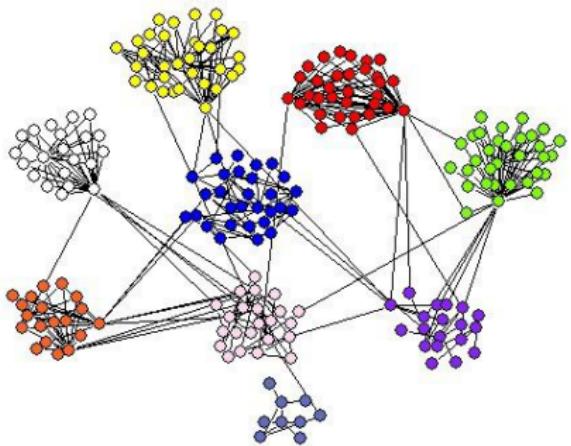


localhost:8080/Kieker/WebGUI/pages/cockpit?projectName=CPU-and-Memory-Example#

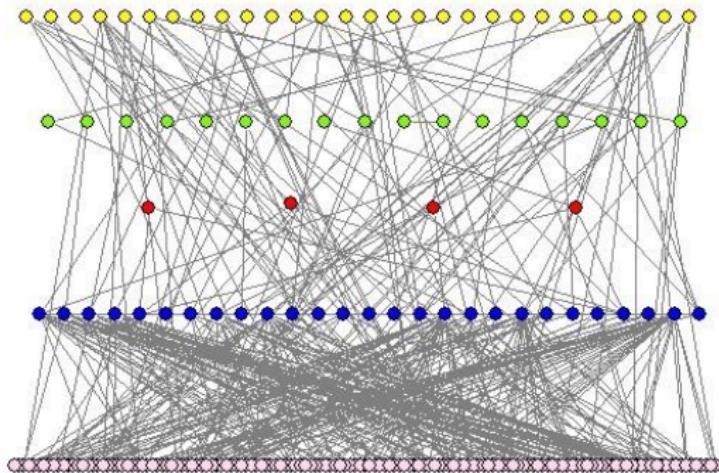
What others are doing with Kieker

Analysis of Calling Networks [Zheng et al. 2011]

Kieker Monitoring & Analysis Framework



Community structure



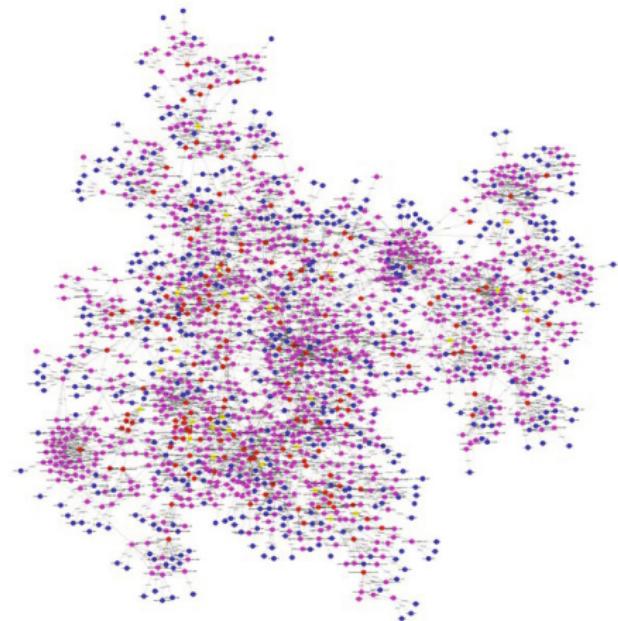
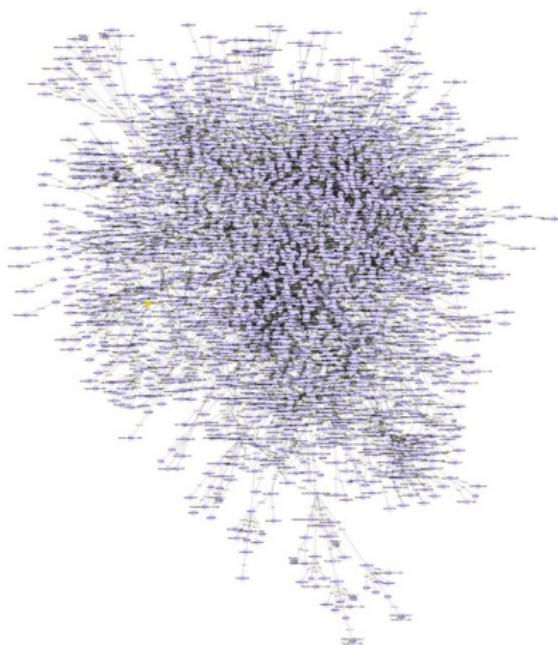
Layered structure (hierarchy)

Xi'an Jiaotong University, Shaanxi [Zheng et al. 2011]

Visualizing “Architecture Warehouses”

Example: Analyzing the JUnit Project [Dąbrowski 2012]

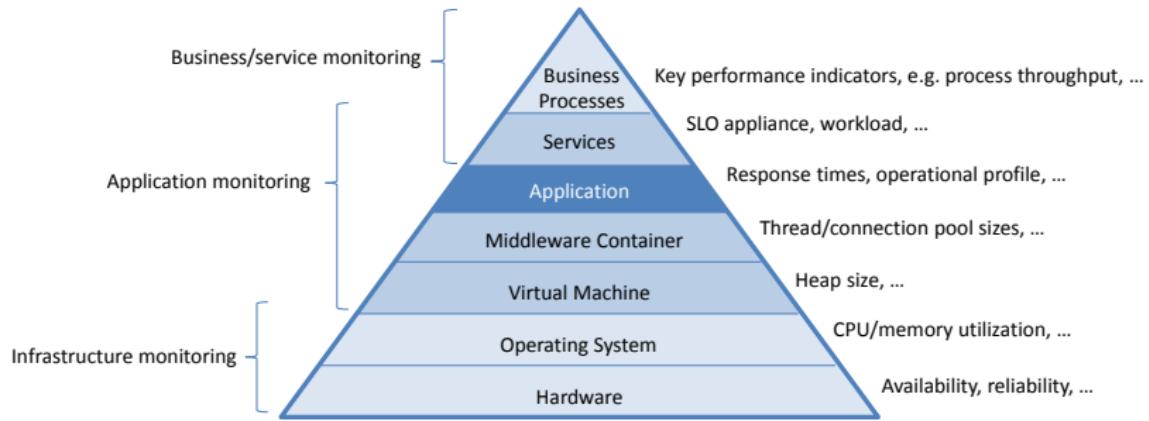
Kieker Monitoring & Analysis Framework



Warsaw University [Dąbrowski 2012]

- 1 Dynamic Software Analysis
- 2 Kieker Monitoring & Analysis Framework
- 3 Application Performance Monitoring
- 4 Reverse Engineering
- 5 Summary and Future Work

Monitoring on different system layers



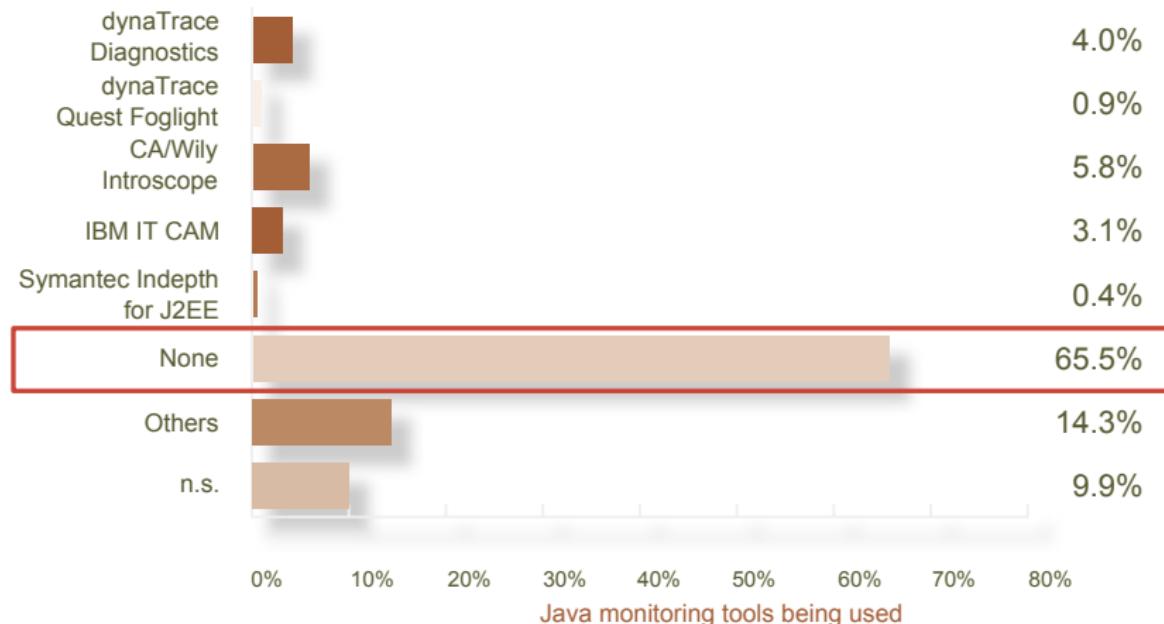
Monitoring practice in the “real world” (based on what we’ve seen)

- Focus on system level (network availability, resource utilization) or business level (key performance indicators)
- No systematic instrumentation on application level
- Monitoring as an “afterthought”: probes are only added when problems occurred.

Application-Level Monitoring in Practice . . . !?

— Among Java Professionals —

Application Performance Monitoring



“Java monitoring largely unknown.”
[codecentric GmbH 2009]

Scaling Facebook to 500 Million Users and Beyond

“Making lots of small changes and watching what happens only works if you’re actually able to watch what happens. At Facebook we **collect an enormous amount of data** — any particular server exports **tens or hundreds of metrics** that can be graphed. This isn’t just system level things like CPU and memory, **it’s also application level statistics** to understand why things are happening.

It’s important that the statistics are from the **real production machines** that are having the problems, when they’re having the problems – **the really interesting things only show up in production**. The stats also have to come from all machines, because a lot of important effects are hidden by averages and only show up in distributions, in particular 95th or 99th percentile.”

Robert Johnson, Facebook Engineering Director

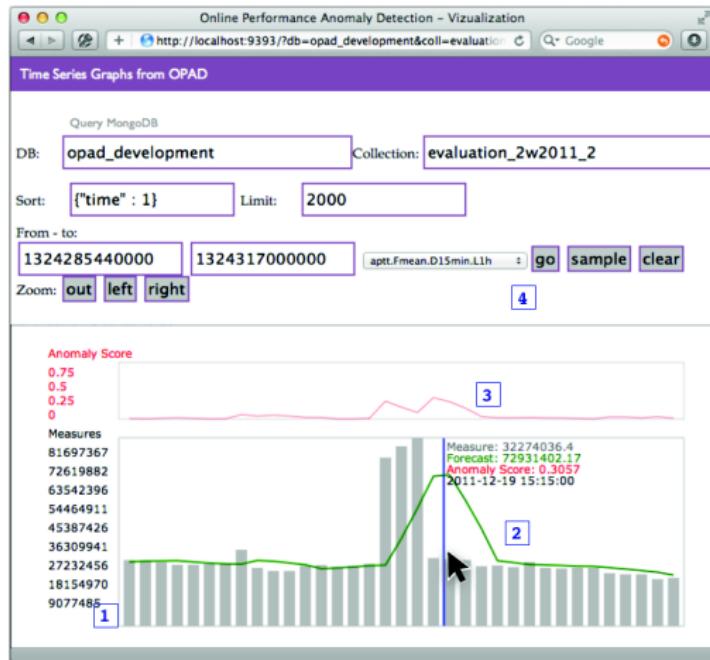
Anomaly Detection + Diagnosis

Kieker @ Xing AG [Bielefeld 2012, Frotscher 2013]

Application Performance Monitoring



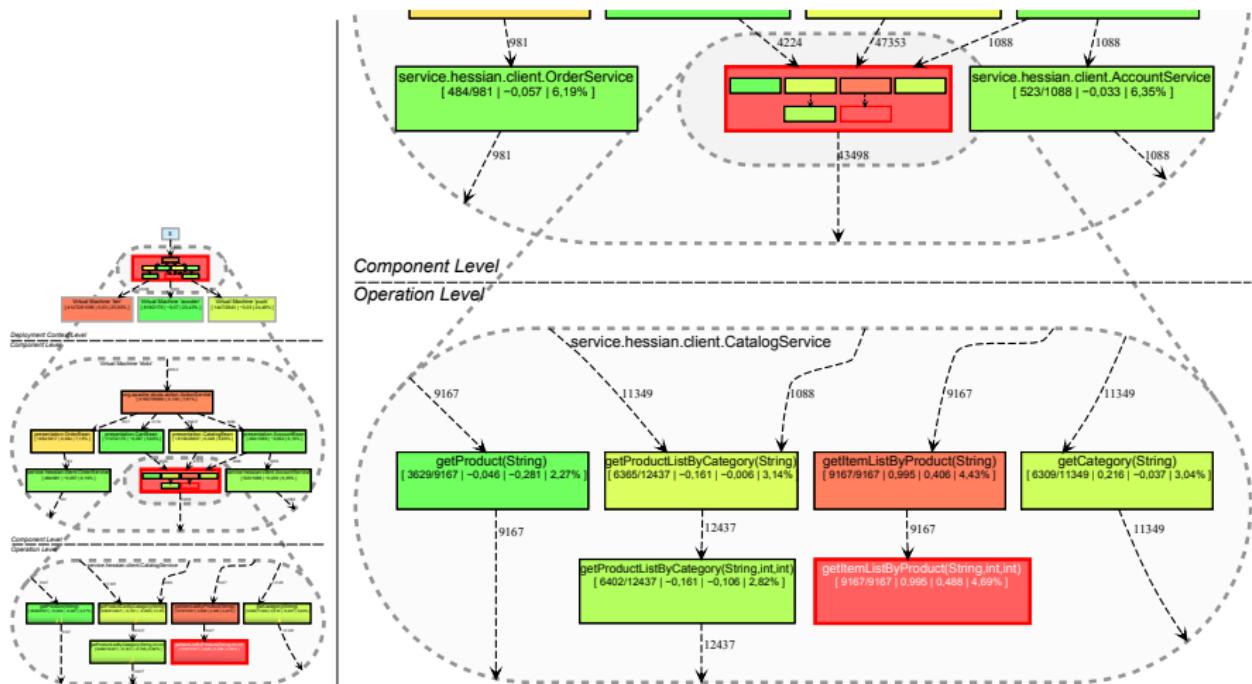
XING



Anomaly Detection + Diagnosis

Previous work [Marwede et al. 2009]

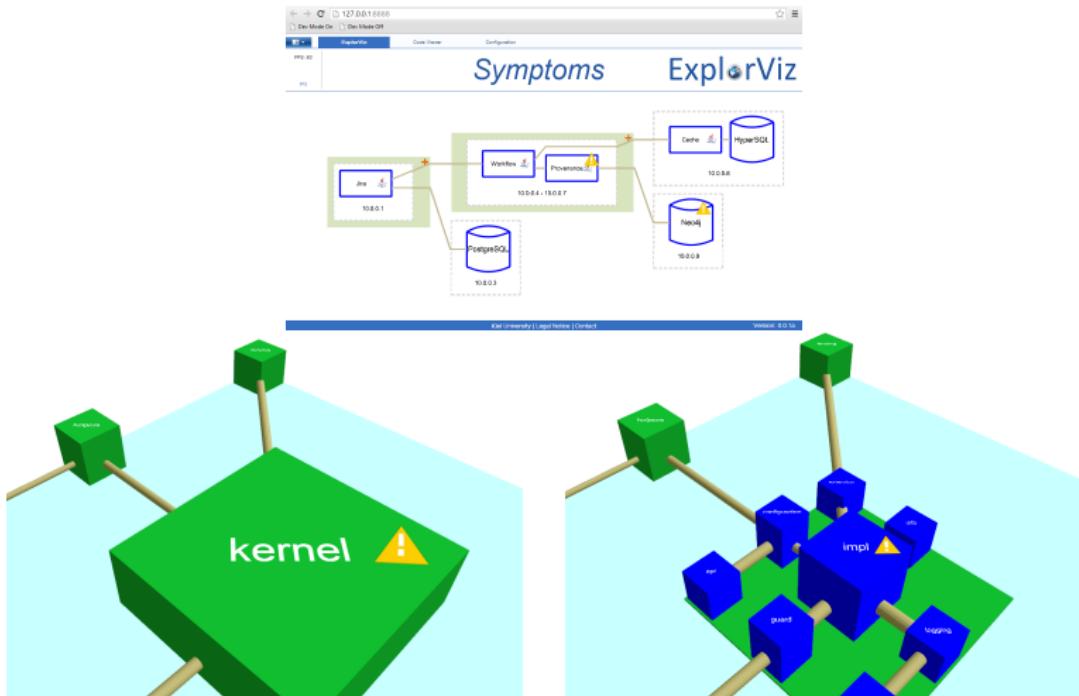
Application Performance Monitoring



Anomaly Detection + Diagnosis

ExplorViz [Fittkau et al. 2013]

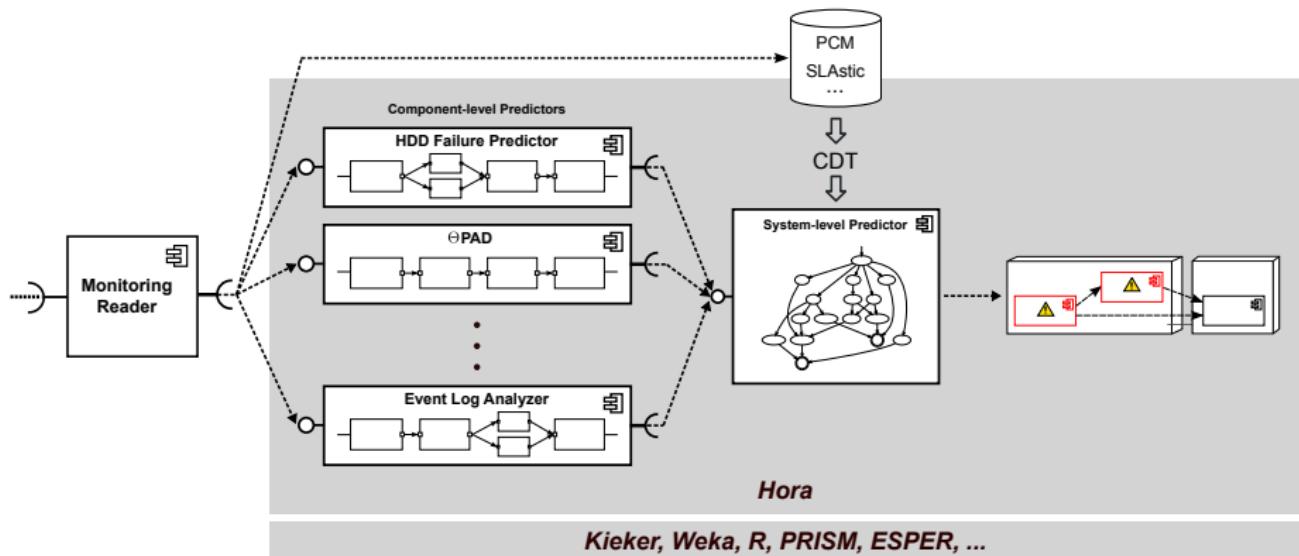
Application Performance Monitoring



Anomaly Detection + Diagnosis

Work at Stuttgart University on online failure prediction [Pitakrat et al. 2014]

Application Performance Monitoring

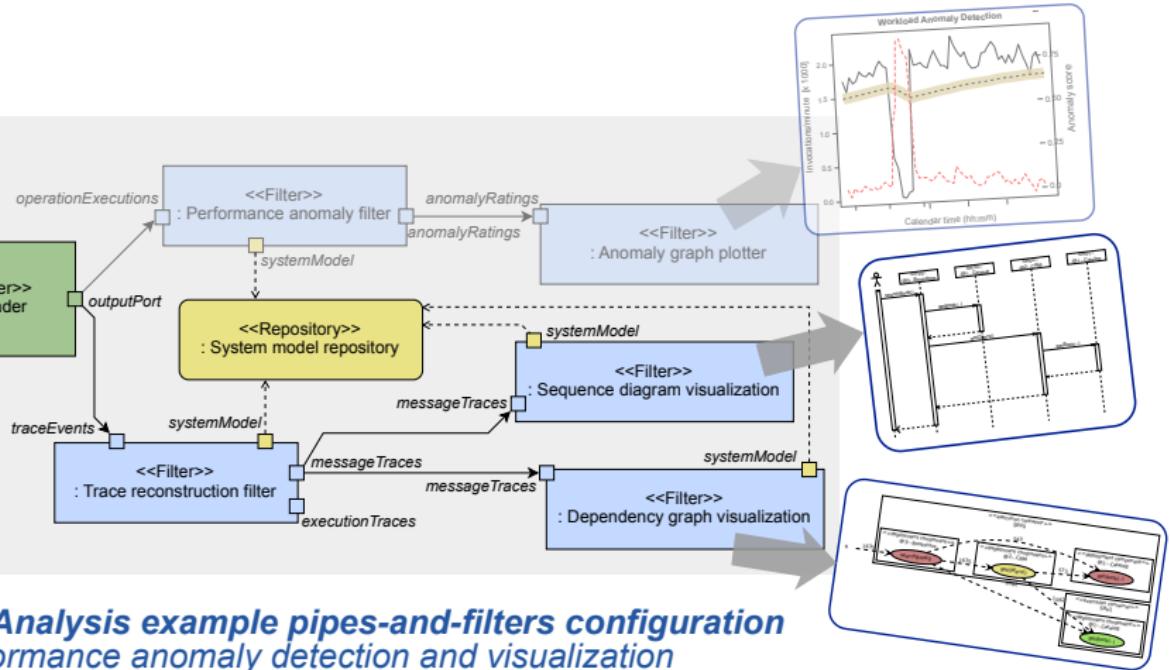


- 1 Dynamic Software Analysis
- 2 Kieker Monitoring & Analysis Framework
- 3 Application Performance Monitoring
- 4 Reverse Engineering
 - Reverse Engineering of Java EE
 - Reverse Engineering of C#
 - Reverse Engineering of Visual Basic 6
 - Reverse Engineering of COBOL
 - Reverse Engineering of C / C++
 - Reverse Engineering of Perl
 - Kieker in Space
- 5 Summary and Future Work

Extending the Pipe-and-Filter Configuration

Here: for Reverse Engineering

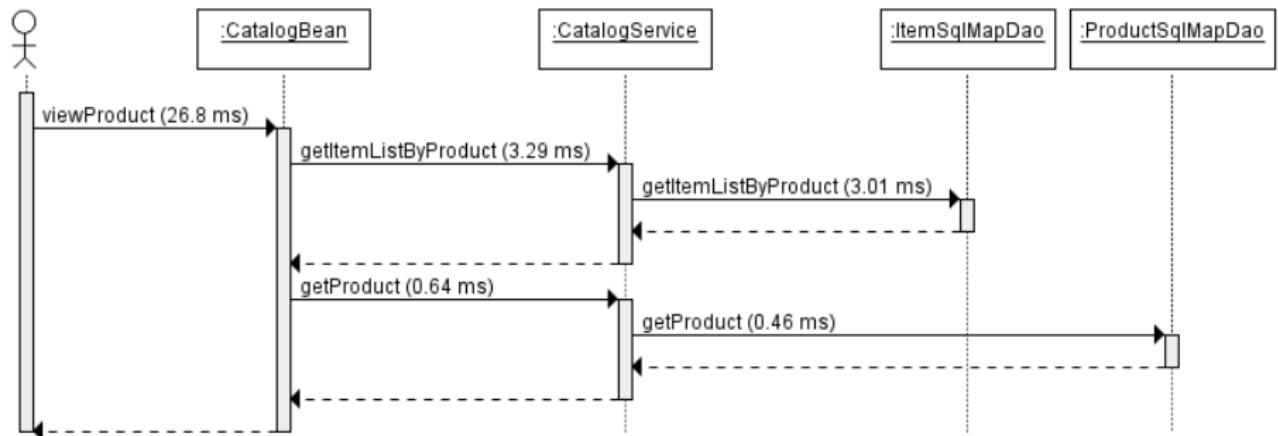
Reverse Engineering



Kieker.Analysis example pipes-and-filters configuration

- Performance anomaly detection and visualization
- Architecture and trace reconstruction/visualization

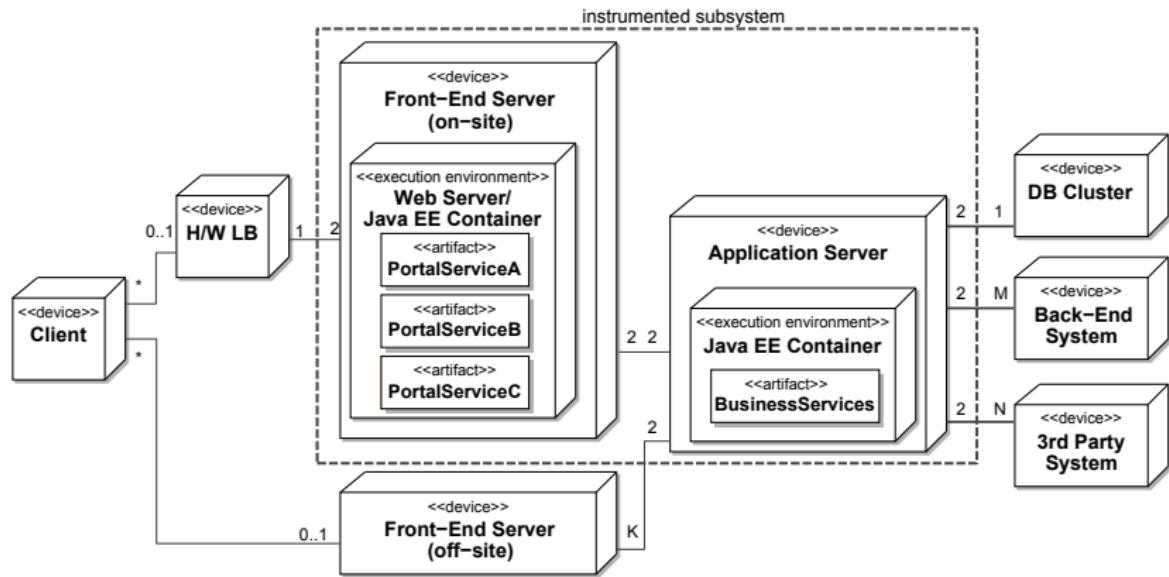
Generated Sequence Diagram



Reverse Engineering of Java EE

Customer portal at EWE TEL GmbH
Reverse Engineering > Reverse Engineering of Java EE

Distributed Enterprise Java System



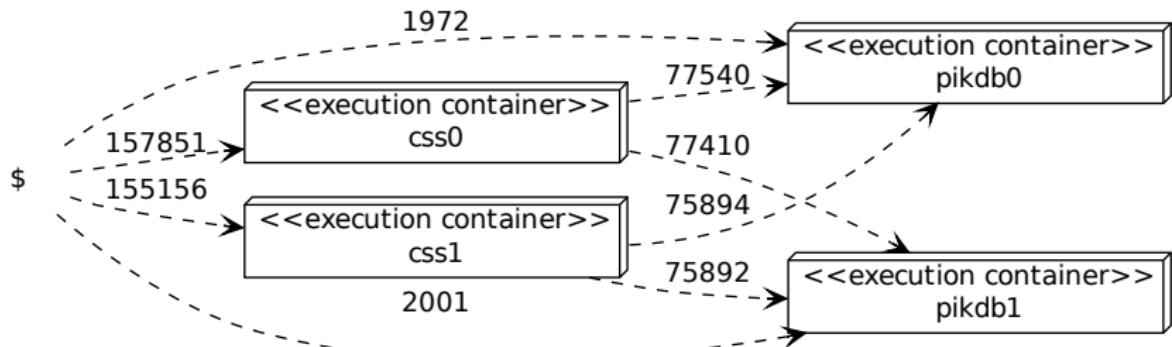
Servlet, Spring and CXF/SOAP probes



Software Architecture Level

Component dependency graph

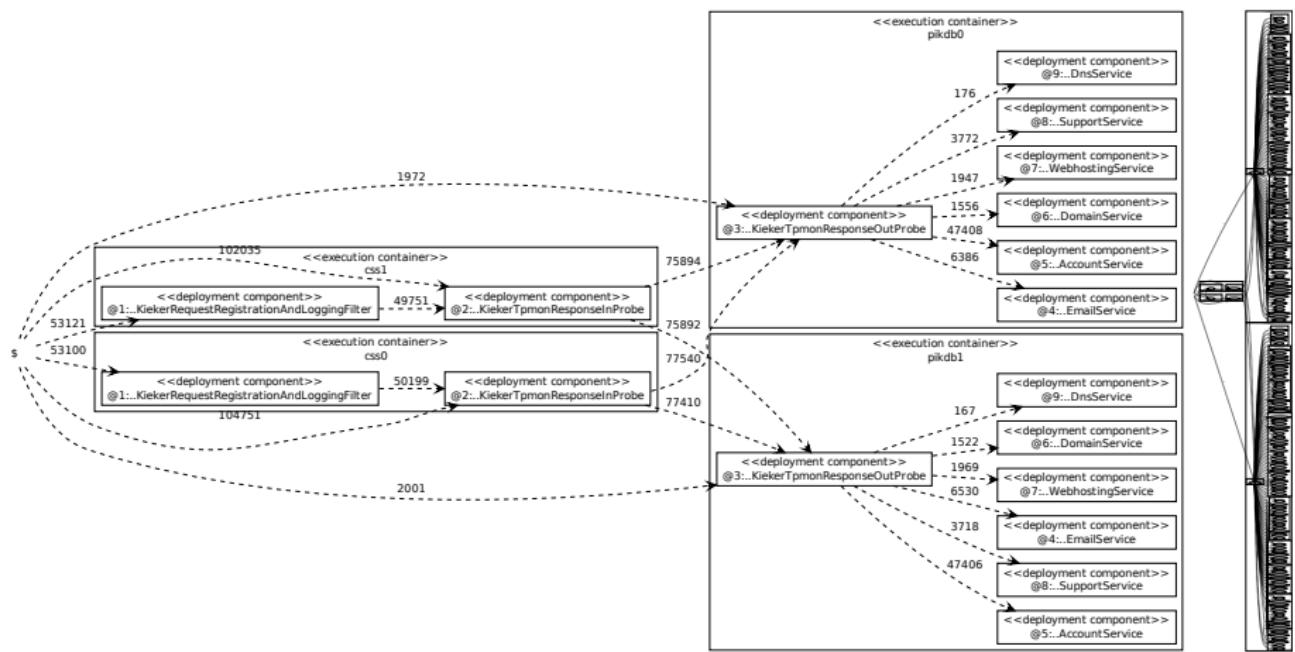
Reverse Engineering > Reverse Engineering of Java EE



System Architecture Level

Component dependency graph

Reverse Engineering > Reverse Engineering of Java EE



Dynamic Analysis for Modernization

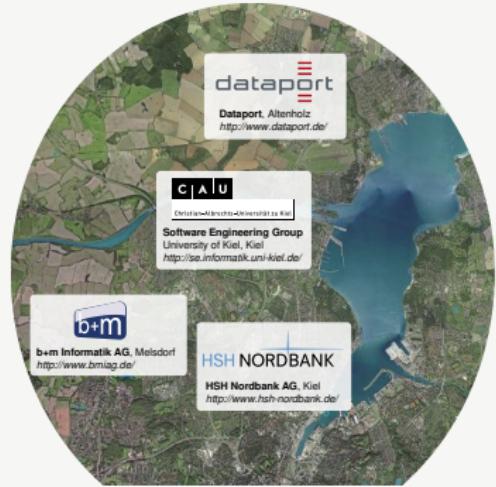
KoSSE Project DynaMod, <http://kosse-sh.de/dynamod>

Reverse Engineering ▷ Reverse Engineering of C#



Motivation: Emergent Architectures

Project Consortium



Bundesministerium
für Bildung
und Forschung

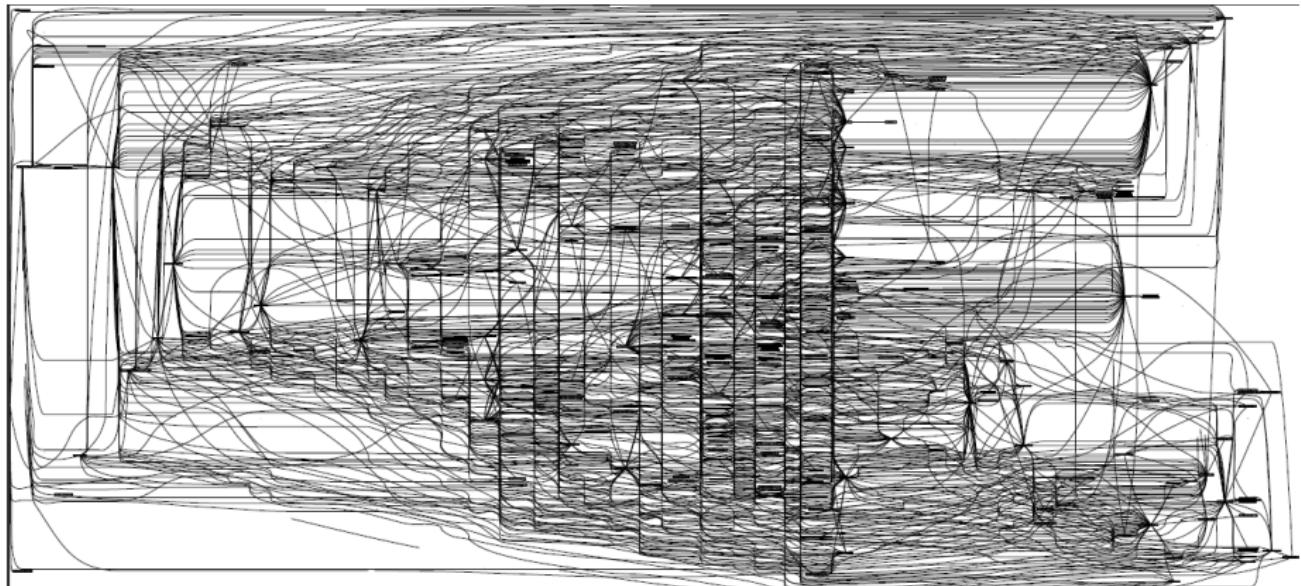
Reverse Engineering of C#

Complete Test Suite of Nordic Analytics [Magendanz 2011]

Reverse Engineering > Reverse Engineering of C#



Christian-Albrechts-Universität zu Kiel



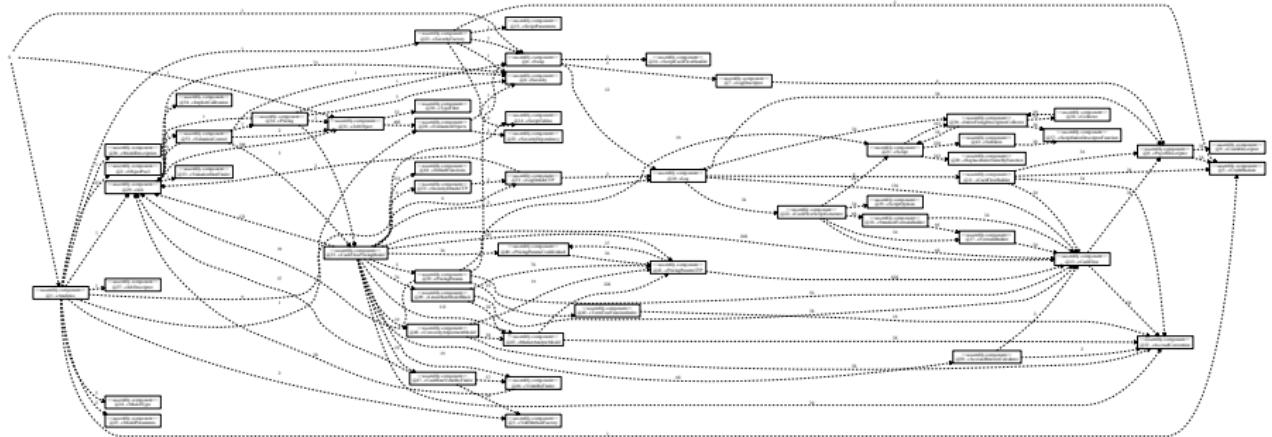
Case study at HSH Nordbank AG.



After Selecting a single Use Case

Reverse Engineering of C#

Reverse Engineering > Reverse Engineering of C#



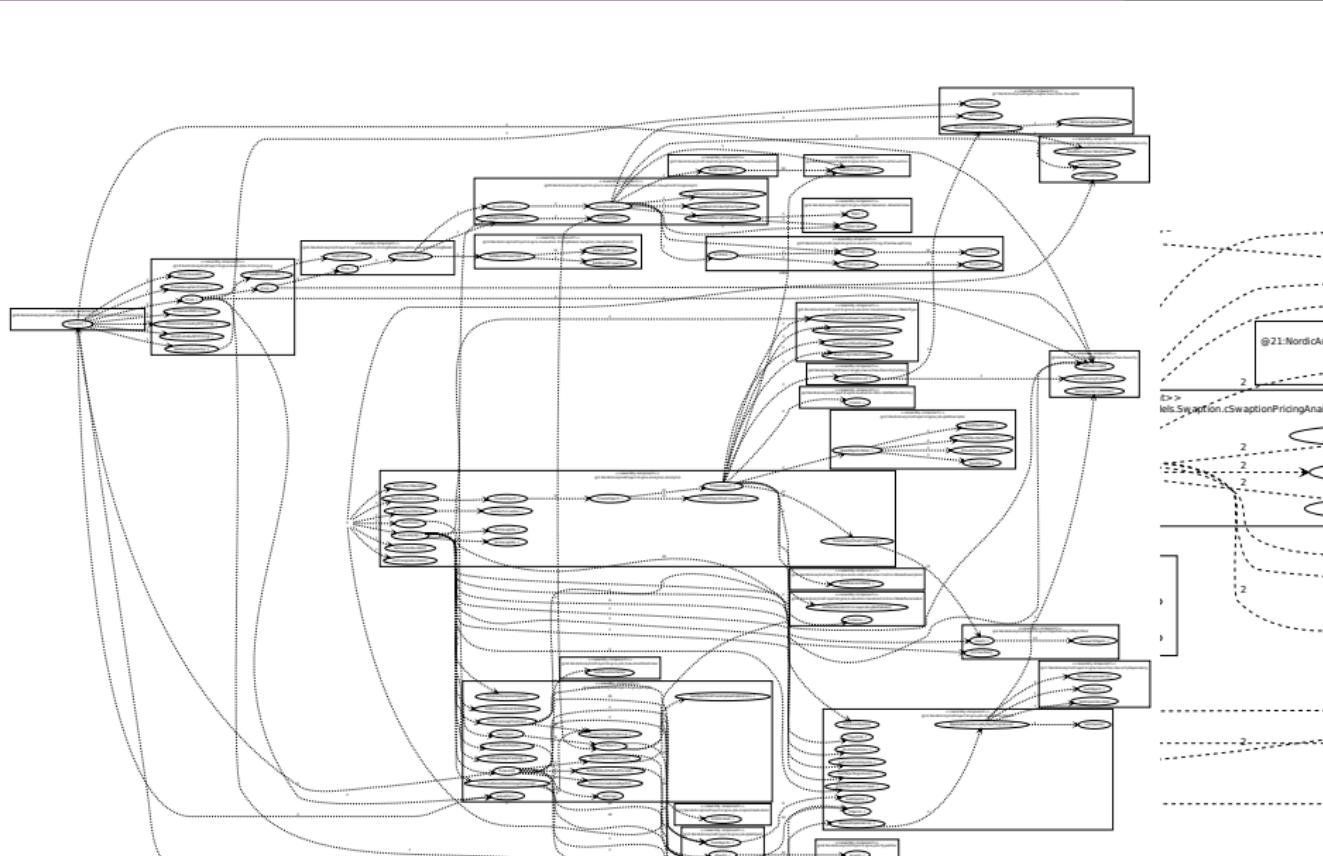
Zoom to the Operation Level

Reverse Engineering of C#

Reverse Engineering > Reverse Engineering of C#



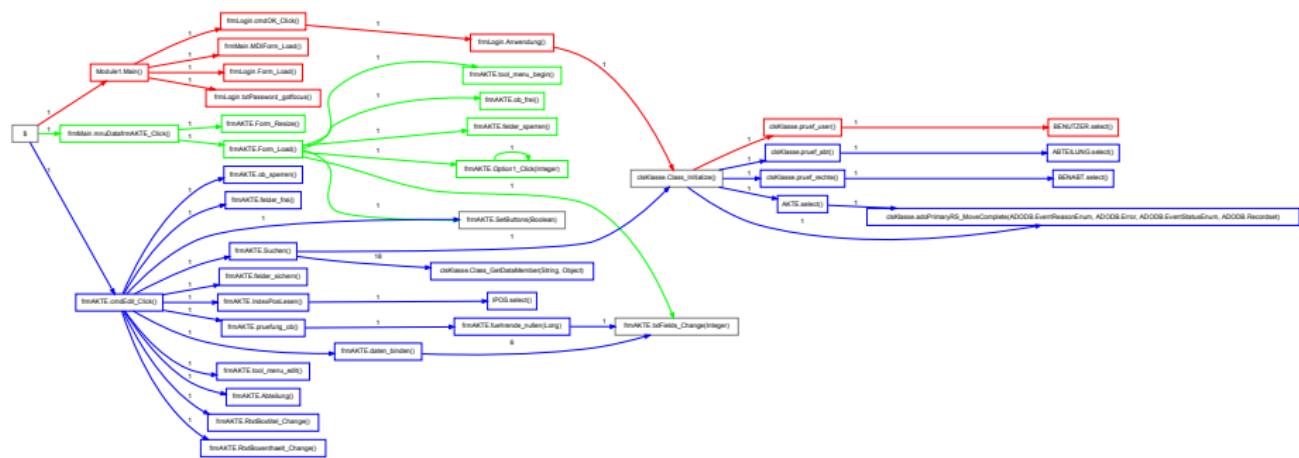
Christian-Albrechts-Universität zu Kiel



Reverse Engineering of Visual Basic 6

Analysis of Specific Traces

Reverse Engineering > Reverse Engineering of Visual Basic 6

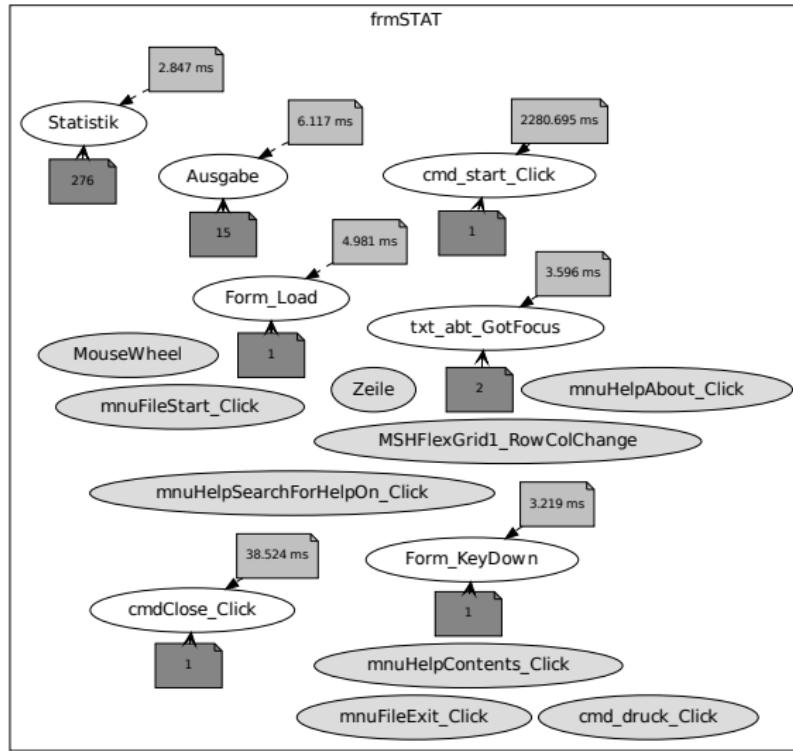


Case study at Dataport

Identification of unused Functions

Reverse Engineering of Visual Basic 6

Reverse Engineering > Reverse Engineering of Visual Basic 6



Reverse Engineering of COBOL

With consideration of non-instrumentable modules [Knoche et al. 2012]

Reverse Engineering > Reverse Engineering of COBOL

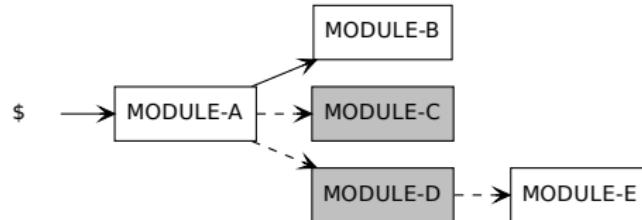
```
IDENTIFICATION DIVISION.  
PROGRAM-ID. MODULE-X.  
  
PROCEDURE DIVISION.  
  
    CALL "MODULE-Y".  
    GOBACK.
```

Literal block

Before-Call Injection Point
Called Module: "MODULE-Y"

Literal block

AOP-based COBOL instrumentation

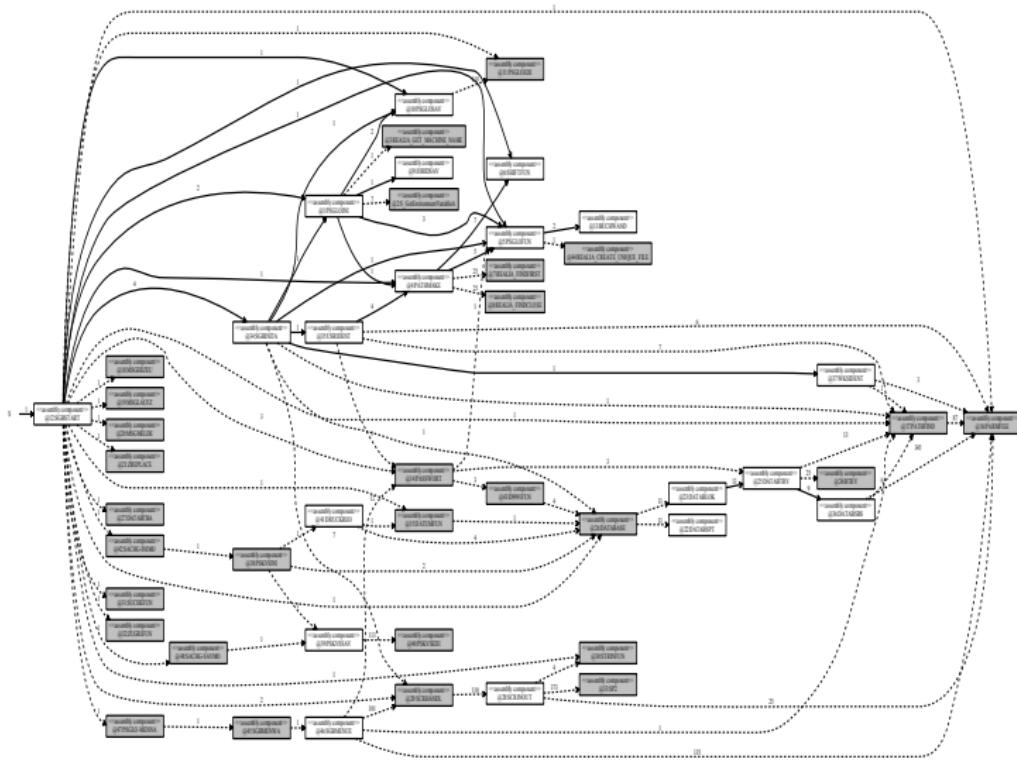


Module-level call dependency graph with assumed dependencies

Analysis of a PC COBOL System

[Richter 2012]

Reverse Engineering > Reverse Engineering of COBOL



C++ Digital Signal Processing

Kiel Real-time Audio Toolkit (KiRAT)

Reverse Engineering > Reverse Engineering of C / C++



Christian-Albrechts-Universität zu Kiel

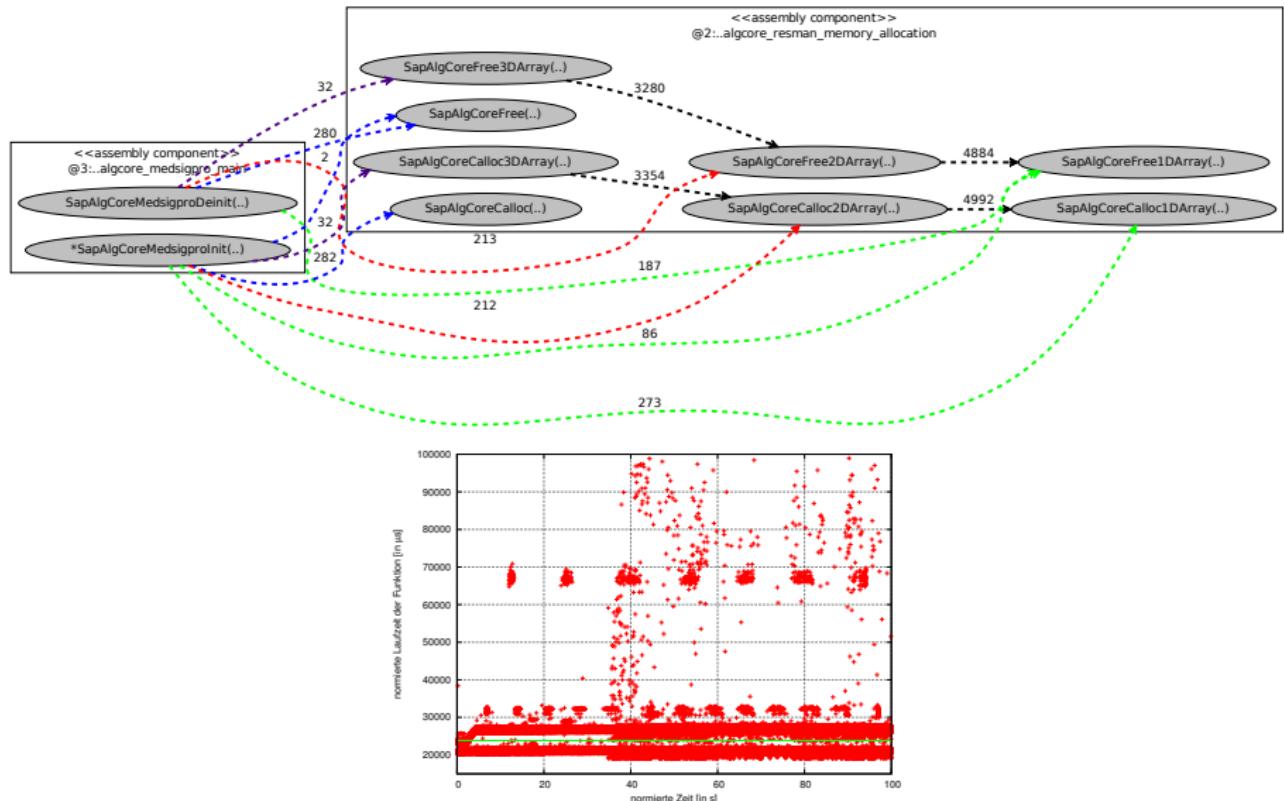


Source: <http://www.dss.tf.uni-kiel.de/en/research/kirat>

Reverse Engineering of C++

Analysis of the algorithmic kernel [Mahmens 2014]

Reverse Engineering > Reverse Engineering of C / C++



Reverse Engineering of Perl

Visualizing Dependencies in EPrints

Reverse Engineering > Reverse Engineering of Perl



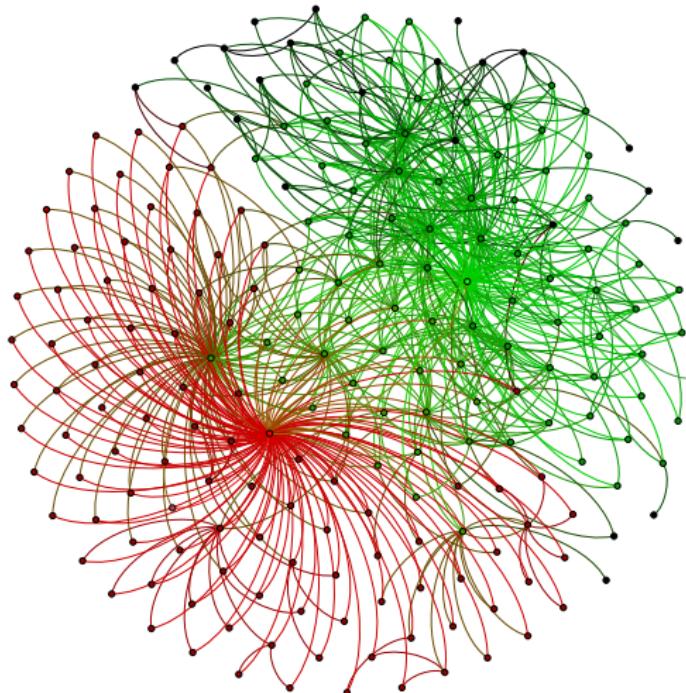
First shot with full instrumentation:



Visualizing Dependencies in EPrints

Customized visualization with Gephi [Wechselberg 2013]

Reverse Engineering > Reverse Engineering of Perl



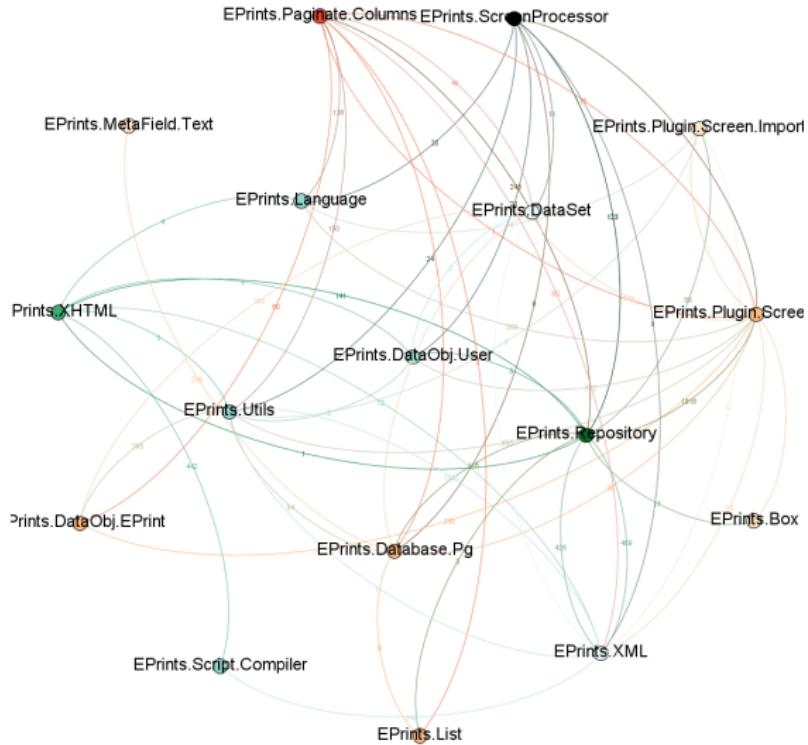
Red: EPrints.PluginFactory.

Green: EPrints.Repository.

Visualizing Dependencies in EPrints

Refinement with adaptive monitoring [Jensen 2013]

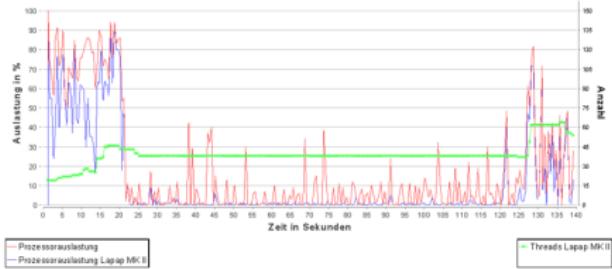
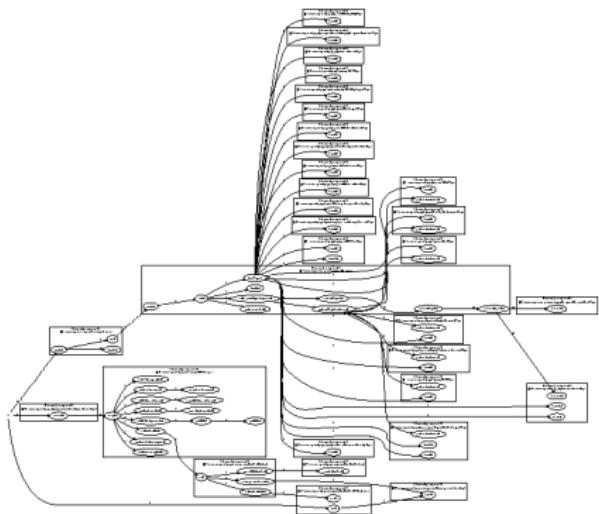
Reverse Engineering ▷ Reverse Engineering of Perl



Kieker in Space

Some analysis results [Harms 2013]

Reverse Engineering > Kieker in Space



Summary

- Monitoring for dynamic analysis may provide valuable information about the **real** behavior of a software system.
 - Combined with static analysis.
- Monitoring Data may be employed for various purposes, for instance
 - Fault localization and diagnosis
 - Architecture discovery
 - Capacity management (SLAastic project, PhD Thesis André van Hoorn)
- Flexibility is required, particularly for architecture discovery

Future Work

- Extending the Web-based configuration interface & cockpit
- Model-driven instrumentation & analysis
- Application to workflow monitoring, parallelization and Enterprise Architecture Management

References

- T. C. Bielefeld. Online performance anomaly detection for large-scale software systems. Diploma thesis, University of Kiel, Germany, Mar. 2012. URL <http://eprints.uni-kiel.de/15488/>.
- codecentric GmbH. Performance survey 2008. http://www.codecentric.de/export/sites/www/_resources/pdf/performance-survey-2008-web.pdf, Mar. 2009.
- B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, and R. Koschke. A systematic survey of program comprehension through dynamic analysis. *IEEE Transactions on Software Engineering*, 35(5):684–702, 2009. ISSN 0098-5589. doi: 10.1109/TSE.2009.28.
- R. Dąbrowski. On architecture warehouses and software intelligence. In T.-h. Kim, Y.-h. Lee, and W.-c. Fang, editors, *Proceedings of the 4th International Mega-Conference on Future Generation Information Technology (FGIT 2012)*, volume 7709 of *Lecture Notes in Computer Science*, pages 251–262. Springer, 2012. ISBN 978-3-642-35584-4. doi: 10.1007/978-3-642-35585-1_35.
- N. C. Ehrmke. Everything in sight: Kieker's WebGUI in action (tutorial). In S. Becker, W. Hasselbring, A. van Hoorn, and R. Reussner, editors, *Proceedings of the Symposium on Software Performance: Joint Kieker/Palladio Days (KP'DAYS '13)*, volume 1083 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013. URL <http://eprints.uni-kiel.de/22528/>.
- F. Fittkau, J. Waller, C. Wulf, and W. Hasselbring. Live trace visualization for comprehending large software landscapes: The explorviz approach. In *1st IEEE International Working Conference on Software Visualization (ViSSOFT 2013)*, September 2013. URL <http://eprints.uni-kiel.de/21840/>.
- T. Frotscher. Architecture-based multivariate anomaly detection for software systems. Master's thesis, CAU, AG Software Engineering, Oct. 2013. URL <http://eprints.uni-kiel.de/21346/>.
- B. Harms. Reverse-Engineering und Analyse einer Plug-in-basierten Java-Anwendung. Master's thesis, Kiel University, Nov. 2013. URL <http://eprints.uni-kiel.de/23733/>.
- T. S. Jensen. Adaptives Monitoring von Perl-Applikationen. Master's thesis, Institut für Informatik, Sept. 2013. URL <http://eprints.uni-kiel.de/22398/>.
- H. Knoche, A. van Hoorn, W. Goerigk, and W. Hasselbring. Automated source-level instrumentation for dynamic dependency analysis of COBOL systems. In *Proceedings of the 14. Workshop Software-Reengineering (WSR '12)*, pages 33–34, May 2012. URL <http://eprints.uni-kiel.de/14417/>.
- F. Magendanz. Dynamic analysis of .NET applications for architecture-based model extraction and test generation, Oct. 2011. URL <http://eprints.uni-kiel.de/15489/>.
- S. Mahmens. Architektur-Rekonstruktion mit Kieker durch AOP-basierte Instrumentierung einer C++-Anwendung. Master's thesis, Institut für Informatik, Apr. 2014. URL <http://eprints.uni-kiel.de/24349/>.
- N. S. Marwede, M. Rohr, A. van Hoorn, and W. Hasselbring. Automatic failure diagnosis support in distributed large-scale software systems based on timing behavior anomaly correlation. In *Proceedings of the 13th European Conference on Software Maintenance and Reengineering (CSMR '09)*, pages 47–57. IEEE, Mar. 2009. ISBN 978-0-7695-3589-0. doi: 10.1109/CSMR.2009.15. URL <http://eprints.uni-kiel.de/14469/>.

References (cont'd)

References

- T. Pitakrat, A. van Hoorn, and L. Grunske. Increasing dependability of component-based software systems by online failure prediction. In *Proceedings of the 10th European Dependable Computing Conference (EDCC '14)*, 2014. To appear.
- B. Richter. Dynamische Analyse von COBOL-Systemarchitekturen zum modellbasierten Testen. Diploma thesis, University of Kiel, Germany, Aug. 2012. URL <http://eprints.uni-kiel.de/15489/>.
- M. Rohr, A. van Hoorn, J. Matevska, N. Sommer, L. Stoever, S. Giesecke, and W. Hasselbring. Kieker: Continuous monitoring and on demand visualization of Java software behavior. In C. Pahl, editor, *Proceedings of the IASTED International Conference on Software Engineering 2008 (SE'08)*, pages 80–85, Feb. 2008. ISBN 978-0-88986-715-4. URL <http://eprints.uni-kiel.de/14496/>.
- A. van Hoorn, M. Rohr, and W. Hasselbring. Engineering and continuously operating self-adaptive software systems: Required design decisions. In G. Engels, R. Reussner, C. Momma, and S. Sauer, editors, *Design for Future – Langlebige Softwaresysteme*, volume 537 of *Workshop Proceedings*, pages 52–63. CEUR, Oct. 2009a. URL <http://eprints.uni-kiel.de/14652/>.
- A. van Hoorn, M. Rohr, W. Hasselbring, J. Waller, J. Ehlers, S. Frey, and D. Kieselhorst. Continuous monitoring of software services: Design and application of the Kieker framework. Technical Report TR-0921, Department of Computer Science, University of Kiel, Germany, Nov. 2009b. URL <http://eprints.uni-kiel.de/14459/>.
- A. van Hoorn, J. Waller, and W. Hasselbring. Kieker: A framework for application performance monitoring and dynamic software analysis. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering (ICPE 2012)*, pages 247–248. ACM, Apr. 2012. ISBN 978-1-4503-1202-8. URL <http://eprints.uni-kiel.de/14418/>.
- J. Waller and W. Hasselbring. A comparison of the influence of different multi-core processors on the runtime overhead for application-level monitoring. In V. Pankratius and M. Philippson, editors, *Proceedings of the International Conference on Multicore Software Engineering, Performance, and Tools (MSEPT 2012)*, volume 7303 of *Lecture Notes in Computer Science*, pages 42–53. Springer Berlin / Heidelberg, June 2012. ISBN 978-3-642-31201-4. doi: 10.1007/978-3-642-31202-1_5. URL <http://eprints.uni-kiel.de/15425/>.
- N. B. Wechselberg. Monitoring von Perl-basierten Webanwendungen mittels Kieker. Bachelorarbeit, Kiel University, April 2013. URL <http://eprints.uni-kiel.de/21141/>.
- Q. Zheng, Z. Ou, L. Liu, and T. Liu. A novel method on software structure evaluation. In *Proceedings of the 2nd IEEE International Conference on Software Engineering and Service (IEEE ICSESS 2011)*, pages 251–254. IEEE, July 2011. doi: 10.1109/ICSESS.2011.5982301.