

# *Low Costs – High Speed*

Kostengünstige Performance Engineering Lösungen  
(Java Forum Stuttgart 2009)

M.Sc. Stefan Siegl

NovaTec GmbH

Business Unit Leitung „Application Performance Engineering“

NovaTec  
Ingenieure für neue Informationstechnologien GmbH

Dieselstr. 18/1 D-70771 Leinfelden-Echterdingen Telefon: +49(0)700 / 5280 5280 Fax: +49(0)700 / 5280 5290  
E-Mail: [info@novatec-gmbh.de](mailto:info@novatec-gmbh.de) Internet: [www.novatec-gmbh.de](http://www.novatec-gmbh.de)



CONSULTING  
SOLUTIONS

## Stefan Siegl

- 29 Jahre alt
- Master of Science (Software Technology)
- Seit 5 Jahren als IT Consultant bei der NovaTec GmbH im Bereich Softwarearchitektur.
- Seit 2,5 Jahren Verantwortlicher des Bereichs „Application Performance Engineering“



- Motivation
- „Projektkiller“ Performance
- Software Performance Engineering
- Application Monitoring Werkzeuge
  - Auswahlkriterien: Monitoring Werkzeug
  - Werkzeugvorstellung
    - Manuelle Integration in den Quellcode
    - Transparente Integration
  - Abgrenzung zu den Marktführern
- Fazit

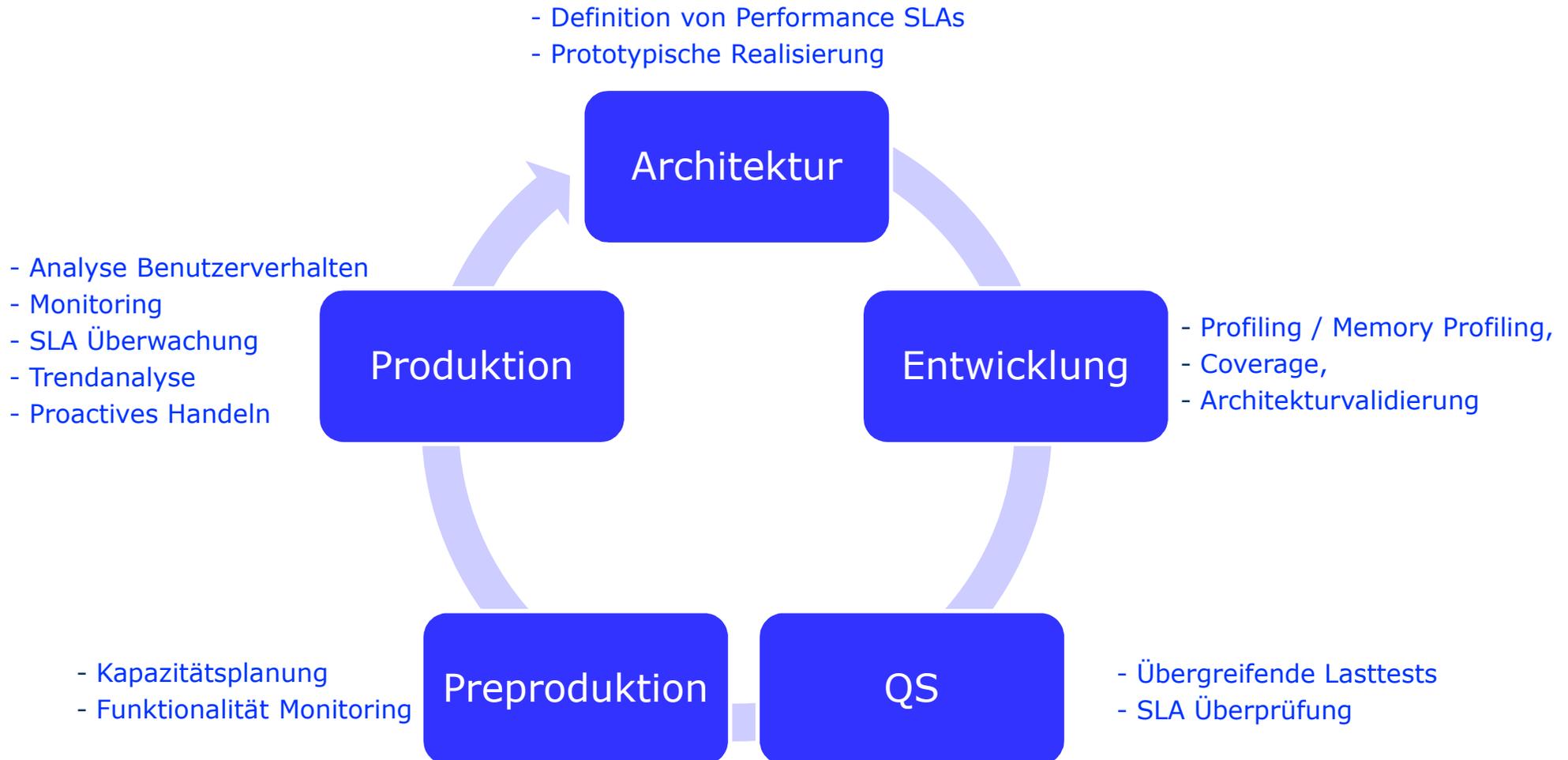
Qualitätsplanung und Sicherstellung ist ein essentieller Bestandteil jeder Ingenieursdisziplin



*85% aller Projekte haben Performanceprobleme  
(Forrester 2004)*

- Ursachen für Performanceprobleme
  - Kein Management von Performance
  - Missverständnis von Best Practices (“make it run, make it fast”)
  - Funktionalität vor Performance
  - Performance als “Add-on” nach der Fertigstellung der Applikation
- Ausweg
  - Risikogetriebenes, proaktives Management von Performance

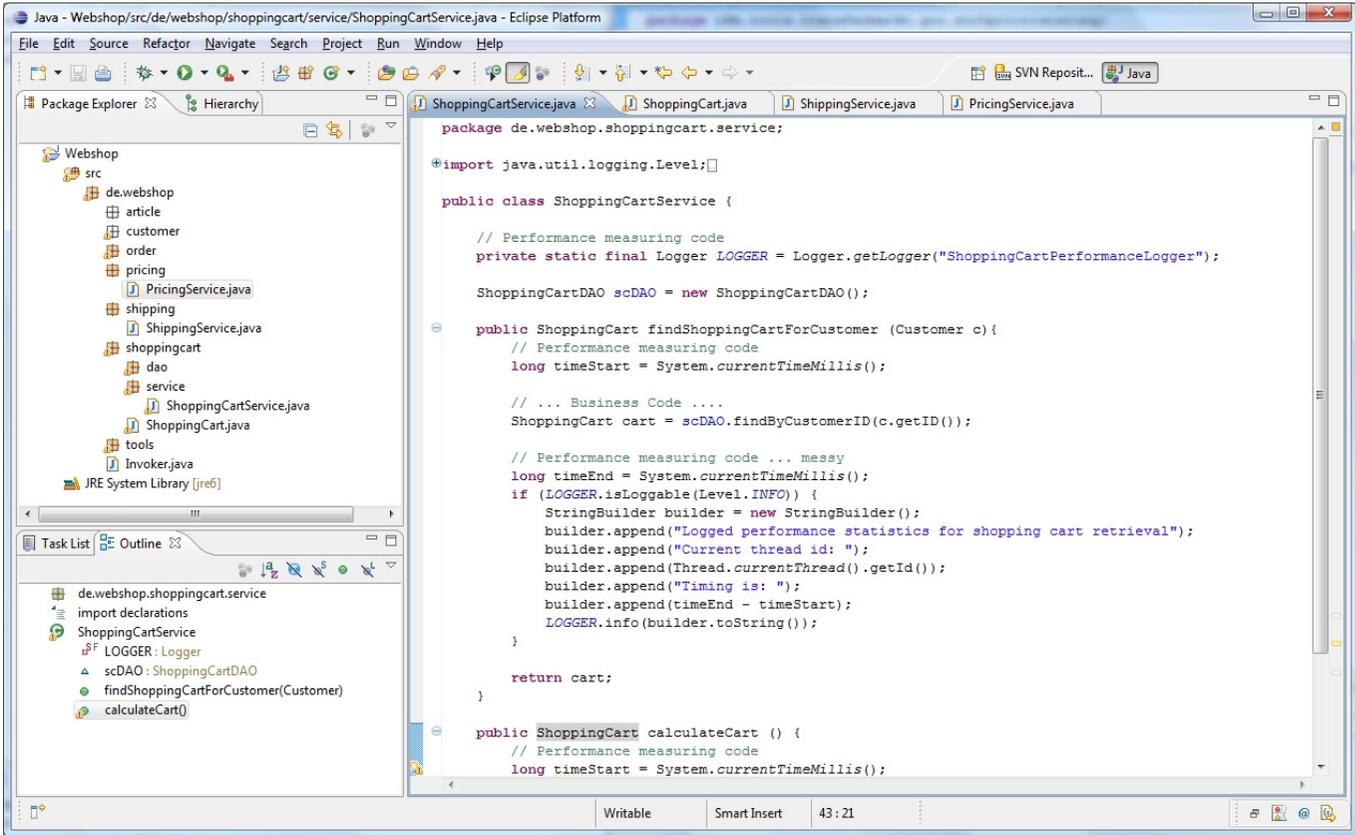
Systematischer, kostengetriebener Ansatz über alle Projektphasen



- Integration in die Anwendung
  - Möglichst transparente Integration ohne Anpassung des Anwendungscodes
  - Optimal: runtime bytecode modification, AOP
- Overhead
  - Möglichst gering, sonst „Äpfel mit Birnen Vergleich“
  - Orientiert sich am Umfang des Monitorings
  - 3-stufiger Ansatz (Agent-Speicherung-GUI) liefert typischerweise geringsten Overhead
  - Optimal : <5% (Produktion)
- Konfiguration und Konfigurationsanpassungen
  - Möglichst einfache und übersichtliche Struktur
  - Konfigurationsanpassung im laufenden Betrieb
  - Out-of-the-box Erkennung von gängigen Frameworks
  - Optimal : Grafische Konfiguration

- Sensoren
  - Methodenlaufzeiten
  - Datenbankinformationen und detaillierte Informationen zu SQL Queries
  - Systemressourcen (zumindest JVM interne Ressourcen)
  - Optimal : Aufbau eines Callbaumes
- Oberfläche
  - Einfache Auswertungsmöglichkeiten
  - Verteilt startbar
  - Optimal: Anpassbare Oberfläche (Richclient oder Webbasiert)
- Historische Informationen
  - Langzeitspeicherung der Daten
  - Historische Vergleichsmöglichkeiten
  - Trendanalysen
- Automatisierte Überwachung
  - Definitionsmöglichkeit von Schranken („SLAs“)
  - Benachrichtigung bei Problemen

- Oft auf Basis von Logging Frameworks und/oder JMX
  - Verkomplizierung des Anwendungscodes, Wartbarkeit sinkt
  - Auswertung kompliziert und zeitaufwendig
  - Bugs im Monitoringcode sind möglich



```
package de.webshop.shoppingcart.service;

import java.util.logging.Level;

public class ShoppingCartService {

    // Performance measuring code
    private static final Logger LOGGER = Logger.getLogger("ShoppingCartPerformanceLogger");

    ShoppingCartDAO scDAO = new ShoppingCartDAO();

    public ShoppingCart findShoppingCartForCustomer (Customer c){
        // Performance measuring code
        long timeStart = System.currentTimeMillis();

        // ... Business Code ...
        ShoppingCart cart = scDAO.findByCustomerID(c.getID());

        // Performance measuring code ... messy
        long timeEnd = System.currentTimeMillis();
        if (LOGGER.isLoggable(Level.INFO)) {
            StringBuilder builder = new StringBuilder();
            builder.append("Logged performance statistics for shopping cart retrieval");
            builder.append("Current thread id: ");
            builder.append(Thread.currentThread().getId());
            builder.append("Timing is: ");
            builder.append(timeEnd - timeStart);
            LOGGER.info(builder.toString());
        }

        return cart;
    }

    public ShoppingCart calculateCart () {
        // Performance measuring code
        long timeStart = System.currentTimeMillis();
```

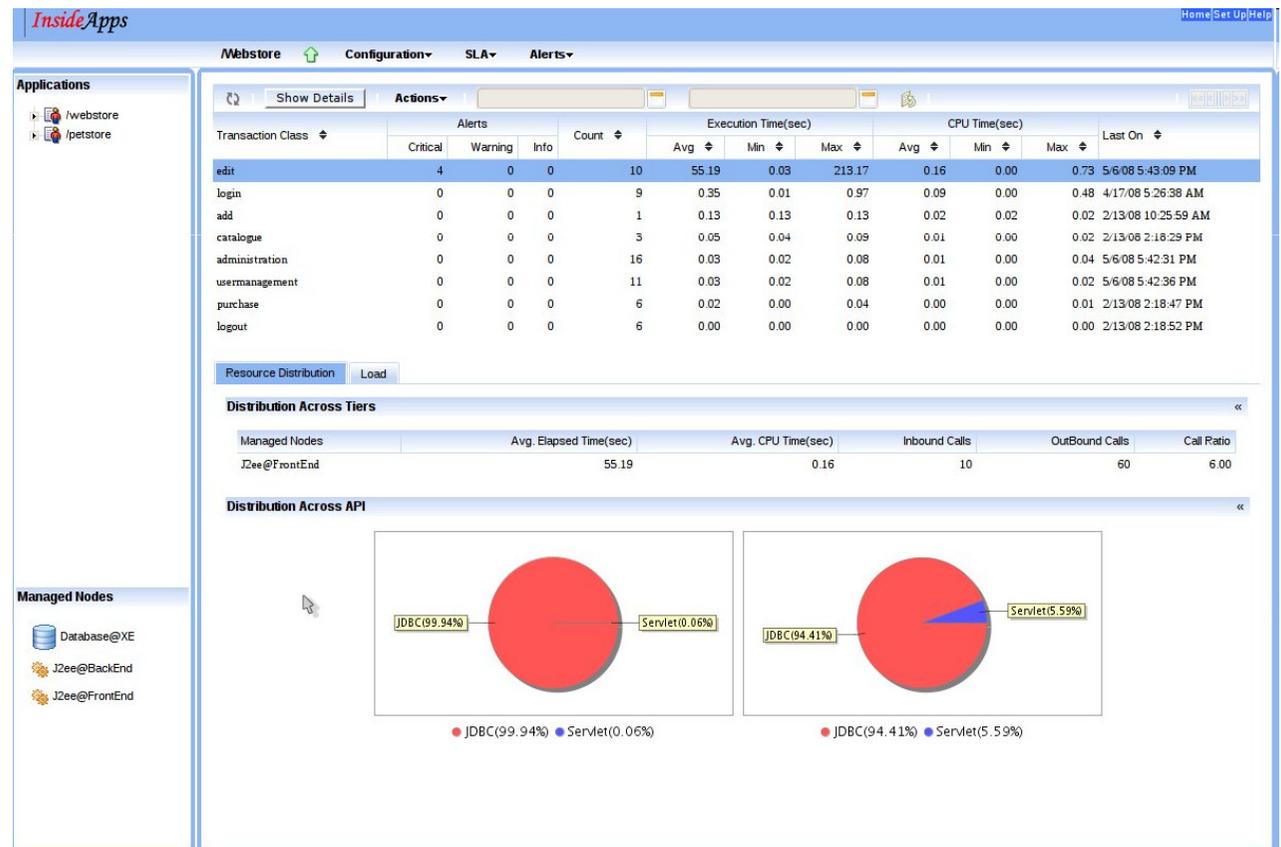
- JaMON
  - Framework zur Integration von Messpunkte in den Anwendungscode
  - Erlaubt Monitoring von SQL & Servlets
- SiMON
  - Simon ist ein Spinoff mit Nanotimer und baumartiger Darstellung



```
// Flexible/Dynamic - A String can represent anything - Page/Query/Method/...  
Monitor mon = MonitorFactory.start("pageHit."+userName);  
//...Code Being Timed...  
mon.stop();
```

Label	Hits	Avg ↓	Total	StdDev	LastValue	Min	Max	Active	AvgActive	MaxActive	FirstAccess
/myapp/pageHit.ssouza, ms.	5,000	200	1,000,000	59	102	20	8,000	1	1	5	9/18/06 9:54:53 PM 1
select * from employees where lname=?, ms.	4,227,398	86	364,624,391	455	102	42	454,990	1	1	5	9/18/06 9:54:53 PM 1
java.sql.Statement.executeQuery(java.lang.String), ms.	3,355,728	70	235,283,654	59	102	42	7,279	1	1	5	9/18/06 9:54:53 PM 1

- Kostenfreies Monitoring Werkzeug für Java (kostenpflichtiger Support)
- Transparente Integration der Messpunkte (bytecode modification)
- SLA Definition anhand von Methodenlaufzeiten
- Einfache historische Analyse (jedoch kein Vergleichsmöglichkeit)
- Komplizierte Konfiguration
- Datenbanküberwachung nur mit Vollversion möglich
- Wenig/Keine Entwicklungsaktivität



- Anwendungsmonitoring auf Basis definierter Performanceziele
- Automatische Untersuchung des „root causes“
- Kategorisierung und Beschreibung der Fehlerursache
- Ziel: Automatisches Auffinden von gängigen Fehlern

Status	Analysis	Server	Operation	Application	Avg. Time	Executions
FAILING	DB Connection Failure	local	ViewCategoryAction	jpetstore	370 ms	16
SLOW	Thread Contention	local	SearchProductsAction	jpetstore	10.13 sec.	13
SLOW	Slow Database	local	ViewProductAction	jpetstore	1.17 sec.	2
OK		local	ClientController	Glassbox Web Client	220 ms	2
OK		local	NewOrderFormAction	jpetstore	45 ms	4
OK		local	NewOrderAction	jpetstore	34 ms	8
OK		local	SignInAction	jpetstore	30 ms	9
OK		local	ViewCartAction	jpetstore	27 ms	4
OK		local	AddItemToCartAction	jpetstore	23 ms	15
OK		local	ViewItemAction	jpetstore	22 ms	55
OK		local	OperationDetailController	Glassbox Web Client	17 ms	8
OK		local	DoNothingAction	jpetstore	16 ms	31
OK		local	OperationHeadController	Glassbox Web Client	7.8 ms	3
OK		local	ViewProductAction	jpetstore	7.0 ms	12
OK		local	ConnectionHelper.getConnections	Glassbox Web Client	3.5 ms	3
OK		local	ConnectionController	Glassbox Web Client	0.75 ms	3
OK		local	ColumnHelper.getColumn	Glassbox Web Client	0.62 ms	6
OK		local	OperationBodyController	Glassbox Web Client	0.48 ms	3
OK		local	OperationHelper.getOperations	Glassbox Web Client	0.25 ms	1984

**SLOW OPERATION: SearchProductsAction**

**Cause: Java bottleneck due to too many operations waiting on the same resource**

Operation ran 13 times since 8/16/06 12:46 PM

Slow 11 times ( 84 %)  
Exceeded 1.0 sec. goal 11 times (84%)

Average Execution Time Overall: 10.13 sec.  
Average Execution Time While Slow: 11.97 sec.

**Technical Summary**

Thread Contention: When the SearchProductsAction operation ran slowly, it took an average of 7.88 sec., including time in method com.ibatis.jpetstore.presentation.action.BaseAction.acquireResource() waiting for threads to release a lock on a Java object.

- Open Source
- Einfache Installation
- Keine Kenntnis der Applikation notwendig
- Konfigurierbar
  - Definition der „Operations“
  - Eigene Sensoren
- Tipp: Kann auch gut mit anderen Monitoring Werkzeugen kombiniert werden



<http://www.glassbox.com/>

**FAILING OPERATION: ViewCategoryAction**

**Cause: Could not connect to the database**

Operation ran 43 times since 8/1/06 12:05 PM

Failed 1 times ( 2 %)      Average Execution Time Overall: 160 ms  
Exceeded 1.0 sec. goal 1 times (2%)      Average Execution Time While Slow: 6.21 sec.

**Technical Summary**

The ViewCategoryAction operation encountered errors when connecting to the jdbc:mysql://localhost/petstore database.

**Technical Details**

**DB Connection Failure**

**Operation:** com.ibatis.jpetsore.presentation.action.ViewCategoryAction  
**Location:** Sampson03

Executions	Run Time (avg)
41 OK	7.6 ms for OK
1 SLOW	6.21 sec. for SLOW
1 FAILING	280 ms for FAILING

**DB Connection Failure**

The operation's thread failed to get a database connection at the following points:

Database Connection URL: jdbc:mysql://localhost/petstore failed for 1 distinct connection requests.

A failure indicated by a data access problem: java.sql.SQLException, SQL state [28000], SQL error code [1045]: Access denied for user 'areallystupiduse'@'localhost' (using password: YES).

Class	Method	Line
com.mysql.jdbc.MySqlIO	checkErrorPacket	2921
com.mysql.jdbc.MySqlIO	checkErrorPacket	770
com.mysql.jdbc.MySqlIO	secureAuth411	3641

- Kosteneffektives Java Monitoringwerkzeug
- Unterstützt alle Java Anwendungen ab Java 1.4
- Transparente Integration (runtime bytecode modification)
- Flexible Basisarchitektur: Sensoren sind Plugins
  - Zeitmessung
  - Systemressourcen
  - Datenbank
  - Exceptions
  - Invocation sequences

The screenshot displays the inspectIT application monitoring tool. The main window shows an 'Invocation Sequence' table with the following data:

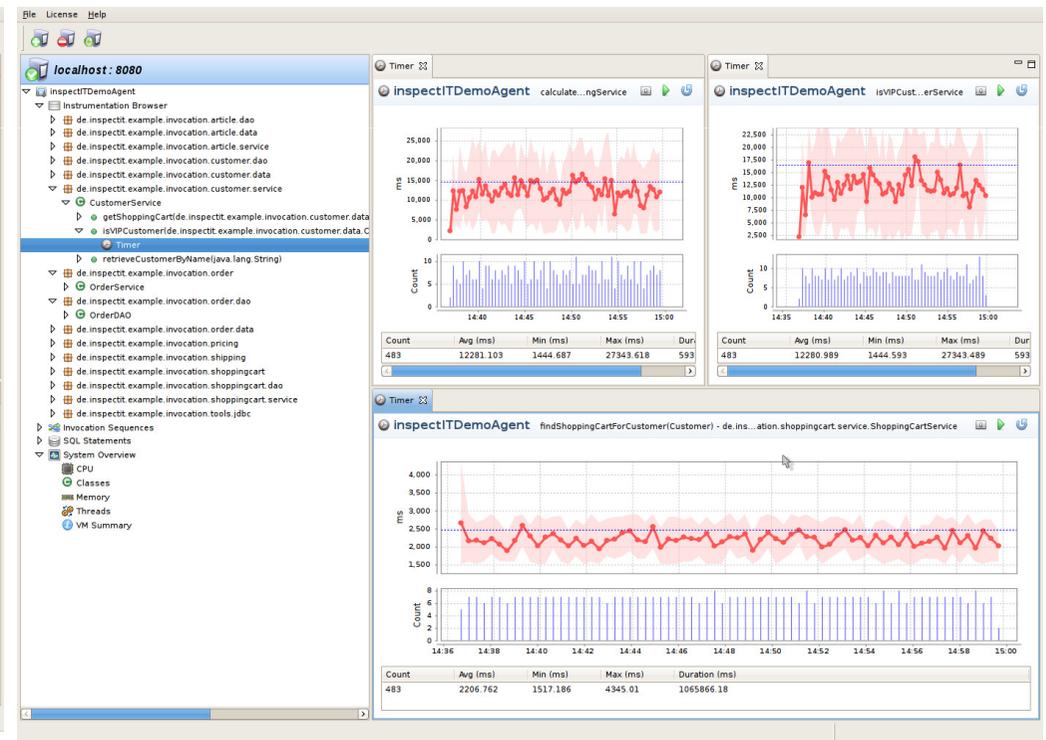
Start Time	Method	Duration (ms)	Child Count
19.06.2009 15:37:11.972	orderCartForCustomer(Customer) - de.inspectit.example.invocation.order.OrderService	32862.692	1.287
19.06.2009 15:33:52.012	orderCartForCustomer(Customer) - de.inspectit.example.invocation.order.OrderService	32816.742	1.587
19.06.2009 15:25:35.760	orderCartForCustomer(Customer) - de.inspectit.example.invocation.order.OrderService	31983.006	1.503
19.06.2009 15:37:12.001	orderCartForCustomer(Customer) - de.inspectit.example.invocation.order.OrderService	30700.306	1.201
19.06.2009 15:31:05.004	orderCartForCustomer(Customer) - de.inspectit.example.invocation.order.OrderService	30078.904	1.459
19.06.2009 15:34:59.645	orderCartForCustomer(Customer) - de.inspectit.example.invocation.order.OrderService	29825.784	1.419
19.06.2009 15:33:08.179	orderCartForCustomer(Customer) - de.inspectit.example.invocation.order.OrderService	29219.016	1.439
19.06.2009 15:44:44.013	orderCartForCustomer(Customer) - de.inspectit.example.invocation.order.OrderService	29024.467	1.347
19.06.2009 15:39:36.941	orderCartForCustomer(Customer) - de.inspectit.example.invocation.order.OrderService	28955.14	1.413
19.06.2009 15:25:35.753	orderCartForCustomer(Customer) - de.inspectit.example.invocation.order.OrderService	28802.178	1.325
19.06.2009 15:43:30.622	orderCartForCustomer(Customer) - de.inspectit.example.invocation.order.OrderService	28459.079	1.249

The bottom pane shows a call hierarchy for the selected method:

Method	Duration (ms)	Exclusive duration	Cpu Duration (nr)	SQL
orderCartForCustomer(Customer) - de.inspectit.example.invocation.order.OrderService	32862.692	1.683	300	
getShoppingCart(Customer) - de.inspectit.example.invocation.customer.service.CustomerService	2987.197	0.174	20	
getArticles() - de.inspectit.example.invocation.shoppingcart.ShoppingCart	0.042	0.042	0	
verifyAllArticlesAvailable(Article[]) - de.inspectit.example.invocation.article.service.ArticleService	1473.84	8.975	20	
calculatePriceFor(Article[]) - de.inspectit.example.invocation.pricing.PricingService	27677.786	0.183	260	
calculatePrice(Article[]) - de.inspectit.example.invocation.pricing.PricingService	2.467	1.619	0	
calculateDiscountForCustomer(Customer) - de.inspectit.example.invocation.pricing.PricingService	27675.136	0.134	260	
isVIPCustomer(Customer) - de.inspectit.example.invocation.customer.service.CustomerService	27675.002	85.698	260	
retrieveOrdersForCustomer(Customer) - de.inspectit.example.invocation.order.OrderService	1468.249	0.264	0	
getArticleDs() - de.inspectit.example.invocation.order.data.Order	0.075	0.075	0	
findByDI(String) - de.inspectit.example.invocation.article.service.ArticleService	72.565	0.242	20	
getPrice() - de.inspectit.example.invocation.article.data.Article	0.077	0.077	0	
findByDI(String) - de.inspectit.example.invocation.article.service.ArticleService	72.549	0.243	0	
getPrice() - de.inspectit.example.invocation.article.data.Article	0.078	0.078	0	
findByDI(String) - de.inspectit.example.invocation.article.service.ArticleService	72.598	0.291	0	
getPrice() - de.inspectit.example.invocation.article.data.Article	0.077	0.077	0	
findByDI(String) - de.inspectit.example.invocation.article.service.ArticleService	72.557	0.246	0	
getPrice() - de.inspectit.example.invocation.article.data.Article	0.082	0.082	0	
findByDI(String) - de.inspectit.example.invocation.article.service.ArticleService	72.592	0.263	0	
getPrice() - de.inspectit.example.invocation.article.data.Article	0.076	0.076	0	
findByDI(String) - de.inspectit.example.invocation.article.service.ArticleService	72.57	0.242	0	



- Schnelle und einfache Installation
- Minimaler Overhead
- Trendanalyse basierend auf historisierten Daten
- Flexible Oberfläche basierend auf Eclipse RCP



- Vereinfachte (meist grafische) Konfiguration
- Tracing von Benutzertransaktionen über mehrere, heterogene Plattformen
- Berechtigungskonzepte
- Integration von Reporting Funktionalitäten
- Flexibel anpassbare Monitoring Oberflächen („Cockpits“, „Dashboards“)
- Integration in gängige Systemmonitoringsuiten
- Integration von Zusatzfunktionalitäten wie Memory Analyse / Thread dumps

- Performance Engineering Werkzeuge können für wenig Geld integriert werden
  - Freie/Günstige Lösungen bieten für kleine Projekte einen optimalen Einstieg/Lösung
  - Werkzeugauswahl sollte anhand der konkreten Projektanforderungen getroffen werden
  - Performance Schulungen / Experten Teams („A fool with a tool is still a fool“)
- Dennoch: Ohne ein „Management von Performance“ bringt auch das Beste Werkzeug wenig