5.– 8. September 2011
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Vorhof zur Hölle

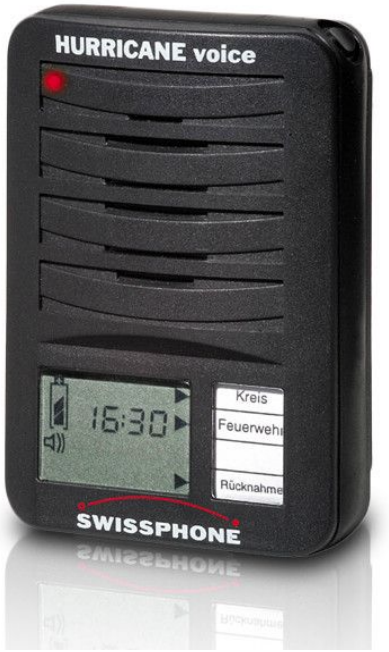
Betriebsaspekte bei der Applikationsentwicklung

Stefan Hildebrandt

consulting.hildebrandt.tk

Bekannt?

- SMS von Nagios?
- Pieper dabei?



Einleitung

- Erwartung an Produktiv eingesetzte Software
  - Anwender ist zufrieden
  - Betrieb und Entwickler haben nur planmäßigen Aufwand
- Störungsfrei
  - System befindet sich immer in einem definierten Zustand
  - Ausnahmen werden aktiv gemeldet (Email, Pager, InApp, ...)
  - Es ist niemand damit beschäftigt, auf das System zu schauen!
- Performant
  - Alle Aufgaben werden in der dafür vorgesehenen Zeit erfüllt
    - Definition?!?
    - Messungen
    - Monitoring

Einleitung

- Es gibt Charaktere die (unbewusst?) einen störungsfreien Betrieb verhindern
  - Der gewisse Kick?
  - Fühlen sich ansonsten nicht wichtig genug?
  - War schon immer so?
  - Stehen nachts gerne auf?
  - Wollen nicht nach Hause?
  - …

Voraussetzung: Fertig!

- 100% „fertige“ Software
  - keine halb umgesetzten Features
  - Keine TODOs
  - Keine nicht tragfähigen „Hacks“
- Gute Testabdeckung
  - Hoher Automatisierungsgrad → CI
  - Repräsentatives Testdatenset
    - Positiv-, Rand- und Negativtests

  →
  - Keine Hecktick von Außen zulassen
  - Übertriebene Anforderungen rechtzeitig anmerken
  - Wenn notwendig „Nein“-Sagen
  - Software Craftsmanship, CCD

Fehler! Was nun?

- Ohne aussagekräftige Fehlermeldungen und Logging
  - Bei Fehlern hilflos!
- Bileblog: How to guarantee perfectly unhelpful error messages
  - http://www.bileblog.org/2003/07/how-to-guarantee-perfectly-unhelpful-error-messages/

How to guarantee perfectly unhelpful error messages (Bileblog)

- Always, always, show stacktraces. I realise that the more naive amongst you might think that this is clutter, or that this is inelegant and/or unprofessional. You lot, bugger off. For the rest of you, just think of how nice stacktraces are. After all, it's your code, why shouldn't the user get to see all of it? So what if it's irrelevant to them, it matters to you, and you're the most important person here!

How to guarantee perfectly unhelpful error messages (Bileblog)

- Wrap your exceptions as much as possible. If you a reasonably tiered system, never let an exception bubble up. Ohno, always wrap it. Best results are achieved if you always wrap an exception, then throw a new specific one, and keep repeating this along every step of the process. This ensures that your stacktraces (when they do finally splatter all over your users console) are rich with content. They will span pages, guaranteeing to impress other developers. If users complain, then feel free to retort with any of the established opensource responses (see article last week).

How to guarantee perfectly unhelpful error messages (Bileblog)

- If it's worth showing once, it's worth showing two or three times. Apparently, modern day users ignore single stacktraces, even if they do span 4-5 pages. To combat this ignorance, log your stacktrace multiple times. This ensures that the user will at least see one of them.
- Use a lot of reflection and many dynamic proxies. This technique is particularly effective if you don't bother with catching specific exceptions, but just throw everything back up. The resultant stacktraces are generic enough that you can probably blame the user for them.

How to guarantee perfectly unhelpful error messages (Bileblog)

- Avoid plain English at all cost. Using coherent easily understood error messages might result in losing the respect of your fellow developers. You will be suspected of spending too much time with soft-skilled people like usability experts, end users, QA folks, or clueful managers. Stick with stacktraces and obscure exceptions. Not only do they guarantee job security, they also ensure your users always come back to you squealing for help. A 5 page long ArrayIndexOutOfBoundsException is a lot more manly than a pathetic little one liner whimpering out “invalid character ‘x’ in query ‘xflibble the wibble at blah’ at position 1″

How to guarantee perfectly unhelpful error messages (Bileblog)

- NullPointerException is your friend. Nothing is easier to fix than an NPE, so never bother testing for that condition or handling it in any way. Let nature (the JVM) takes its course and do the work for you. It looks cooler that way, and you improve your ROI.

Fehlermeldungen (User Interface)

- Unterscheidung interne vs. externe Anwender
  - Was kann und darf dem Anwender zugemutet werden
- Minimum:
  - Aussagekräftige Fehlermeldung
  - Datum, Serverzeit, ggf. Server ID
  - Anweisungen was nun geschehen soll
- Zusätzlich:
  - IDs von Objekten
  - Speziellen Ansprechpartner im Support
  - Ggf. Link in Fehlerdatenbank
- Keine Stacktraces → siehe Logfiles
  - Ggf. über Berechtigungen einblenden (Entwicklung, Test, ...)
- Keine automatische Weiterleitung auf Startseite!

Logging

- Wer ist der Empfänger?
  - Basisbetrieb
  - 1./2. Level-Support
  - Entwickler

  → Unterschiedliche Dateien für unterschiedliche Empfänger
- Logging kostet Leistung (CPU/IO/Plattenplatz)
  - Unterschiedliche Konfigurationen für unterschiedliche Stages
- Fehler im Logging-Code können Fehler in der Anwendung provozieren

  → Auch Log-Settings müssen getestet werden!

Logging für Betrieb

- Betriebszustand des Systems erkennen
  - Kontinuierliches Logging nicht ideal, da Parsen und Aufbereitung der Daten notwendig
    - Alternativen (Performance, JVM) folgen
- Post-Mortem-Analyse
  - Frage: Was war die Ursache für einen Absturz?
  - Sicherstellung das notwendige Informationen ausgegeben werden
    - JVM
      - kill -quit in shutdown-Skript einbauen
      - Achtung: Ausgabe geht auf System.out
      - Dumps bei OutOfMemoryError

Logging für 1./2. Level-Support

- Sollte in der Anwendung ersichtlich sein
  - Z.B. Auftragsstatus mit Fehlermeldung
- Für 2. Level-Support ggf. Logfiles aus denen weitere Aktionen für bereits bekannte Fehler abgeleitet werden können.
- Lösungsdatenbank für bekannte Fehler
  - Z.B. in Form eines Wikis
  - Beschreibung des Fehlers
    - Ursache
    - Auswirkung
  - Vorgehen zur Behebung des Fehlers
    - z.B. Schritt-für-Schrittanweisung für die manuelle Pflege in beteiligten System, Wiederaufnahme, ...

Logging für Entwickler

- Ziele
  - Unterstützung bei der Entwicklung
  - Fehlererkennung und -behebung im Betrieb
  - Weiterentwicklung

Logging: DONTs

- Debug-Informationen unnötigerweise entfernen
  - Verhindert saubere Erkennung der Code-Stelle
- Extensive Aufbereitung von Daten
  - Erhöht die CPU-Last
- Extensives Logging
  - Performance-Probleme durch limitierendes IO
    - Beispiel: Konsolen-Ausgabe
  - Synchronisation auf Logdatei

Logging: DOs

- Angabe von Kontextinformationen
  - NDC – Nested Diagnostic Context
    - Ideal für verschachtelte Strukturen z.B. Batchverarbeitung
  - MDC – Mapped Diagnostic Context
    - Sammeln von Daten
    - Eigener Appender für Fehler, mit mehr MDC Daten
- Resourcen-schonendes Logging
  - if(LOG.isDebugEnabled()){....}
    - Achtung: Fehlerquelle!
  - LOG.debug("Error updating bean {}", bean.getId())
- Exception-Versendung
  - Filter und Limit notwendig

Logging: DOs

- Rekonfiguration im Betrieb
  - Über Austausch der Datei
    - log4j.PropertyConfigurator.configureAndWatch(logFilePath , logFileWatchDelay);
    - Spring Utils
    - Programatisch
  - Injecten des Loggers
    - Kann mittels Parameter umgestellt werden

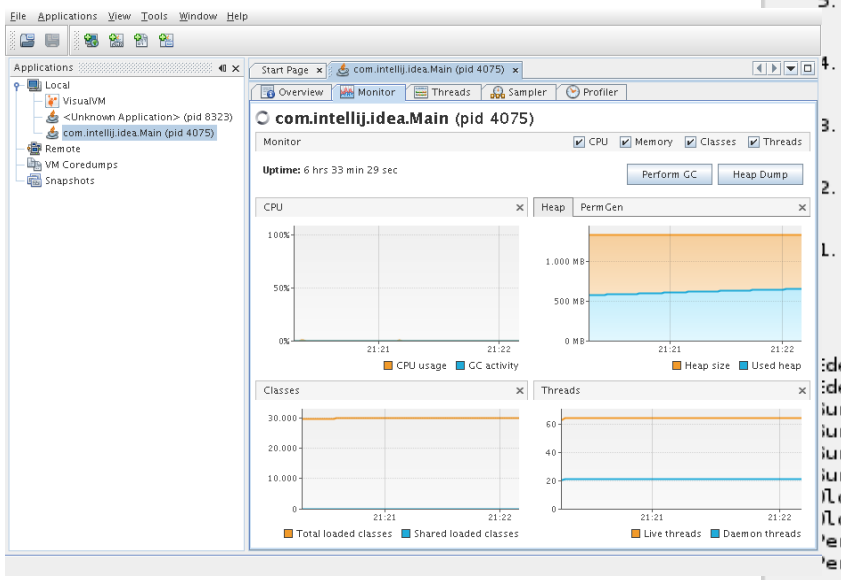
Logging: DOs

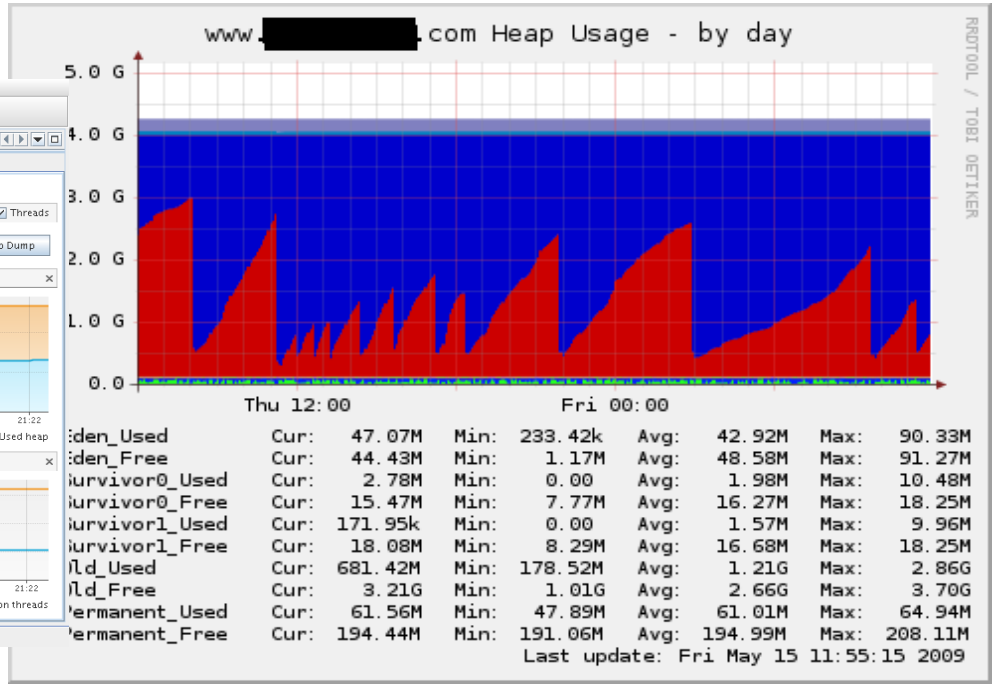
- JMX, Webfrontend

```
public class Log4jLevelChanger {
   public void setLogLevel(String loggerName, String level) {
      if ("debug".equalsIgnoreCase(level)) {
         Logger.getLogger(loggerName).setLevel(Level.DEBUG);
      } else if ("info".equalsIgnoreCase(level)) {
         Logger.getLogger(loggerName).setLevel(Level.INFO);
      } else if ("error".equalsIgnoreCase(level)) {
         Logger.getLogger(loggerName).setLevel(Level.ERROR);
      } else if ("fatal".equalsIgnoreCase(level)) {
         Logger.getLogger(loggerName).setLevel(Level.FATAL);
      } else if ("warn".equalsIgnoreCase(level)) {
         Logger.getLogger(loggerName).setLevel(Level.WARN);
      }
   }
}
```

Wie geht's?

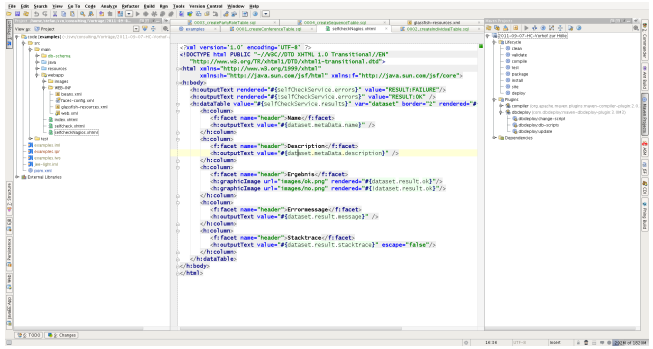
- VM-Überwachung
  - Viele Lösungen, die Speicher- und CPU-Auslastung überwachen
  - Jstat, jvisualvm, Nagios, munin, ...





Wie geht's?

- Applikationsüberwachung
  - Erstellung einer Schnittstelle für die Überwachung
    - Servlet, JMX, Webservice, Rest-Service, …
  - Kritische Komponenten implementieren Check-Interface
  - Zentraler Service nutzt Liste der Services aus IOC-Container zur Überprüfung aller Services
    - Datenbank, Externe Systeme, ...
  - Services cachen, bzw. senden Fehler an zentrale Komponente



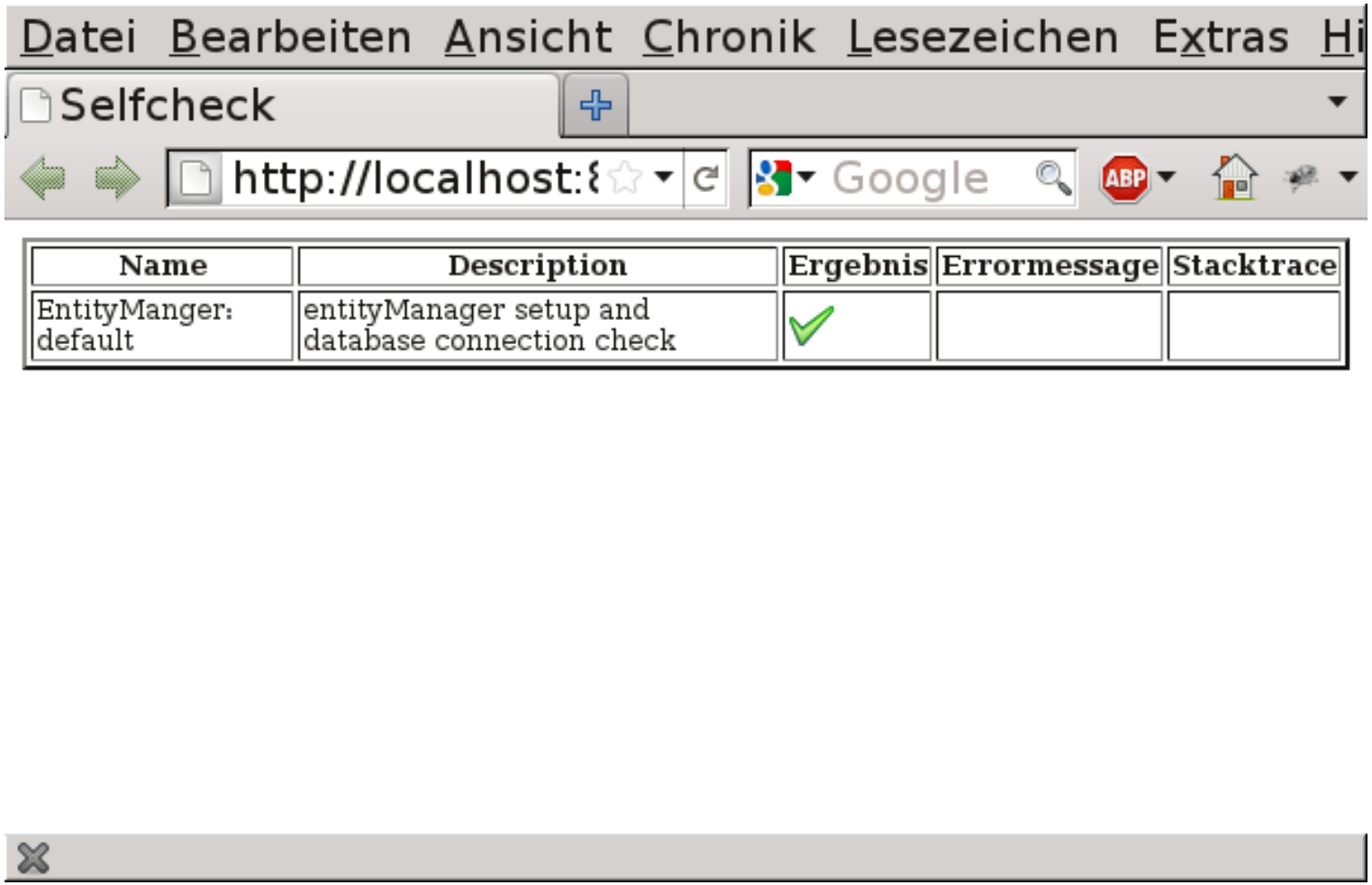
Wie geht's dir?

```
public interface SelfcheckSupport {

   SelfCheckMetaData getMetaData();

   SelfCheckResult check();
}
```

Wie geht's dir?

```
public void doCheck() {
   final Set<Bean<?>> beans = listBeans();
   for (Bean<?> bean : beans) {
      final SelfcheckSupport reference = resolveBean(bean);
      checkBean(bean, reference);
   }
}
```

Wie geht's dir?



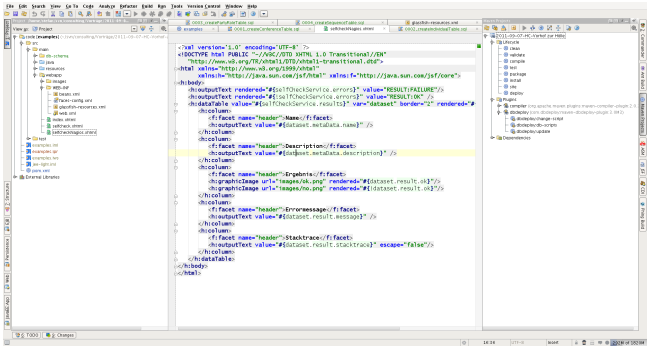
Name	Description	Ergebnis	Errormessage	Stacktrace
EntityManger: default	entityManager setup and database connection check	✓		

Weg damit!

- Release-Erstellung
  - IDE?
  - ANT/MAVEN/IVY/GRADE/...
  - Spezielle Umgebung?
- Übergabe an den Betrieb
  - 1 Artifact (EAR/WAR/JAR/ZIP)
    - Programm, DB-Schema, Tools, ...
  - Keine händische Anpassung an die Umgebung
    - Keine Regelmäßige Anpassung von Konfigurationen
    - Appserver?

db-deploy

- Changelog-Tabelle im Schema
- Änderungen in vielen Deltafiles
- Per Command-Line-App, ant oder Maven-Plugin
  - Erstellung eines Delta-Skripts für ein Schema
  - Ausführen der Änderungen
- Auslieferung an Betrieb
  - Tool
  - Ausgangsskripte



sh/ant/gradle/puppet/chef

- z.B. Template-basierte Ansätze
  - Basisartifakt wird vorm installieren mit einer Properties-Datei/Datenbank neu konfiguriert
- Entpacken und Konfiguration von Appserver

Passt alles rein?

- Abschätzung ob ausreichend Clients versorgt werden können
  - Bestimmung der Kapazität auf Zielsystem
  - Footprint einer Session / eines Requests prüfen
- Tools: Eclipse Memory Analyzer
  - http://www.eclipse.org/mat/

Löcher?

- Typische Memory Leaks
  - HttpSession
    - Zu viele Objekte in der Session
      - Ohne Aufräumen
    - Framework-Schwächen
      - JSF 1.0 ViewStates (z.B. a4j LRUMap)
  - Selbstgebaute Caches

    → Caching Frameworks verwenden
  - Fehler bei der Verwendung von:
    - ThreadLocals
    - Threads

Müllabfuhr

- Auswahl
  - Stark unterschiedliche Anforderungen
    - Rest vs. Session
    - Interactive vs. Batch
  - Multi Core Server
    - Parallel GC
- Überwachung
  - GC Logging
  - Analyse-Tools
  - Überwachung
    - z.B.: > 20% GC Time in 5 Minuten
  - GCViewer
    - http://www.tagtraum.com/gcviewer.html

Bummelzug

- Zwei Arten von Performance
  - Durchsatz
  - Latenz
- Messungen
  - Von „Außen“
    - Profiling
    - Messwerkzeuge
  - Von „Innen“
    - Framework interne Funktionen
      - z.B. Hibernate Statistics
    - Geschickter Einsatz von Messwerkzeugen
      - Hibernate Lazy-Initialization
      - Fachliche Messungen

Klassisches Profiling

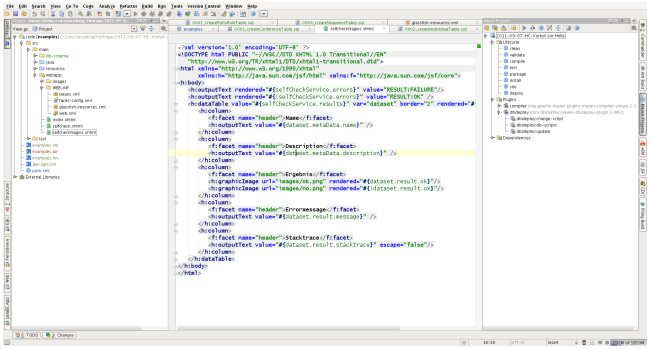
- Bei der Entwicklung hilfreich
  - Anzeige der Aufwändigsten Methoden
  - Rein Drillen
  - Relative Aussage für HotSpot meistens OK
- Großer Overhead
  - Verfälscht Relation
    - zu externen Systemen
    - kleine Methoden
  - Nicht für Produktivsysteme geeignet
- VisualVM
  - Teil des JDKs

Einfache Messwerkzeuge

- Jetm
  - Einfach, Schnell, robust
  - http://jetm.void.fm
- Java Simon - Simple Monitoring API
  - JDBC-Monitor Treiber
  - http://code.google.com/p/javasimon/
- JAMon
  - Threshholds , Active Counter, JDBC-Monitor Treiber
  - http://jamonapi.sourceforge.net
- Kicker
  - Tracing
  - https://se.informatik.uni-kiel.de/kieker/

Beispiel

- Einbindung
  - Deklarativ an Schnittstellen: Servlet-Filter, JaxWS-Interceptoren, JDBC-Driver
  - AOP TODO
  - Programmatisch



Dicke Hose -APM Tools

- Application performance-Management
- Features:
  - Basisdatenmonitoring (Speicher, CPU, (IO))
  - Performance Monitoring über Systemgrenzen hinweg
    - Datenbank
    - Appserver (Java, .Net, …)
    - Clients (Java, .Net, Browser, ...)
  - Versprechen Lösung für Betrieb, Entwicklung und Testabteilung zu sein
- Schwer zu Realisieren:
  - Tracing
  - Vermessung von Geschäftsprozessen

Dicke Hose – Meine Erfahrungen

- Deutlich höherer Einarbeitungshürde
  - Dann hilfreich
- Für mehr Informationen bitte melden!

Fragen?

Vielen Dank!

Stefan Hildebrandt

consulting.hildebrandt.tk

consulting.hildebrandt.tk

- Freiberuflicher Software Entwickler, Software Architekt, Berater und Coach
  - Java Entwicklung auf dem Server
    - JEE (EJB, JPA, WS, CDI)
    - Spring, Seam
    - CleanCode, TDD, CodeReviews
    - Buildmanagement, CI
  - Betriebsaspekte
    - Tomcat, JBoss, JVM, Linux, Solaris
    - Fehlersuche, Performance-Analyse, ...
  - Agile Methoden
  - Email: consulting@hildebrandt.tk
  - Twitter: @hildebrandttk