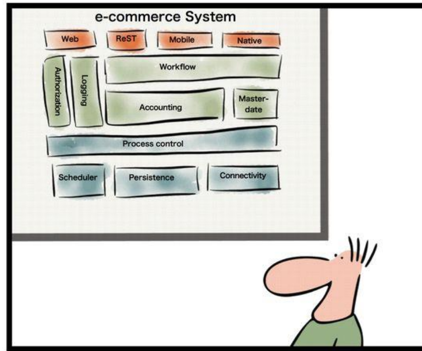
Architekturvorgaben und Entwicklungsrichtlinien automatisiert prüfen

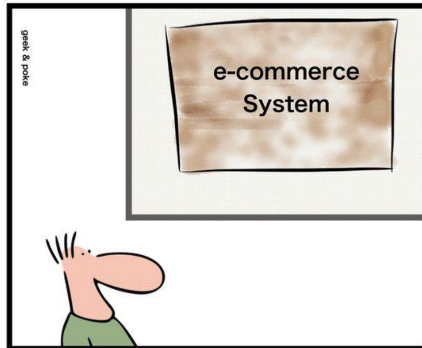
Max Bechtold
David Burkhart

Entwicklertag Karlsruhe 2016

HOW TO DRAW THE ARCHITECTURE OF YOUR SYSTEM



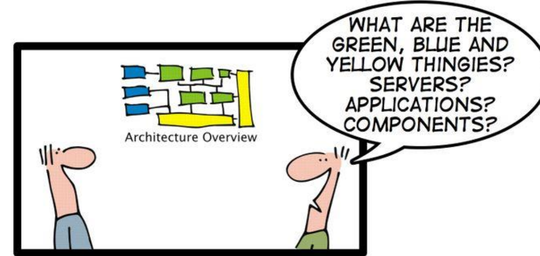
RULE 1: JUST MAKE IT NICE

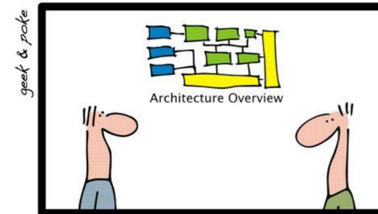


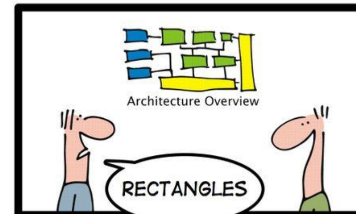
RULE 2: AND NOT REALISTIC!!!

http://geekandpoke.typepad.com/geekandpoke/2012/04/architecture-drawing.html

ENTEPRISE ARCHITECTURE MADE EASY







PART 1: DON'T MESS WITH THE GORY DETAILS

http://geekandpoke.typepad.com/geekandpoke/2009/11/enterprise-architecture-made-easy-part-1.html

Vorgaben und Richtlinien

© 2016 andrena objects ag

Vorgaben und Richtlinien – Wozu?

- ✓ Einheitlicher Stil
- ✓ Vermeidung von Antipatterns
- ✓ Vermeidung zukünftigen Aufwands
- ✓ Weniger Einarbeitung durch Konsistenz
- ✓ Verbesserte Wartbarkeit und Erweiterbarkeit

© 2016 andrena objects ag

Feedback auf allen Ebenen

Save / before Commit

- Support in der IDE
- Direktes Feedback für den Entwickler

Commit / Push / Integration

- Support im Continuous Build
- Feedback für das gesamte Team

Release / Delivery

- Support im Release-Build
- Trends über Sprints hinweg, Historisierung

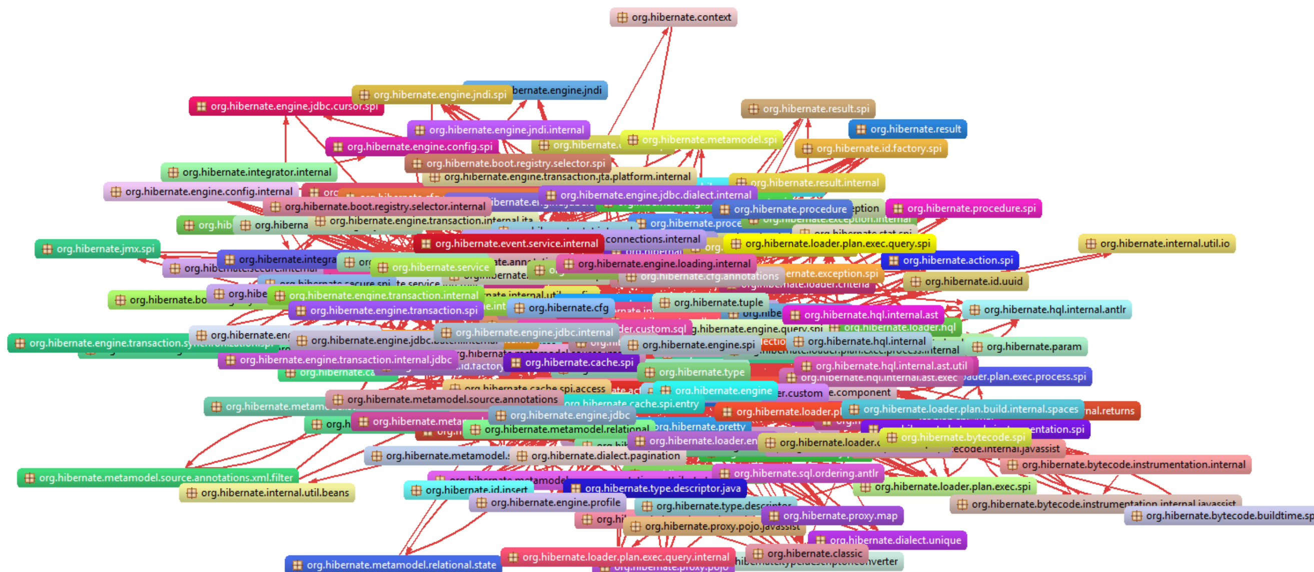
© 2016 andrena objects ag

Agenda

- **Konventionen & Checks**
  - Packages / Module
  - Files

© 2016 andrena objects ag

Modul / Paket - Ebene



© 2016 andrena objects ag

Acyclic dependencies principle

Modul-Abhängigkeiten

- Projekte im Workspace
- Maven-Module

Muss meistens zyklenfrei sein

Paket-Abhängigkeiten

Sind oft nicht zyklenfrei

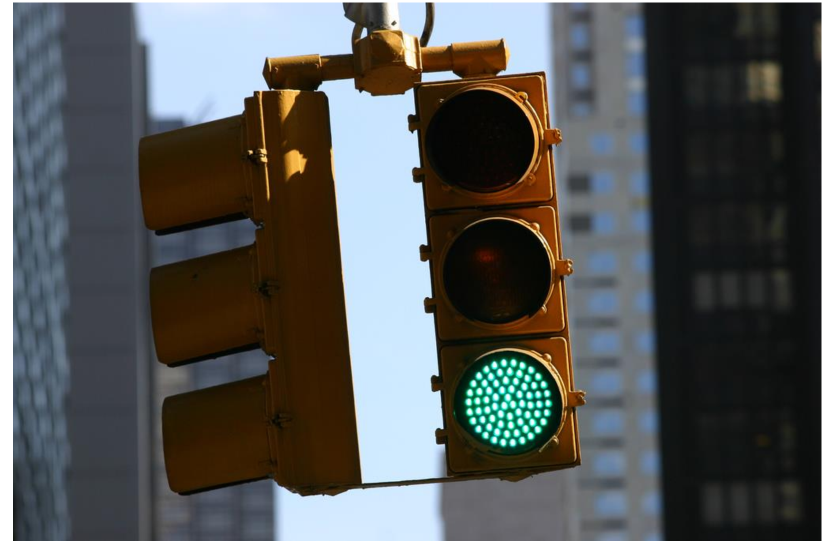
- Verhindern weitere Module
- Machen Refactoring schwieriger
- Microservices?

© 2016 andrena objects ag

Paketzyklen verhindern

*Feedback beim **Build***

- JDepend plain (z.B. Unit-Test)
  - Benötigt class-Files
- Maven Enforcer Rule
- SonarQube
- **Modularisieren!**



https://flic.kr/p/Boiy7

© 2016 andrena objects ag

Paketzyklen reparieren

***Project USUS**: Feedback beim Speichern*

CourseListView → OpenCourseEditor ··· → CourseEditor

CourseEditor → ReloadCourseListAfterJobDone

ReloadCourseListAfterJobDone → ReloadCourseList

ReloadCourseList ⇄ CourseListView

© 2016 andrena objects ag

Demo: Paketzyklen

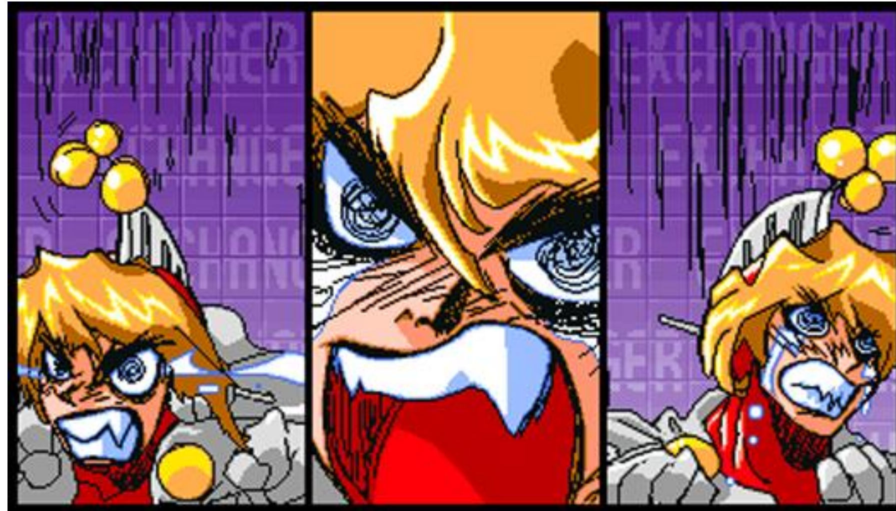
© 2016 andrena objects ag

Paketabhängigkeiten allgemeiner

- Macker
- JDepend
- Als PMD Regel
- Kommerzielle Tools
- **Modularisieren!**

© 2016 andrena objects ag

Demo: Paket-Abhängigkeiten



https://innig.net/macker/faq.html

© 2016 andrena objects ag

Modulabhängigkeiten: Frameworks / Libraries

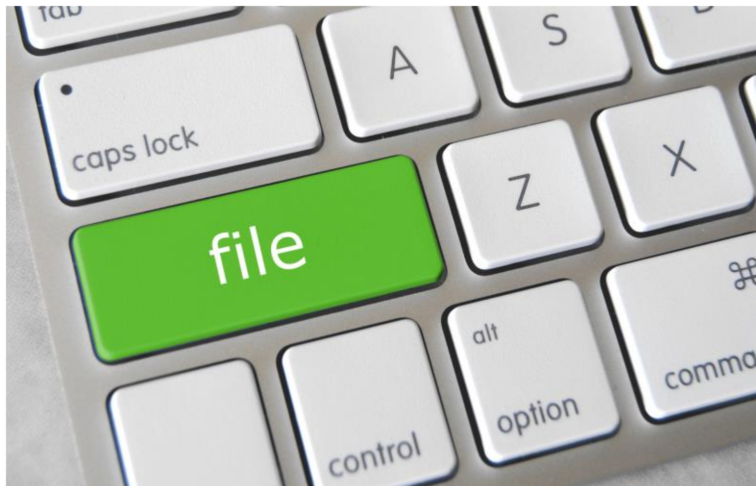
Maven Enforcer Plugin

- Constraints an Dependencies (Verbote, Versionen, Scopes, etc.)
- Constraints an die Umgebung (OS, Dateien, Variablen, etc.)
- Eigene Regeln

http://maven.apache.org/enforcer/enforcer-rules/index.html

© 2016 andrena objects ag

File - Ebene



http://flic.kr/p/roafis

© 2016 andrena objects ag

Checks für Java-Code & Ressourcen

- Großteil der Entwicklungszeit auf File-Ebene
- Plug-Ins für IDE & Build
  - Checkstyle
  - PMD
  - Findbugs, Sonar(Lint), …
- 'handgemachte' Checks per Unit-Test

© 2016 andrena objects ag

Checks für Java-Code

Methode 1: Konventionen prüfen per **Classpath-Scan**

- Z.B. mit **`ClassPathScanningCandidateComponentProvider`**
  - → org.springframework.context
  - liefert Typinformation & Annotationen
  - kombinierbar mit Java Reflection API

- noch mächtiger: **`Reflections`**-Library
  - → org.reflections
  - Checks von Typhierarchien, Methodensignaturen, Aufrufstellen

Beispiel: Einfache Namenskonventionen testen

© 2016 andrena objects ag

Demo: Classpath-Scan

© 2016 andrena objects ag

Checks für Java-Code

Methode 2: eigene Regeln für **Checkstyle**

- erweiterbar durch reguläre Ausdrücke
  - `RegExpSingleline` und `RegExpMultiline`

Beispiele

- keine TODO-Kommentare im Code
- keine Unit-Tests mit `@Ignore`
- kein auskommentierter Code

© 2016 andrena objects ag

Checks für Java-Code

Methode 3: eigene Regeln für **PMD**

- AST-basierte Checks
- Regeln konfigurieren mit XPath

Beispiele

- keine Aufrufe von gefährlichen/missverständlichen Methoden
- keine verbotenen APIs/Frameworks einsetzen
  - Logging-Frameworks, Date-API
- keine geschachtelten Typdeklarationen

© 2016 andrena objects ag

Demo: Eigene Checkstyle/PMD-Regeln

© 2016 andrena objects ag

Checks von Ressourcen

Methode 1: eigene **Checkstyle**-Regeln

- anwendbar auf beliebige textbasierte Ressourcen
  - `java, properties, xml, sql, ...`

Beispiel: SQL-Skripte testen

- keine anonymen Constraints (→später: `drop constraint` …)
- Namenskonventionen für Constraints / Tabellen / Spalten …

→ JDBC-Metadaten! (weitergehende Checks der aktuellen DB)

© 2016 andrena objects ag

Demo: SQL-Constraints prüfen

© 2016 andrena objects ag

Checks von Ressourcen

Methode 2: **Files** parsen

- Libraries für Datei-Formate nutzen!
  - z.B. JAXB für XML

Beispiele:

- sortierte Dependencies in pom.xml (→ test-Scope am Ende)
- keine redundanten Properties in verschiedenen Modulen

© 2016 andrena objects ag

Demo: Properties prüfen

© 2016 andrena objects ag

Fazit

© 2016 andrena objects ag

Best practices

- Regeln / Konfiguration der Tools versionieren
  - ✓ *Gleiche Konfiguration über alle Ebenen hinweg*
- Zero Warnings
- Steigende Severity auf späteren Ebenen
- Trotz Automatisierung: Reviews + Bewusstsein

© 2016 andrena objects ag

Automatisiere Vorgaben und Richtlinien!

- Oft einfacher, als gedacht
- Insgesamt lohnt sich der Aufwand
- Bewusstsein wird durch automatisierte Checks geschaffen
- Weniger „Threads“ im Kopf der Entwickler/-innen

© 2016 andrena objects ag

Links

- http://www.projectusus.org/
- https://github.com/BenRomberg/no-package-cycles-enforcer-rule
- https://github.com/nidi3/jdepend
- https://github.com/andrena/macker-maven-plugin
- https://pmd.github.io/pmd-5.4.2/customizing/xpathruletutorial.html
- http://maven.apache.org/enforcer/enforcer-rules/index.html
- http://checkstyle.sourceforge.net/config_regexp.html
- http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/context/annotation/ClassPathScanningCandidateComponentProvider.html
- https://github.com/ronmamo/reflections

© 2016 andrena objects ag

Danke für die Aufmerksamkeit!

© 2016 andrena objects ag