



Continuous Architecture Management

Erkennen und Verhindern von struktureller Erosion

Ingmar Kellner
hello2morrow GmbH
Mai 2012





Agenda

- ▶ Motivation des Software Architekten
- ▶ Ursachen und Folgen struktureller Erosion
- ▶ Lösungsansätze
- ▶ Entwicklungsprozess



Motivation

Als Software Architekt will ich ...

- ▶ ... dass meine Software eine hohe Qualität hat
- ▶ ... wissen, wie die interne Struktur wirklich aussieht
- ▶ ... strukturelle Erosion verhindern
- ▶ ... wissen, wo aktuell die Problemstellen sind
- ▶ ... eine aktuelle Dokumentation der Architektur
- ▶ ... existierende Software Module wiederverwenden



Und dennoch sieht es oft so aus:





Offensichtliche Widerstände

- ▶ Zeitdruck
- ▶ Was bedeutet gute Qualität?
- ▶ Es ist zu kompliziert Metriken zu berechnen und sinnvolle Schwellenwerte zu definieren
- ▶ Es ist aufwändig, die Architekturdokumentation mit der Entwicklung abzugleichen
- ▶ Die Abhängigkeiten zwischen Teilen der Software manuell zu überprüfen ist nicht machbar
- ▶ Fehlende Toolunterstützung, z.B. zeigt die IDE keine Warnung an, wenn eine unerlaubte Abhängigkeit eingebaut wird.



Realitätscheck

- ▶ Gibt es bei Ihnen verbindliche Qualitätsregeln?
- ▶ Werden diese Regeln täglich automatisch geprüft?
- ▶ Gibt es eine formelle Architekturdefinition?
- ▶ Wird der Code automatisch und täglich auf Einhaltung der Architekturdefinition geprüft?
- ▶ Denken Sie, dass in diesem Bereich mehr zu tun ist?



- ▶ Motivation
- ▶ Ursachen und Folgen der strukturellen Erosion
- ▶ Lösungsansätze
- ▶ Entwicklungsprozess

Einige Gründe für die strukturelle Erosion

- ▶ Wissen und Fähigkeiten im Team sind ungleich verteilt
- ▶ Kopplung und Komplexität wachsen schnell
- ▶ Vielfach gibt es keine (aktuelle) formale Architektur
- ▶ Mythos der agilen Superhelden
- ▶ In den meisten Projekten wird die Qualität am Schluss betrachtet
- ▶ Desinteresse an Qualität: Management betrachtet die Software als “black box”
- ▶ Das Gesetz der Software Entropy [Lehmann]
 - ▶ Eine Software, die benutzt wird, wird verändert werden.
 - ▶ Wenn eine Software verändert wird, steigt die Komplexität, es sei denn, man arbeitet aktiv dagegen.



Symptome der Erosion (Robert C. Martin)

- ▶ Rigidity – The system is hard to change because every change forces many other changes.
- ▶ Fragility – Changes cause the system to break in conceptually unrelated places.
- ▶ Immobility – It's hard to disentangle the system into reusable components.
- ▶ Viscosity – Doing things right is harder than doing things wrong.
- ▶ Opacity – It is hard to read and understand. It does not express its intent well.

Overall: *“The software starts to rot like a bad piece of meat”*



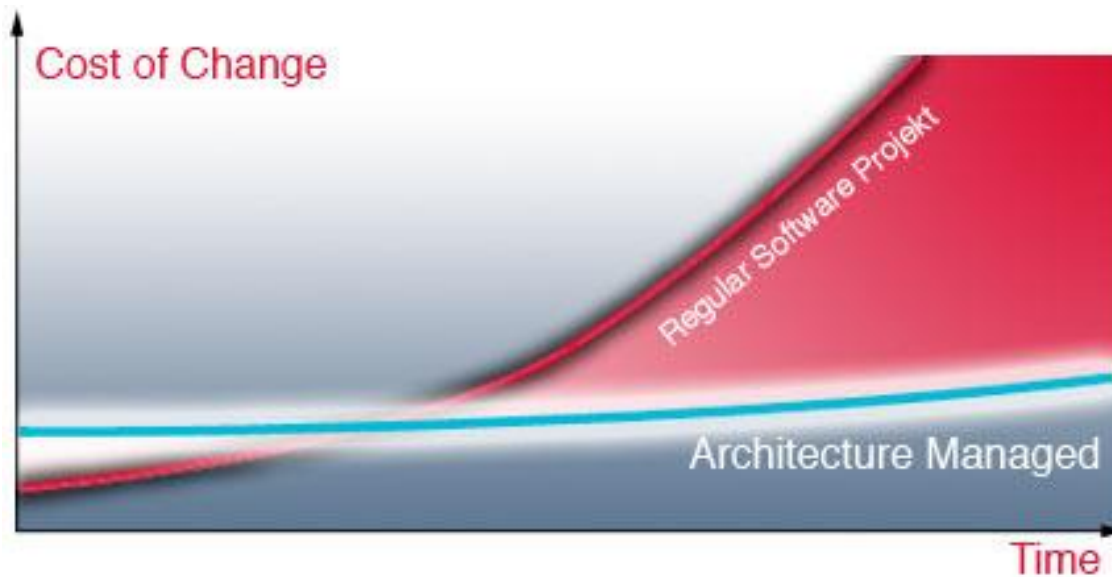
Was ist technische Qualität?

Unsere Definition: “Technical quality of software can be defined as the level of conformance of a software system to a set a set of rules and guidelines derived from common sense and best practices. Those rules should cover software architecture, programming in general, testing and coding style.”

- ▶ Technische Qualität lässt sich nicht allein durch Testen erreichen
- ▶ Technische Qualität manifestiert sich in jeder Codezeile
- ▶ Vier Aspekte technischer Qualität:
 - ▶ Architektur (inkl. Abhängigkeitsstruktur)
 - ▶ Software Metriken
 - ▶ Programmierregeln
 - ▶ Testbarkeit und Testabdeckung
- ▶ Welcher dieser Aspekte hat den größten Einfluss auf die Gesamtkosten?
- ▶ Messung der Qualität über Metriken und Zählung von Regelverletzungen



Kosten struktureller Erosion





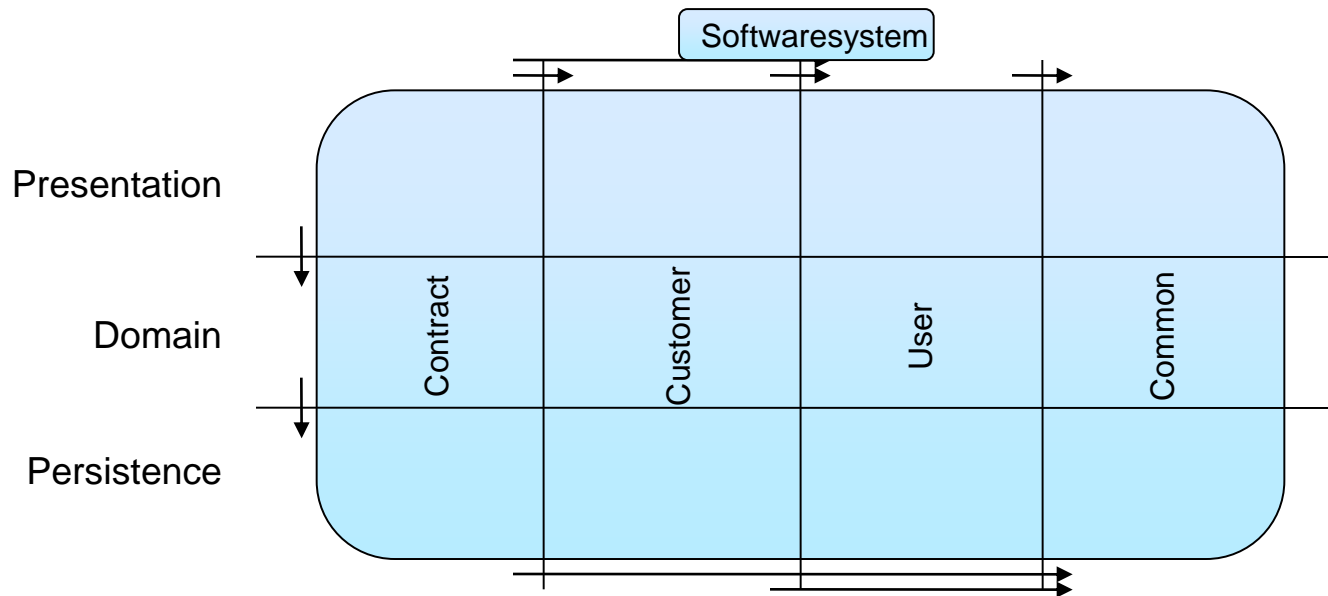
- ▶ Motivation
- ▶ Ursachen und Folgen der strukturellen Erosion
- ▶ Lösungsansätze
- ▶ Entwicklungsprozess



Architektur - Definition

- ▶ “The architecture represents the software structures that form the skeleton of the application.” [ASD]
- ▶ *Architecture is the fundamental organization of a system embodied in its **components**, their **relationships** to each other, and to the **environment**, and the **principles** guiding its design and evolution. [IEEE 1471]*

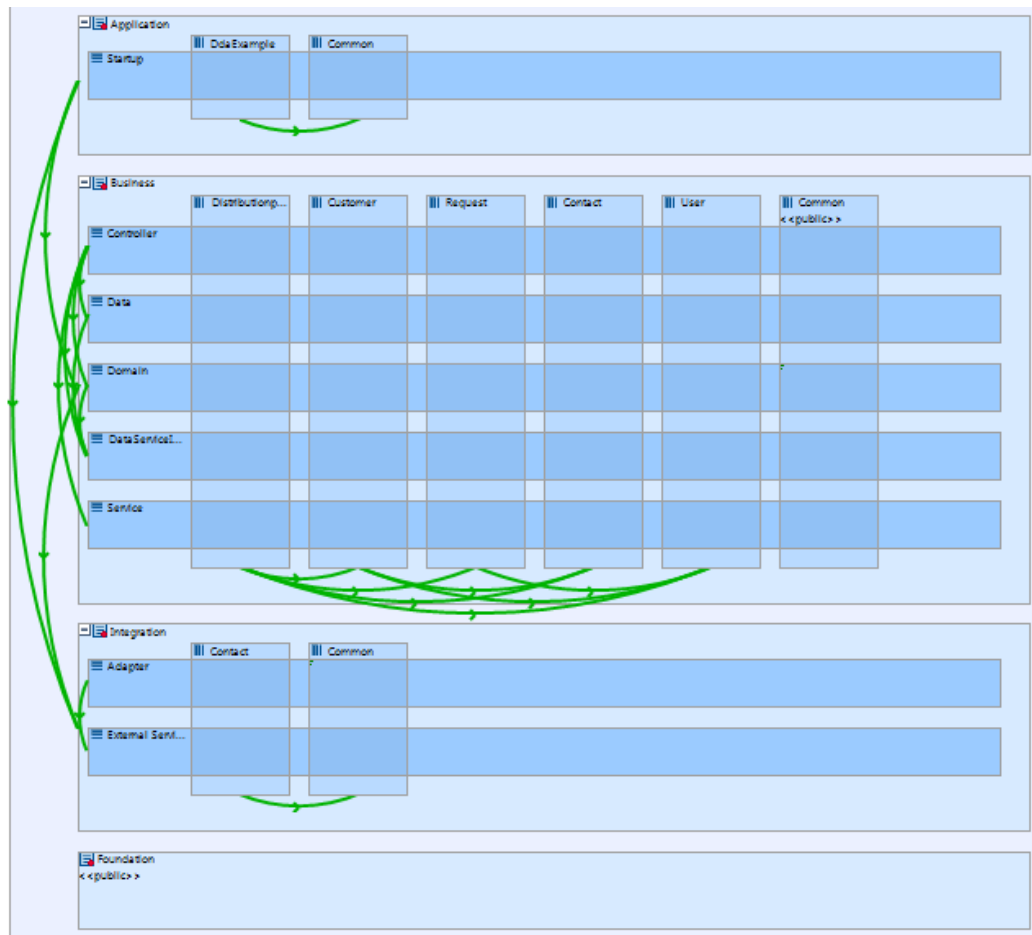
Erstellen einer Logischen Architektur



- Schritt 1: Teile horizontal in technische Aspekte
- Schritt 2: Teile vertikal in fachliche Aspekte
- Schritt 3: Definiere erlaubte Abhängigkeiten



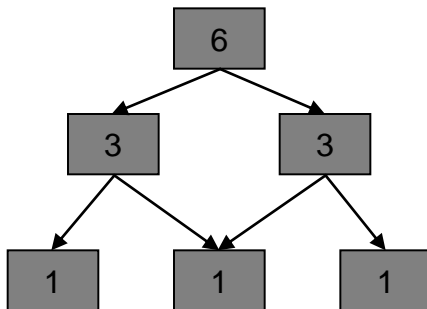
Etwas komplexere Logische Architektur



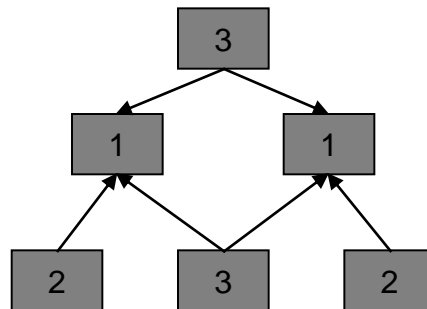


Metrik für den Kopplungsgrad [LSD]

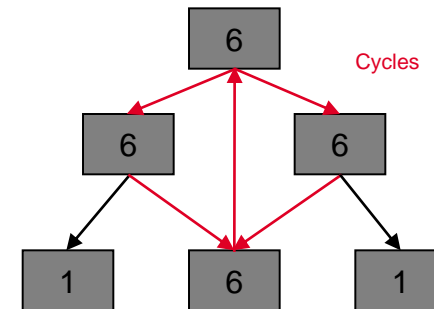
- ▶ Depends upon = Die Anzahl von Komponenten, von der eine Komponente direkt und indirekt abhängt (+1 für sich selbst).
- ▶ ACD (Average Component Dependency) = Die Summe aller “depends upon” Werte, geteilt durch die Anzahl aller Komponenten



$$\text{ACD} = 15/6 = 2,5$$



Dependency Inversion
 $\text{ACD} = 12/6 = 2$



$$\text{ACD} = 26/6 = 4,33$$



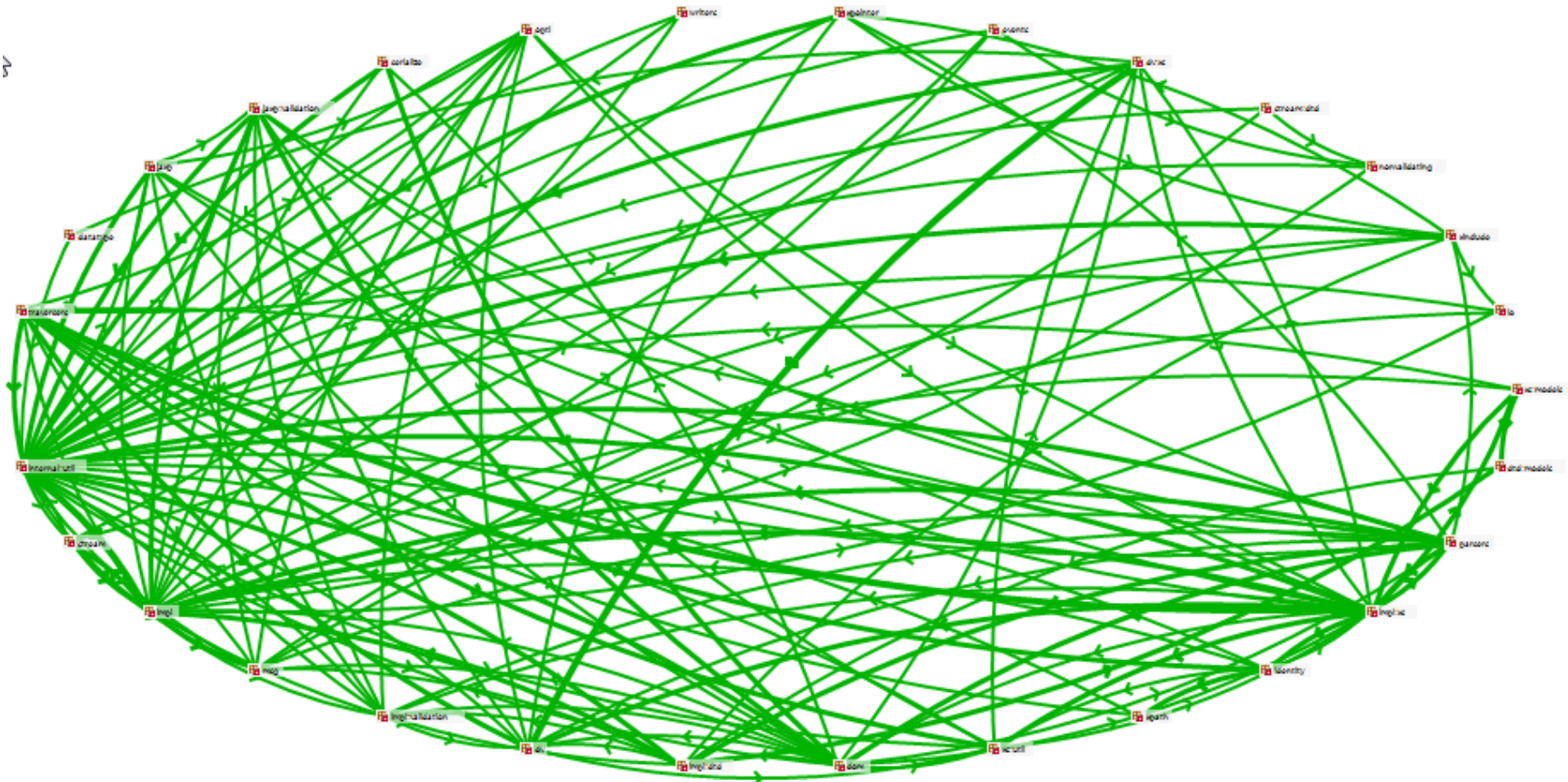
Der Effekt von zyklischen Abhängigkeiten

Zyklische Abhängigkeiten haben u.a. einen negativen Einfluss auf:

- ▶ Testbarkeit
- ▶ Verständlichkeit
- ▶ Wiederverwendung
- ▶ Erweiterbarkeit
- ▶ Build & Release Management
- ▶ Team building

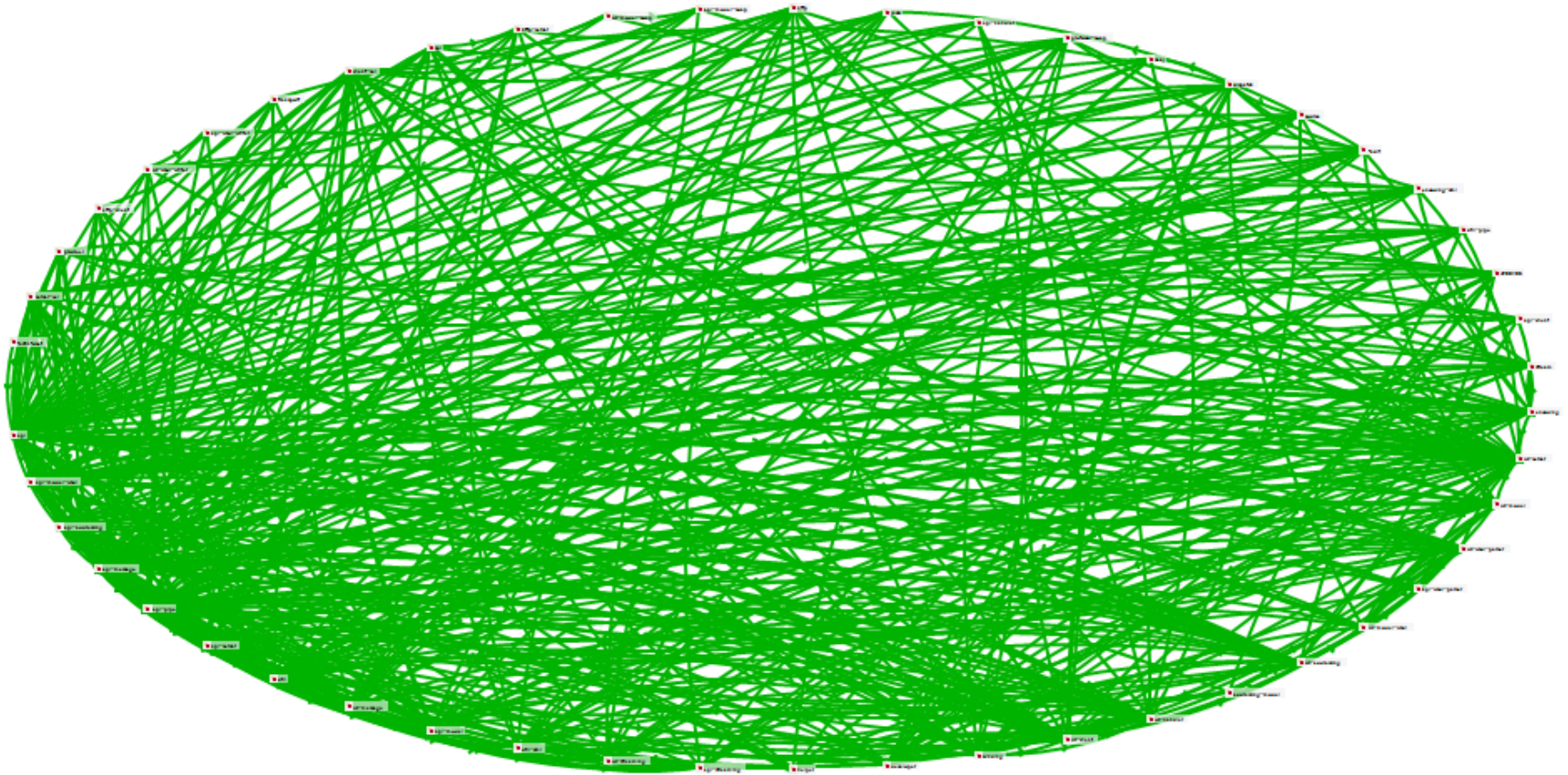
“Cyclic physical dependencies in large, low-level subsystems have the greatest capacity to increase the overall cost of maintaining a system“ [John Lakos in LSD]

Strukturelle Erosion am Bsp des JDK 1.6



Zyklengruppe von 29 Packages

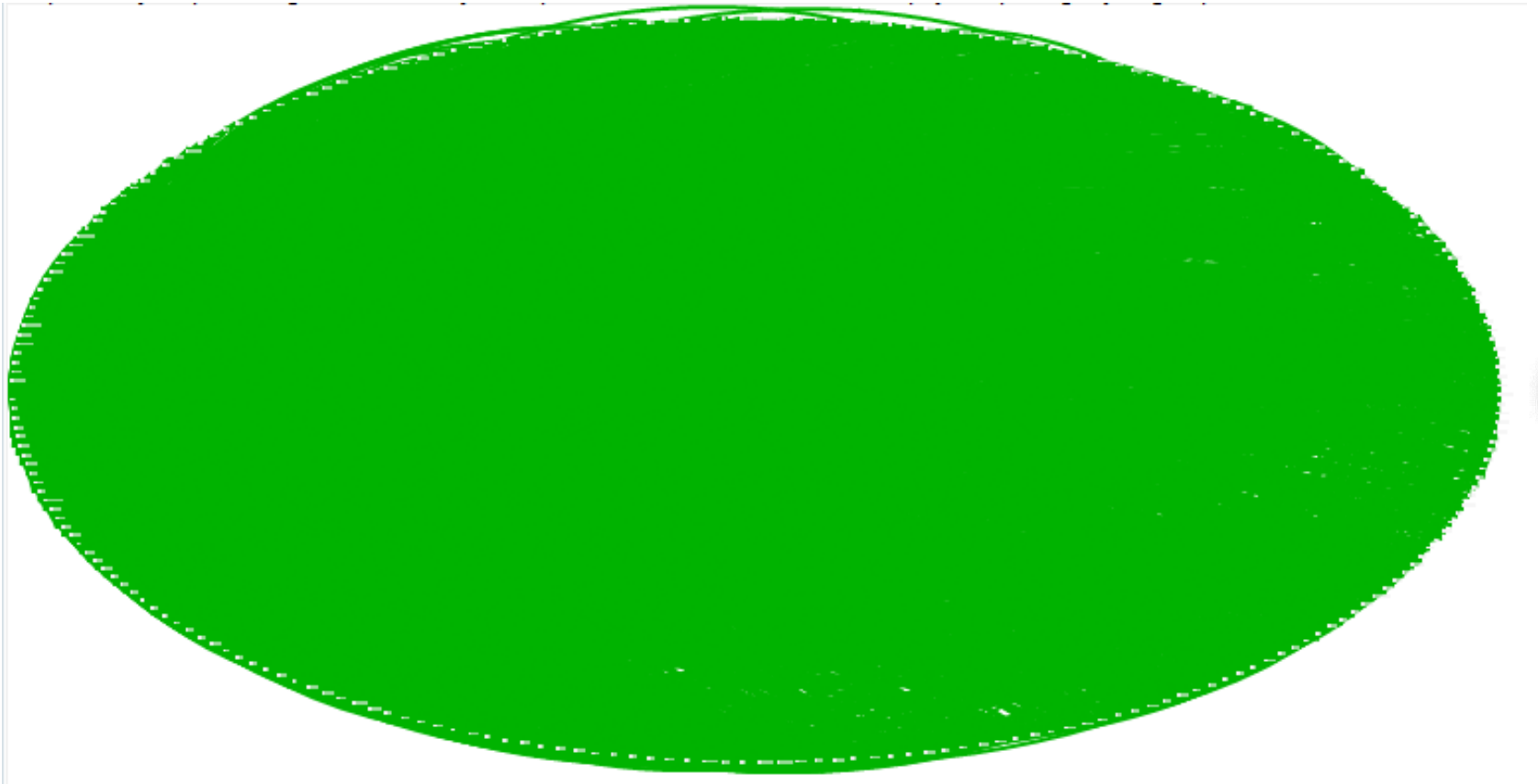
Strukturelle Erosion am Bsp des JDK 1.6



Zyklengruppe von 50 Packages



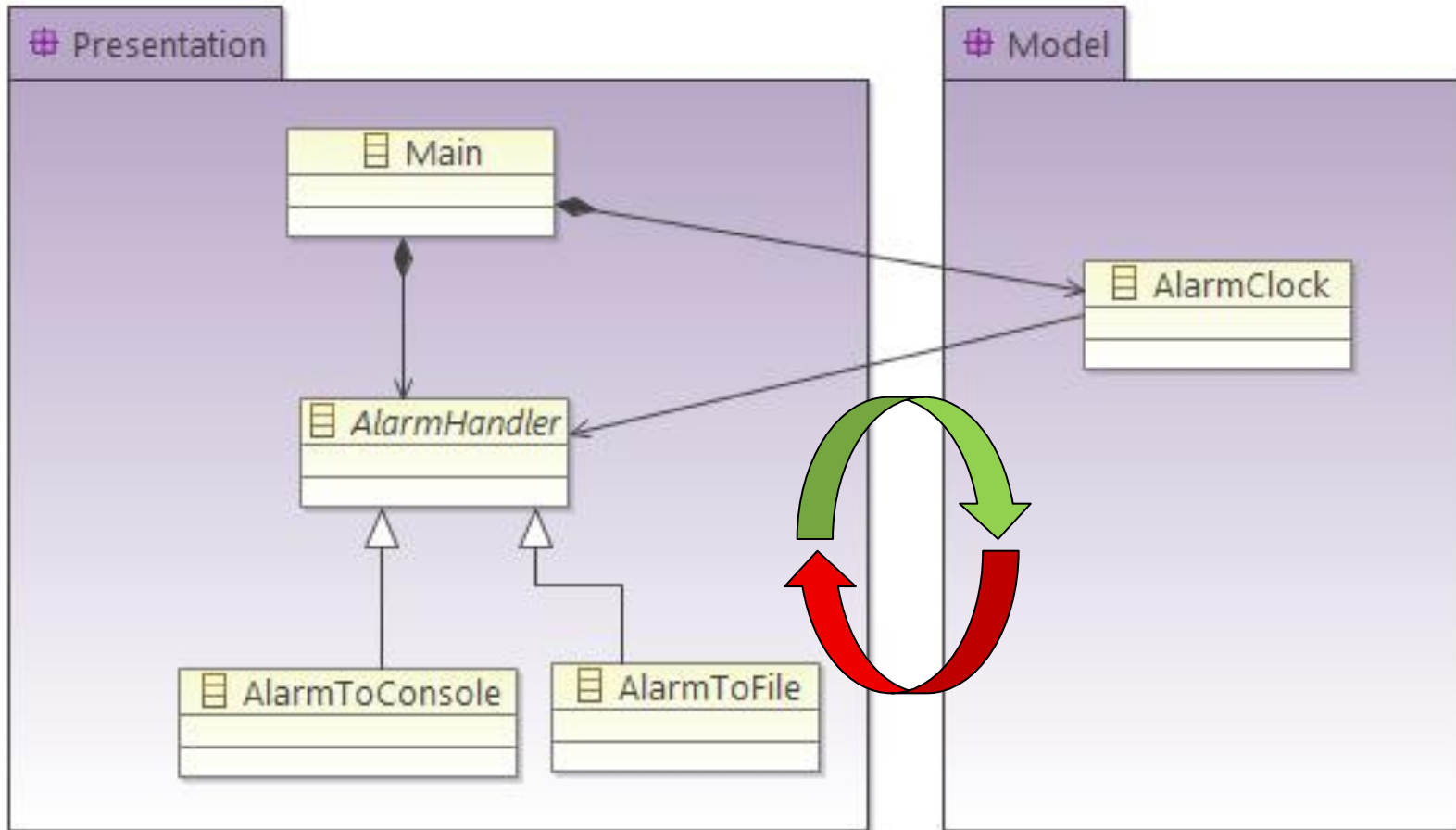
Strukturelle Erosion am Bsp des JDK 1.6



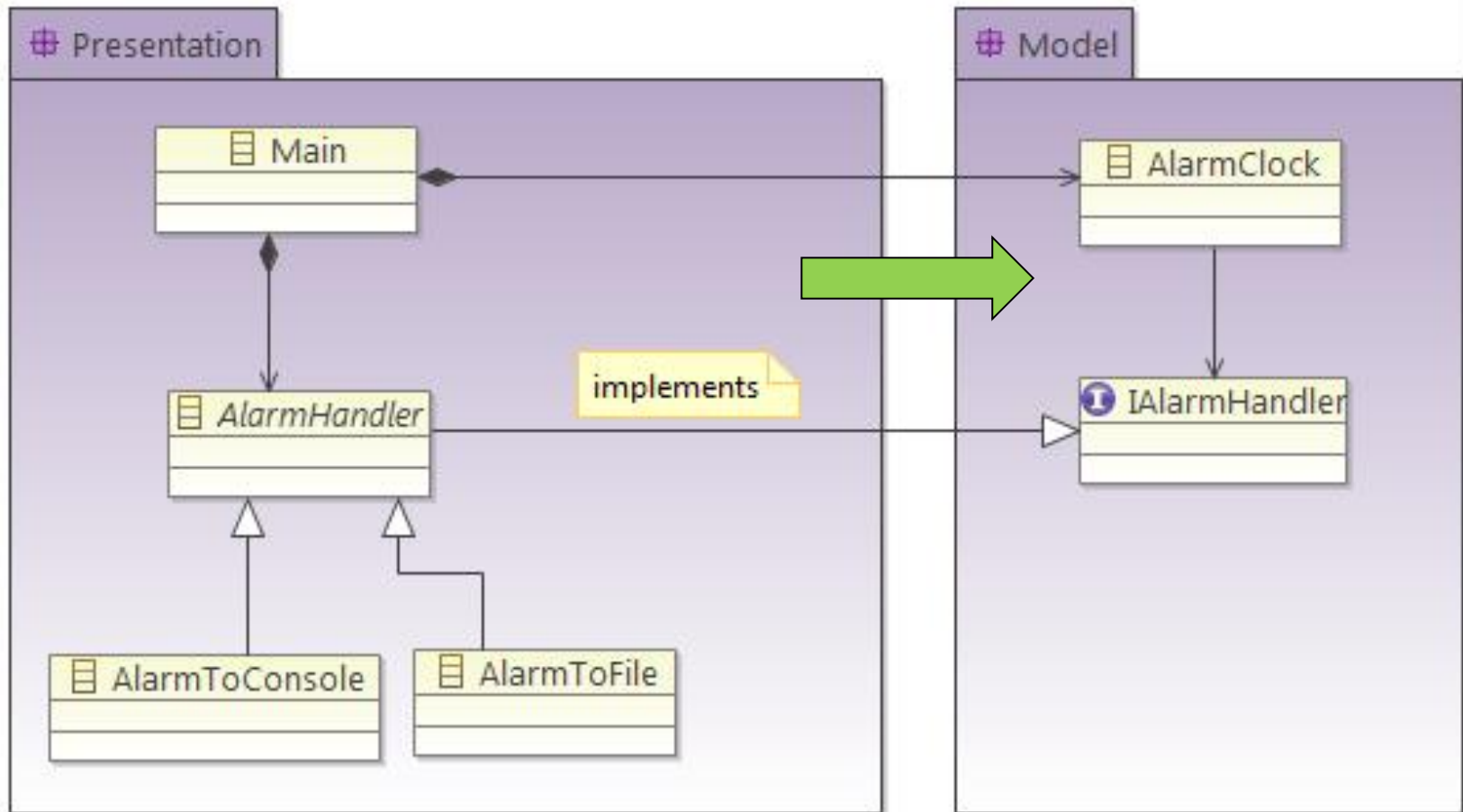
Zyklengruppe von 250 Packages



Bsp für das Auflösen eines Zyklus



Umkehrung der Abhängigkeit durch ein Callback Interface





Regeln für die Mikro Ebene

- ▶ Methoden implementieren Verhalten
 - ▶ Beschränkung der Komplexität (Cyclomatic Complexity)
 - ▶ Beschränkung von Parametern

- ▶ Klassen gruppieren Methoden
 - ▶ Zyklen sollten vermieden werden
 - ▶ Beschränkung der Größe (LOC, Anzahl Methoden)
 - ▶ Kopplungsgrad im Auge behalten

- ▶ Packages gruppieren Klassen
 - ▶ Zyklen sollten verboten sein
 - ▶ Beschränkung der Anzahl enthaltener Klassen



- ▶ Motivation
- ▶ Ursachen und Folgen der strukturellen Erosion
- ▶ Lösungsansätze
- ▶ Entwicklungsprozess



Prinzipien aus dem Agile Manifesto:

“[...] Through this work we have come to value:

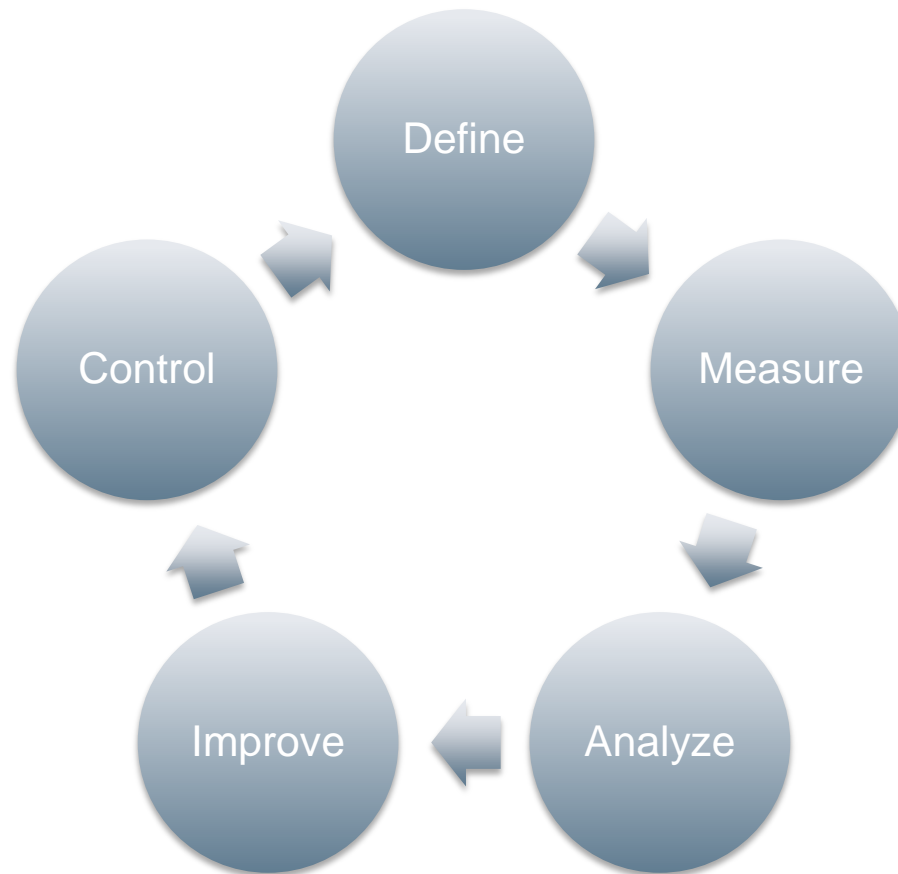
- ▶ **Individuals and interactions** over processes and tools
- ▶ **Working software** over comprehensive documentation
- ▶ **Customer collaboration** over contract negotiation
- ▶ **Responding to change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more.”

Trugschluss: Man braucht keine Prozesse, Werkzeuge, Architekturdefinitionen, etc.



Verbesserungen erfordern Transparenz





Architektur Workflow



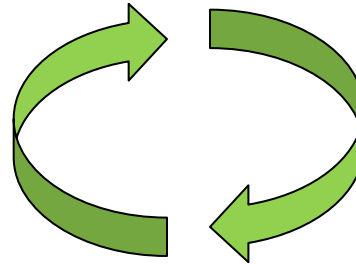
ARCHITECT

Definiert Architektur,
Schwellenwerte und
Aufgaben



DEVELOPERS

Implementieren Use
Cases und Aufgaben
unter Beachtung der
Architektur und
Schwellenwerten

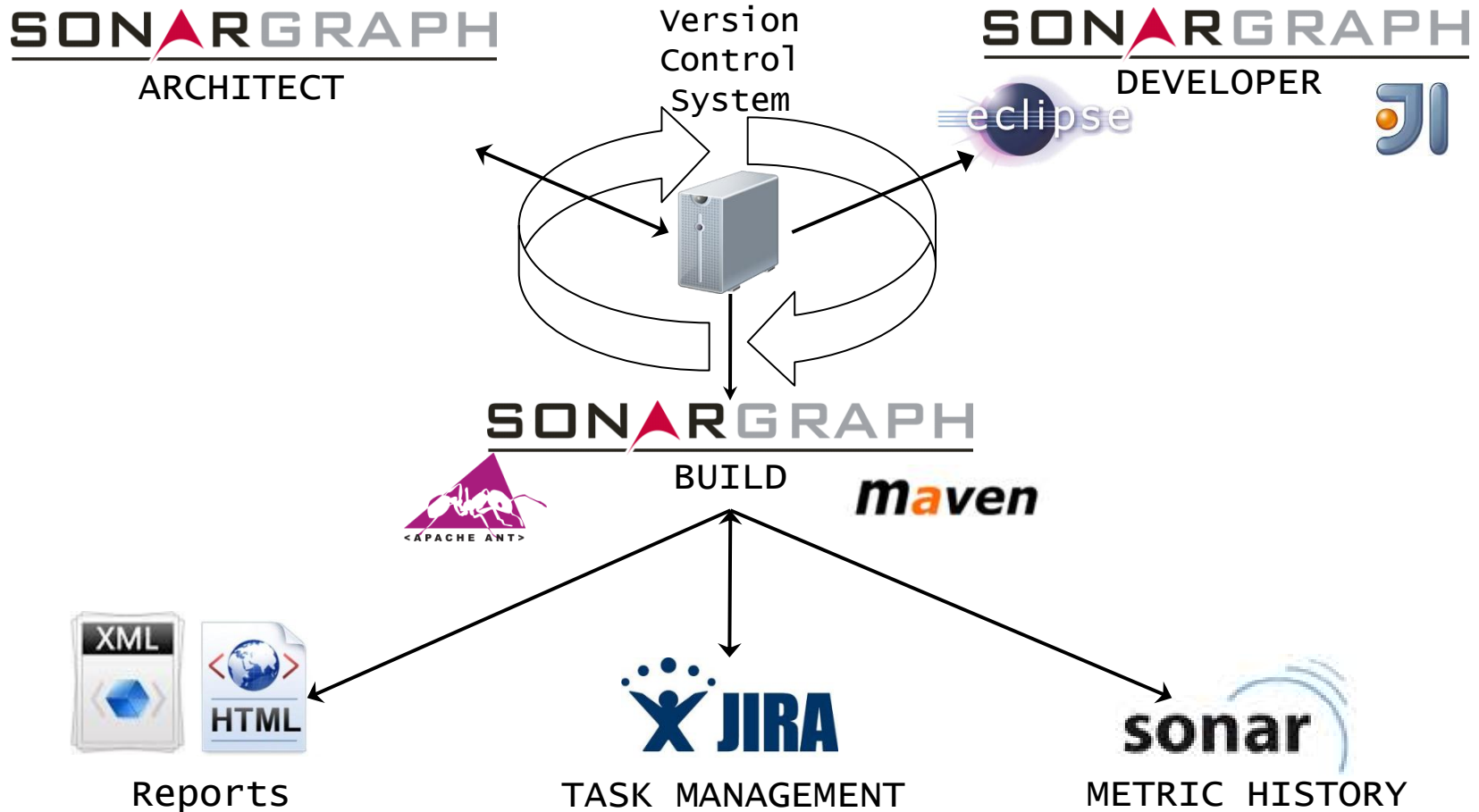


BUILD Server

Überprüft die
Einhaltung der Regeln



Architecture Workflow with Sonargraph





Take away: Wenige Regeln können viel bewirken

- ▶ Definition einer Zyklen-freien logische Architektur und einer konsistente Package Namenskonvention. Alle Packages müssen logischen Architekturelementen zugeordnet werden.
- ▶ Keine Zyklen zwischen Packages
- ▶ Kontrolle des Kopplungsgrades (ACD und NCCD mit vernünftigen Schwellenwerten)
- ▶ Beschränkung der Größe von Java Sourcen (700 Zeilen)
- ▶ Beschränkung der Zyklomatischen Komplexität von Methoden (z.B. 20)
- ▶ Regelmäßige, automatische Überprüfung der Regeln

Qualität muss als Zielvorgabe von allen Managementebenen mitgetragen werden!



Weitere Informationen

- ▶ Whitepaper, DZone RefCard, etc. auf unserer Web-Seite: <http://www.hello2morrow.com>
- ▶ Meine e-mail: i.kellner@hello2morrow.de

Referenzen

- ▶ [MMM] The Mythical Man-Month, F. P. Brooks, Addison-Wesley, 1975, 1995
- ▶ [GOF] Design Patterns, Gamma et al., Addison-Wesley 1994
- ▶ [LSD] Large-Scale C++ Software Design, John Lakos, Addison-Wesley 1996
- ▶ [EXP] Extreme Programming, Kent Beck, Addison-Wesley 1999
- ▶ [AUP] Applying UML and Patterns, Craig Larman, Prentice Hall 2000
- ▶ [TOS] Testing Object-Oriented Systems, Beizer, Addison-Wesley 2000
- ▶ [ASD] Agile Software Development, Robert C. Martin, Prentice Hall 2003