

Software-Projekt

Prof. Dr. Rainer Koschke

Fachbereich Mathematik und Informatik
Arbeitsgruppe Softwaretechnik
Universität Bremen

Wintersemester 2007/08

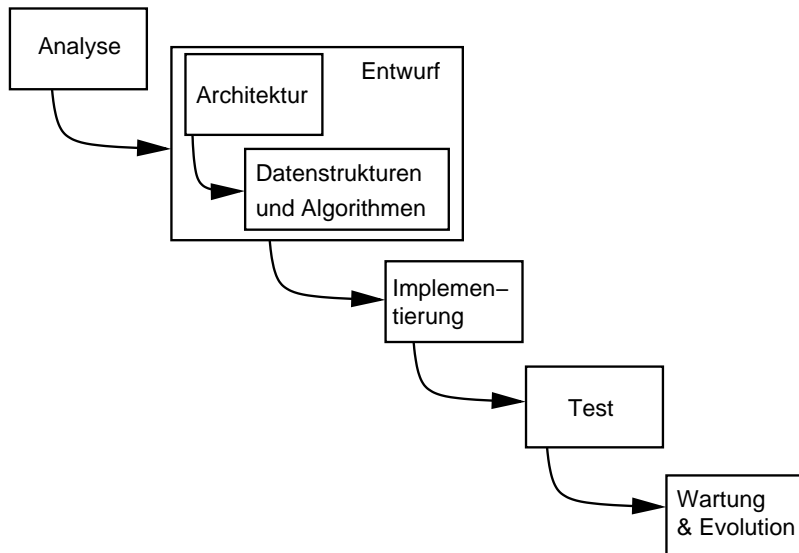
Überblick I

- 1 Entwurfsmuster

- 1 Entwurfsmuster
 - Was ist ein Entwurfsmuster?
 - Bestandteile eines Entwurfsmusters
 - Kategorien von Entwurfsmustern
 - Entwurfsmuster Factory Method
 - Entwurfsmuster Observer
 - Wiederholungsfragen

- Verstehen, was Entwurfsmuster sind
- Qualitäten und Einsetzbarkeit der Entwurfsmuster kennen

Kontext



Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.

Christopher Alexander (Architekt und Mathematiker),
“A pattern language”, 1977.

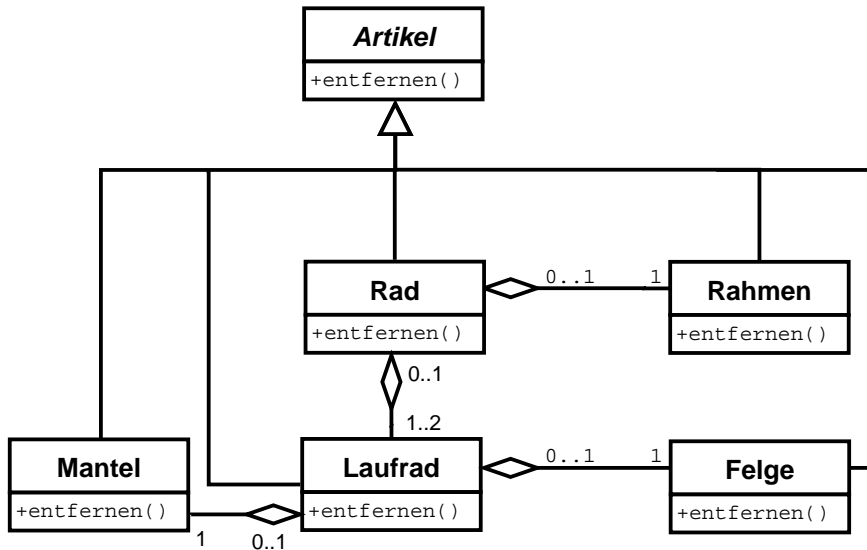
Definition

Entwurfsmuster: „Musterlösung“ für ein wiederkehrendes Entwurfsproblem.

Bestandteile eines Entwurfsmusters

- **Name** (kurz und beschreibend)
- **Problem**: Was das das Entwurfsmuster löst
- **Lösung**: Wie es das Problem löst
- **Konsequenzen**: Folgen und Kompromisse des Musters.

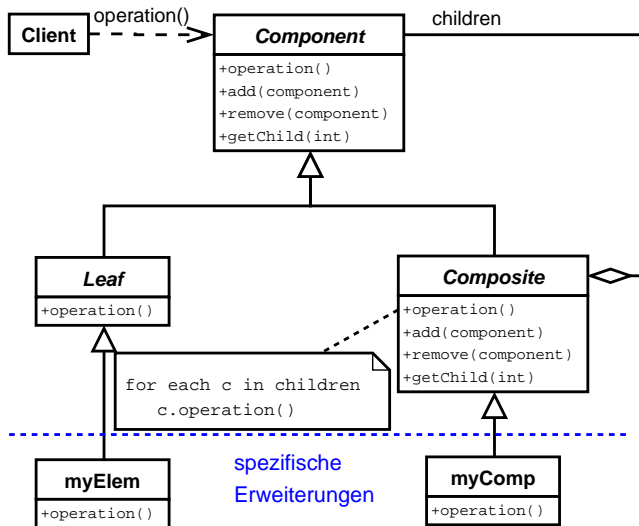
Beispielentwurfsproblem



- **Name:** *Composite*
- **Zweck:** Teil-von-Hierarchie mit einheitlicher Schnittstelle beschreiben (überall wo ein Ganzes benutzt werden kann, kann auch ein Teil benutzt werden und umgekehrt)
- **Motivation:** ... Einführung anhand eines konkreten Beispiels. ...
- **Anwendbarkeit:**
 - wenn Teil-von-Beziehung beschrieben werden soll
 - uniforme Schnittstelle für alle Elemente der Hierarchie

Beschreibung von Mustern (Gamma u. a. 2003) II

- Struktur:**



- **Teilnehmer:**

- *Component:*

- deklariert einheitliche Schnittstelle
 - implementiert Standardverhalten
 - deklariert Schnittstelle, um seine Teile zu verwalten
 - (optional) definiert Schnittstelle, um Behälter zu erhalten

- *Leaf:*

- repräsentiert atomare Komponente
 - definiert Verhalten für atomare Komponenten

- *Composite:*

- definiert Verhalten für zusammengesetzte Komponenten
 - speichert Teile
 - implementiert Operationen zur Verwaltung von Teilen

- *Client:*

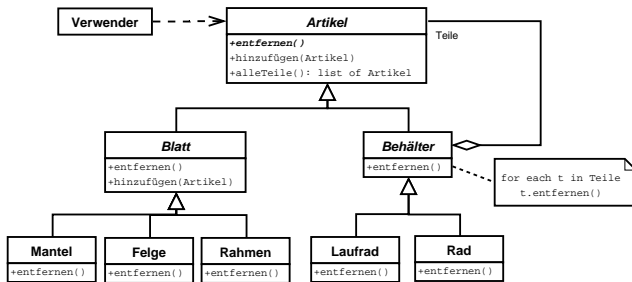
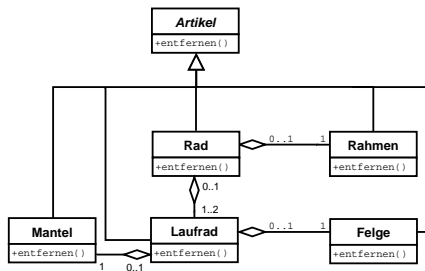
- manipuliert Objekte der Komponenten nur durch die Schnittstelle von *Composite*

- **Kollaborationen:**

- *Clients* benutzen *Component*-Schnittstelle
- falls Empfänger ein *Leaf* ist, antwortet es direkt
- falls Empfänger ein *Composite* ist, wird die Anfrage an Teile weitergeleitet (möglicherweise mit weiteren eigenen Operationen vor und/oder nach der Weiterleitung)

- **Konsequenzen:**
 - zweiteilt die Klassenhierarchie in *Leaf* und *Composite* mit einheitlicher Schnittstelle
 - uniforme Verwendung auf Seiten des *Clients*
 - neue Komponenten können leicht hinzugefügt werden
 - könnte die Struktur unnötig allgemein machen (dynamische versus statische Typisierung)
- **Implementierung:** ... Hinweise zur Implementierung ...

Dynamische versus statische Typisierung



Kategorien von Entwurfsmustern

- Erzeugungsmuster
 - betreffen die Erzeugung von Objekten
 - Beispiel: Factory Method
- Strukturelle Muster:
 - betreffen Komposition von Klassen und Objekten
 - Beispiel: Composite
- Verhaltensmuster:
 - betreffen Interaktion und Verantwortlichkeiten
 - Beispiel: Observer

Erweiterung für andere Domänen

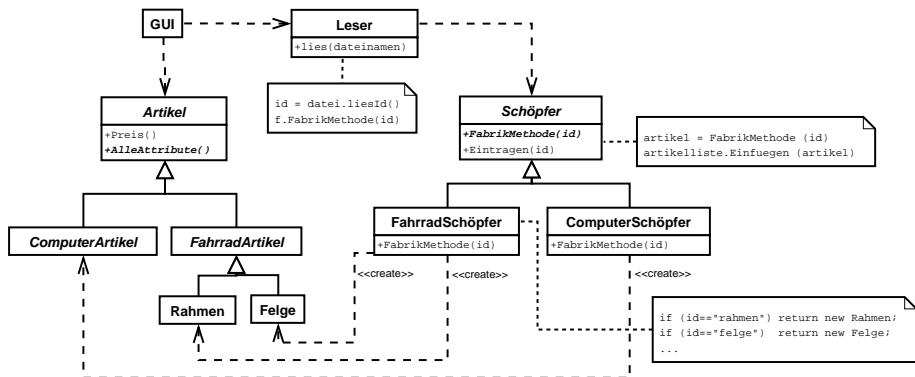
Anforderungen:

- Artikeldaten sollen von einer Datei gelesen werden können
- zukünftig sollen andere Domänen unterstützen (Fahrrad, Computer und Klettern)
- die Objekte dieser Domänen sind unterschiedlich
- notwendige Anpassungen sollen einfach realisiert werden können

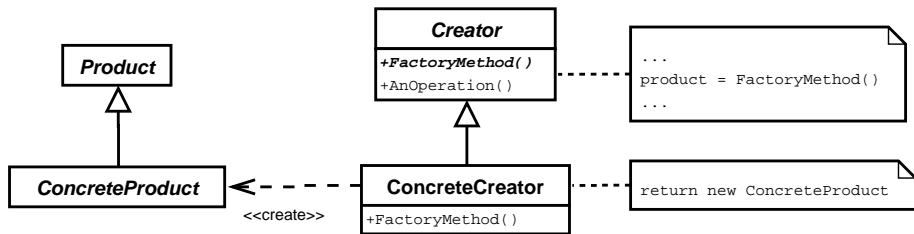
Lösungsstrategien:

- die Klassen der Benutzungsschnittstelle beziehen sich nur auf die Schnittstelle der abstrakten Klasse Artikel
- Datei hat gleiche Syntax für alle Domänen (nur die Inhalte variieren)
- die Artikel werden beim Einlesen der Datei als Objekte erzeugt
 - aber der Leser muss doch die Konstruktoren der Objekte kennen, oder was?

Lösungsstrategie



Entwurfsmuster: *Factory Method*



- eine Klasse weiß in manchen Fällen nicht im Voraus, von welcher Klasse ein zu erzeugendes Objekt sein soll
- die konkreten Unterklassen einer Klasse sollen dies entscheiden
- Verantwortlichkeit wird an Unterklassen delegiert, und das Wissen über die Unterklasse, an die delegiert wird, soll nur an einem Punkt vorhanden sein

- Product
 - deklariert die Schnittstelle von Objekten, die die Fabrikmethode erschafft
- ConcreteProduct
 - implementiert die Product-Schnittstelle
- Creator
 - deklariert die Fabrikmethode
 - (optional) implementiert eine Standardfabrikmethode, die ein spezifisches konkretes Objekt erzeugt
 - kann Fabrikmethode aufrufen, um ein Product-Objekt zu erzeugen
- ConcreteCreator
 - überschreibt die Fabrikmethode, um konkretes Product-Objekt zu erschaffen

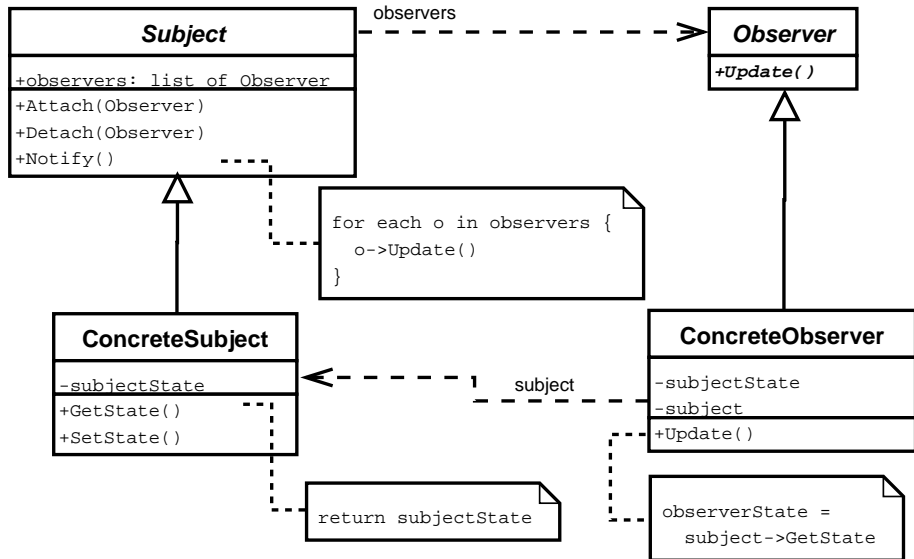
Anwendbarkeit

- Komponenten hängen von anderen Komponenten ab
- Änderung der einen Komponente muss Änderung der anderen nach sich ziehen
- Komponenten sollen lose gekoppelt sein: Komponente kennt seine Abhängigen nicht im Voraus (zur Übersetzungszeit)

Lösungsstrategie

- Abhängige registrieren sich bei Komponente
- Komponente informiert alle registrierten Abhängigen über Zustandsänderung

Entwurfsmuster Observer: Struktur



2008-01-14

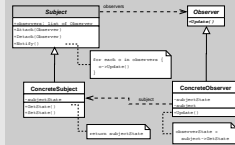
Software-Projekt

└─ Entwurfsmuster

└─ Entwurfsmuster Observer

└─ Entwurfsmuster Observer: Struktur

Entwurfsmuster Observer: Struktur



Man beachte die Beziehung zu den Architekturmustern Blackboard und Model-View-Controller.

Entwurfsmuster Observer: Teilnehmer

- Subject
 - kennt seine Observer (zur Laufzeit)
 - kann beliebig viele Observer haben
 - stellt Schnittstelle zur Verfügung, um Observer zu registrieren und abzutrennen
- Observer
 - deklariert Schnittstelle für die Update-Nachricht
- ConcreteSubject
 - hat einen Zustand, der ConcreteObserver interessiert
 - sendet Bekanntmachung via Notify(), wenn sich Zustand ändert (SetState)
- ConcreteObserver
 - kennt ConcreteSubject-Objekt
 - verarbeitet Zustand dieses Subjects
 - implementiert Update, um auf veränderten Zustand zu reagieren

- abstrakte Kopplung zwischen Subject und Observer
- unterstützt Rundfunk (Broadcast)
- unerwartete Updates, komplizierter Kontrollfluss
- viel Nachrichtenverkehr, auch dann wenn sich ein irrelevanter Aspekt geändert hat

Entwurfsmuster Observer: Verfeinerungen

- Push-Modell
 - Subject sendet detaillierte Beschreibung der Änderung
 - umfangreiches Update
 - vermeidet GetState(), aber nicht Update()
- Pull-Modell
 - Subject sendet minimale Beschreibung der Änderung
 - Observer fragt gegebenenfalls die Details nach
 - erfordert weitere Nachrichten, um Details abzufragen
- Explizite Interessen
 - Observers melden Interesse an spezifischem Aspekt an; Aspekt wird zusätzlicher Parameter von Update

- Was ist ein Entwurfsmuster?
- Warum sind sie interessant für die Software-Entwicklung?
- Erläutern Sie eines der in der Vorlesung vorgestellten Entwurfsmuster.

- Das Standardbuch zu Entwurfsmustern ist das von Gamma u. a. (2003)

- 1 Basili und Turner 1975** BASILI, V. ; TURNER, J.: Iterative Enhancement: A Practical Technique for Software Development. In: IEEE Transactions on Software Engineering 1 (1975), Dezember, Nr. 4, S. 390–396
- 2 Bass u. a. 2003** BASS, Len ; CLEMENTS, Paul ; KAZMAN, Rick: Software Architecture in Practice. 2nd edition. Addison Wesley, 2003
- 3 Berling und Runeson 2003** BERLING, T. ; RUNESON, P.: Evaluation of a perspective based review method applied in an industrial setting. In: IEE Proceedings 150 (2003), Juni, Nr. 3
- 4 Boehm 2000** BOEHM, Barry: Project Termination Doesn't Equal Project Failure. In: IEEE Computer (2000), September, S. 94–96
- 5 Boehm 1979** BOEHM, Barry W.: Guidelines for verifying and validating software requirements and design specifications. In: EURO IFIP 79, IFIP, 1979, S. 711–719
- 6 Boehm 1988** BOEHM, Barry W.: A spiral model of software development and enhancement. In: IEEE Computer 21 (1988), Mai, Nr. 5, S. 61–72

- 7 **Brügge und Dutoit 2004** BRÜGGE, Bernd ; DUTOIT, Allen H.:
Objektorientierte Softwaretechnik. Prentice Hall, 2004
- 8 **Buschermöhle u. a. 2006** BUSCHERMÖHLE, Ralf ; EEKHOFF, Heike ;
JOSKO, Bernhard: Success – Erfolgs- und Misserfolgskfaktoren bei der
Durchföhrung von Hard- und Softwareentwicklungsprojekten in
Deutschland. www.offis.de/umfragesuccess. 2006
- 9 **Buschmann u. a. 1996** BUSCHMANN, Frank ; MEUNIER, Regine ;
ROHNERT, Hans ; SOMMERLAD, Peter ; STAL, Michael:
Pattern-oriented Software Architecture: A System of Patterns. Bd. 1.
Wiley, 1996
- 0 **EN ISO 9241-10:1995 1995** : Ergonomische Anforderungen für
Bürotätigkeiten mit Bildschirmgeräten – Teil 10: Grundsätze der
Dialoggestaltung. Europäische Norm, ISO-Standard. 1995
- 1 **Fjeldstadt und Hamlen 1984** FJELDSTADT ; HAMLEN: Application
Program Maintenance Study: Report to Our Respondents. In: Proc.
GUIDE 48, IEEE Computer Society Press, April 1984

- 2 **Frühauf u. a. 2000** FRÜHAUF ; LUDEWIG ; SANDMAYR:
Software-Prüfung. 4. Auflage. vdf Zürich, 2000
- 3 **Fusaro u. a. 1997** FUSARO, P. ; LUNIBILE, F. ; VISAGGIO, G.: A replicated experiment to assess requirements inspection techniques. In: Empirical Software Engineering 2 (1997), S. 39–57
- 4 **Gamma u. a. 2003** GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John: Desig Patterns—Elements of Reusable Object-Oriented Software. Addison Wesley, 2003
- 5 **Gilb 1988** GILB, Tom: Principles of Software Engineering Management. Harlow, UK : Addison-Wesley, 1988
- 6 **Gornik 2001** GORNIK, David: IBM Rational Unified Process: Best Practices for Software Development Teams / IBM Rational Software. 2001 (TP026B, Rev 11/01). – White Paper
- 7 **Harel 1987** HAREL, David: State Charts: a Visual Formalism for Complex Systems. In: Science of Computer Programming 8 (1987), Nr. 3, S. 231–274

- 8 **Hayes 1997** HAYES, Frank: Managing User Expectations. In: Computerworld (1997), November
- 9 **Hayes 1999** HAYES, W.: Research synthesis in software engineering: a case for meta-analysis. In: Proc. 6th Int. Symp. on Software Metrics, IEEE Computer Society Press, November 1999, S. 143–151
- 0 **Hofmeister u. a. 2000** HOFMEISTER, Christine ; NORD, Robert ; SONI, Dilip: Applied Software Architecture. Addison Wesley, 2000 (Object Technology Series)
- 1 **IEEE P1471 2002** : IEEE Recommended Practice for Architectural Description of Software-intensive Systems—Std. 1471-2000. ISBN 0-7381-2518-0, IEEE, New York, NY, USA. 2002
- 2 **IEEE Standard 830.1998 1998** : IEEE Recommended Practice for Software Requirements Specifications, IEEE Standard 830.1998. 1998
- 3 **IEEE-Std-1058 1987** : ANSI/IEEE Standard for Software Project Management Plans. ANSI/IEEE Std. 1058.1-1987. 1987
- 4 **IEEE Std 610.12-1990 1990** : IEEE Standard Glossary of Software Engineering Terminology. IEEE Standard 610.12-1990. 1990

- 5 **IEEE Std. 829-1998 1998** : IEEE Standard for Software Test Documentation. IEEE Standards Board, IEEE Std. 829-1998. 1998
- 6 **IEEE Std. 982-1989 1989** : IEEE Standard Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software. IEEE Standards Board, IEEE Std. 982-1989. Juli 1989
- 7 **ISO 9241-11:1998 1998** : Ergonomic requirements for office work with visual display terminals (VDTs) – Part 11: Guidance on usability. ISO-Standard. 1998
- 8 **Kazman u. a. 1996** KAZMAN, Rick ; ABOWD, Gregory ; BASS, Len ; CLEMENTS, Paul: Scenario-Based Analysis of Software Architecture. In: IEEE Software (1996), November, S. 47–55
- 9 **Kruchten 1998** KRUCHTEN, Phillipe: The Rational Unified Process: An Introduction. Reading, Mass.: Addison-Wesley, 1998
- 0 **Lehman und Belady 1985** LEHMAN, M. ; BELADY, L.: Program Evolution: Processes of Software Change. London : Academic Press, 1985

- 1 **Liskov 1988** LISKOV, Barbara: Data Abstraction and Hierarchy. In: SIGPLAN Notices 23 (1988), Mai, Nr. 5
- 2 **Liskov und Wing 1994** LISKOV, Barbara ; WING, Jeanette M.: A Behavioral Notion of Subtyping. In: ACM Transactions on Programming Languages and Systems 16 (1994), Nr. 6, S. 1811–1841
- 3 **Ludewig 2003** LUDEWIG, Jochen: Einführung in die Softwaretechnik. Vorlesungs-Skriptum. 2003
- 4 **Ludewig und Lichter 2006** LUDEWIG, Jochen ; LICHTER, Horst: Software Engineering – Grundlagen, Menschen, Prozesse, Techniken. dpunkt.verlag, 2006
- 5 **Maaß 2001** MAASS, Susanne: Soziotechnische Systeme. Vorlesungs-Skriptum. 2001
- 6 **McKee 1984** MCKEE, J.R.: Maintenance as a Function of Design. In: AFIPS National Computer Conference, Las Vegas (Ch. 27), 1984
- 7 **Miller u. a. 1998** MILLER, J. ; WOOD, M. ; ROPER, M.: Further experiences with scenarios and checklists. In: Empirical Software Engineering 3 (1998), S. 37–64

- 8 Murphy u. a. 1995** MURPHY, Gail C. ; NOTKIN, David ; SULLIVAN, Kevin: Software Reflexion Models: Bridging the Gap Between Source and High-Level Models. In: Proc. of the Third ACM SIGSOFT Symposium on the Foundations of Software Engineering, 1995, S. 18–28
- 9 Murphy u. a. 2001** MURPHY, Gail C. ; NOTKIN, David ; SULLIVAN, Kevin J.: Software Reflexion Models: Bridging the Gap between Design and Implementation. In: IEEE Transactions on Software Engineering 27 (2001), April, Nr. 4, S. 364–380
- 0 Neumann 1999** NEUMANN, Peter G.: Risks to the Public in Computers and Related Systems. In: ACM SIGSOFT Software Engineering Notes 24 (1999), Nr. 1, S. 31
- 1 Object Management Group 2003** OBJECT MANAGEMENT GROUP: OMG Unified Modeling Language Specification. March 2003. – Version 1.5
- 2 OMG** OBJECT MANAGEMENT GROUP (OMG): Unified Modeling Language. <http://www.uml.org>

- 3 Parnas 1972** PARNAS, David L.: On the Criteria to Be Used in Decomposing Systems into Modules. In: Communications of the ACM 15 (1972), Dezember, Nr. 12
- 4 Partsch 1998** PARTSCH, H.: Requirements-Engineering systematisch - Modellierung für softwaregestützte Systeme. Berlin, Heidelberg, New York : Springer-Verlag, 1998
- 5 P.C.Lockemann und (Hrsg) 1987** P.C.LOCKEMANN (Hrsg.) ; (HRSG), J.W.Schmidt (Hrsg.): Datenbank-Handbuch. Springer, 1987
- 6 Porter und Votta 1998** PORTER, A. ; VOTTA, L.: Comparing detection methods for software requirements inspection: a replication using professional subjects. In: Empirical Software Engineering 3 (1998), S. 355–379
- 7 Porter u. a. 1995** PORTER, A. ; VOTTA, L. ; BASILI, V.: Comparing detection methods for software requirements inspection: a replicated experiment. In: IEEE Transactions on Software Engineering 21 (1995), Nr. 5, S. 563–575

- 8 **Pressman 2003** PRESSMAN, Roger: Software Engineering – A Practitioner’s Approach. Fünfte Ausgabe. McGraw-Hill, 2003
- 9 **Royce 1970** ROYCE, W.: Managing the Development of Large Software Systems. In: Proceedings Westcon, IEEE Computer Society Press, 1970, S. 328–339
- 0 **Rödiger 2004** RÖDIGER, Karl-Heinz: Vortrag Rechtlicher Rahmen der Software-Entwicklung in der Vorlesung Software-Projekt. Vorlesungs-Skriptum. 2004
- 1 **Sandahl u. a. 1998** SANDAHL, K. ; BLOMKVIST, O. ; KARLSSON, J. ; KRYSANDER, C. ; LINDVALL, M. ; OHLSSON, N.: An extended replication of an experiment for assessing methods for software requirements inspections. In: Empirical Software Engineering 3 (1998), S. 327–354
- 2 **Sauer und Cuthbertson 2003** SAUER, C. ; CUTHBERTSON, C.: The State of IT Project Management in the UK. <http://www.cw360ms.com/pmsurveyresults/surveyresults>. 2003

- 3 **Shaw und Garlan 1996** SHAW, Mary ; GARLAN, David: Software Architecture – Perspectives on an Emerging Discipline. Upper Saddle River, NJ : Prentice Hall, 1996. – ISBN ISBN 0-13-182957-2
- 4 **Shull u. a. 2000** SHULL, Forrest ; RUS, Ioana ; BASILI, Victor: How Perspective-Based Reading Can Improve Requirements Inspections. In: IEEE Computer 33 (2000), Nr. 7, S. 73–79
- 5 **Sneed 2004** SNEED, Harry: Vorlesung Software-Engineering. Vorlesungsskriptum, Wirtschaftsinformatik, Uni Regensburg. 2004
- 6 **Sommerville 2004** SOMMERVILLE, Ian: Software Engineering. Siebte Ausgabe. Addison-Wesley, 2004
- 7 **Standish Group 1994**
- 8 **Standish Group 2004** STANDISH GROUP: Third Quarter Research Report. <http://www.standishgroup.com>. 2004
- 9 **Stimely 1990** STIMELY, Gwen L.: A stepwise approach to developing software documentation. In: ACM SIGDOC Asterisk Journal of Computer Documentation 14 (1990), Nr. 4, S. 121–124

- 0 **Störrle 2005** STÖRRLE, Harald: UML 2 für Studenten. Pearson Studium, 2005. – ISBN 3-8273-7143-0
- 1 **Sun microsystems 1999** SUN MICROSYSTEMS: Code Conventions for the Java™ Programming Language. April 1999. – URL <http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html>
- 2 **Winter 2003** WINTER, Andreas: Einführung in die Softwaretechnik. Vorlesungs-Skriptum. 2003
- 3 **Zuser u. a. 2004** ZUSER, W. ; GRECHENIG, T. ; KÖHLE, M.: Software Engineering mit UML und dem Unified Process. Zweite Ausgabe. Pearson Studium, 2004