Vorlesungsmodul Softwaretechnik 3
- VorlMod SoftwareTk3 -

Matthias Ansorg

29. September 2003 bis 14. Januar 2004

Zusammenfassung

Studentische Mitschrift zur Vorlesung Softwaretechnik 3 »Modellieren und Entwickeln mit Muster« bei Prof. Dr. Quibeldey-Cirkel (Wintersemester 2003/2004) im Studiengang Informatik an der Fachhochschule Gießen-Friedberg. Die Inhalte stimmen weitgehend mit dem Skript von Prof. Renz [0] überein. Jedoch setzt Prof. Dr. Quibeldey-Cirkel die Schwerpunkte auf Analysemuster, Refactoring und Modultest mit JUnit. Dieser Stoff stammt aus [4], [6], [7]. In der Veranstaltung bei Prof. Quibeldey-Cirkel werden relativ gute Englischkenntnisse verlangt.

- **Bezugsquelle:** Die vorliegende studentische Mitschrift steht im Internet zum Download bereit: Persönliche Homepage Matthias Ansorg :: InformatikDiplom `http://matthias.ansorgs.de/InformatikDiplom`
- **Lizenz:** Diese studentische Mitschrift ist public domain, darf also ohne Einschränkungen oder Quellenangabe für jeden beliebigen Zweck benutzt werden, kommerziell und nichtkommerziell; jedoch enthält sie keinerlei Garantien für Richtigkeit oder Eignung oder sonst irgendetwas, weder explizit noch implizit. Das Risiko der Nutzung dieser studentischen Mitschrift liegt allein beim Nutzer selbst. Einschränkend sind außerdem die Urheberrechte der angegebenen Quellen zu beachten.
- **Korrekturen und Feedback:** Fehler zur Verbesserung in zukünftigen Versionen, sonstige Verbesserungsvorschläge und Wünsche bitte dem Autor per e-mail mitteilen: Matthias Ansorg <`mailto:matthias@ansorgs.de`>.
- **Format:** Die vorliegende studentische Mitschrift wurde mit dem Programm LyX(graphisches Frontend zu LaTeX) unter Linux geschrieben und mit pdfLaTeXals pdf-Datei erstellt. Grafiken wurden mit dem Programm xfig unter Linux erstellt und als pdf-Dateien exportiert.
- **Dozent:** Prof. Dr. Klaus Quibeldey-Cirkel.
- **Verwendete Quellen:** .
- **Klausur:** siehe Kapitel 1.4.

Inhaltsverzeichnis

1 Organisation **2**
- 1.1 Einstufungstest 2
- 1.2 Hausübungen 2
- 1.3 Lehrportal 3
- 1.4 Klausur 3

2 Einführung **4**

3 Einführung in Entwurfsmuster **6**
- 3.1 Symmetrie und Musterbegriff 6
- 3.2 Einige Entwurfsmuster 7
- 3.3 Muster konkret 7
- 3.4 Musterkategorien 8

4 Analysemuster: Observations and Measurements **8**
- 4.1 Associated Observation 8

5 Analysemuster: Accounting 8
5.1 Event . . . 9
5.2 Accouting Entry . . . 9
5.3 Posting Rule . . . 9
5.4 Account . . . 9
5.5 Accounting Transaction . . . 9
5.6 Reversal Adjustment . . . 9
5.7 Difference Adjustment . . . 9
5.8 Replacement Adjustment . . . 9

6 Analysemuster: Referring to Objects 9
6.1 Exkurs: Kritische Aspekte . . . 9

7 Application Facades 9
7.1 Controller . . . 10

8 Object Constraint Language 10

9 Übungsaufgaben 10
9.1 Testklausur . . . 10
9.1.1 Muster Conversion Ratio . . . 11
9.1.2 Klinikinformationssystem . . . 11
9.1.3 Analyse versus Design . . . 11
9.1.4 Attribut versus Assoziation I . . . 12
9.1.5 Attribut versus Assoziation II . . . 12
9.1.6 CRC . . . 13
9.1.7 Begriff-Klasse-Objekt . . . 13
9.2 Martin Fowlers Definition objektorientierter Analyse . . . 13
9.3 Hausübung 1 . . . 13
9.4 Hausübung 2 . . . 14
9.5 Organization Hierachy . . . 14
9.6 Vor- und Nachteile der Verwendung von Mustern . . . 14
9.7 Kritik eines Domänenmodells . . . 14

1 Organisation

1.1 Einstufungstest

Es gab zu Beginn der Veranstaltung einen Einstufungstest. Er war anonym, man konnte aber seine eigenen Ergebnisse anhand eines Pseudonyms erfahren. Der Einstugungstest bestand aus der regulären Klausur SoftwareTk1 bei Prof. Quibeldey-Cirkel. Für die Auswertung wurde dasselbe Notenschema verwendet.

1.2 Hausübungen

Zwei Hausübungen sind Klausurvoraussetzung, entsprechend der Prüfungsordnung. Dabei muss die erste Hausübung im Wesentlichen richtig bearbeitet werden und von der zweiten Hausübung zumindest ein einseitiger Erfahrungsbericht vorliegen, auch wenn man an dieser Aufgabe gescheitert ist. Mit einer ausgefeilten Lösung der Zweiten Hausübung können bis 30% der Punkte der Klausur als Bonuspunkte erworben werden. Das heißt, man kann auch ohne eine Lösung der Zweiten Hausübung in der Klausur die Note 1 bekommen. Die 30% Bonuspunkte führen von Note 4 (50%) zu 80%, d.h. einer Note 2.

Die Hausübungen können mit bis zu 2 (nach anderer Information: 3) Studenten pro Gruppe gelöst werden. Jede Gruppe muss eine eigene Lösung vorschlagen, auch »nicht elegante« Lösungen werden akzeptiert. Mit ordentlich gelösten Hausübungen können bis insgesamt 30% der Punkte der Klausur als Bonuspunkte erreicht werden. In den Übungsstunden werden Ergebnisse in anonymisierter Form durchgesprochen.

Technische Organisation Die Hausübungen werden im Lehrportal abgegeben. Alle Formate, die mit Windows Office gelesen werden können, werden akzeptiert: HTML, PPT, PDF, GIF, TIF, aber nichts Proprietäres.

Erste Hausübung Objektorientierte Analyse und Design zum deutschen Namensrecht. Dazu wird die UML-OCL verwendet. Das zu erstellende Domänenmodell ist in UML ein Klassendiagramm auf abstraktester Ebene, inkl. den Assoziationen der Klassen. Jede Assoziation sollte mit ein bis zwei Rollennamen versehen werden! Der erste Teil der Aufgabe besteht darin, ein Domänenmodell ohne OCL zu erstellen, dabei möglichst viel Strukturinformation durch UML zu modellieren. Im zweiten Teil soll dann die OCL verwendet werden. Es gibt eine PDF-Version zur OCL, erhältlich von der OMG, enthalten in der UML-Spezifikation. Deadline: 2003-12-09 für die erste Hausübung, zweiter Teil.

Zweite Hausübung Ein sog. »Heimpraktikum«. Dazu soll die Eclipse-Plattform mit JDT als IDE verwendet werden. Eclipse ist OpenSource und auch unter Linux verwendbar. Sie unterstützt außer Java weitere Hochsprachen. Zur Integration von UML in Eclipse mit Codegenerierung gibt es das Omondo UML Plugin unter http://www.eclipse-uml.com. »In-Think« meint: aus Code UML generieren und andersherum, man erstellt also beides gleichzeitig.

Mit dieser IDE soll dann das Muster »Application Fassade« in Java implementiert werden. Die Lösung besteht aus einem Programm mit GUI als lauffähiger Software, UML und einer technischen Diskussion des Ergebnisses. Es besteht die Möglichkeit, die zweite Hausübung zu einem Projekt zu erweitern, das man später seinen Bewerbungsunterlagen beilegen kann. Das kann man mit Prof. Quibeldey-Cirkel in seiner Sprechstunde abklären.

In Martin Fowlers Buch »UML Distilled« steht das hervorragende Kapitel »UML und Programmierung«. Darin ist nämlich die (Teil-)Implementierung von Application Facade enthalten. Das Domänenmodell war hier die Patientenbeobachtung im Krankenhaus, die Implementierung geschah in Java. Dabei mussten natürlich Abstriche gemacht werden, weil Abhängigkeit von den Fähigkeiten der Sprache besteht.

Bei Application Facade ist das Domänenmodell vorgegeben. Zuerst soll man es »quick and dirty« funktionsfähig implementieren, dann mit Refactoring es zu einer schöneren Funktion mit gleicher Funktionalität machen. Danach ist der UnitTest mit JUnit durchzuführen. Der Unit-Test ist ein Verfahren in XP: man programmiert immer nur soviel, wie von der Testspezifikation gefordert wird. Es ist ein zielorientierter Prozess (Test-First-Ansatz; es wird nur programmiert, was vom Kunden gefordert wird, nichts irgendwie allgemeingültigeres). Refactoring und JUnit soll man in der zweiten Hausübung selbst spielerisch ausprobieren. Wer nachweisen kann, dass er mit der IDE gearbeitet hat und sagt, dass er nicht damit klargekommen ist, erhält die Zulassung zur Klausur, aber keine Bonuspunkte für diese zweite Hausübung. Dem Kunden ist es egal, wie schön der Code ist; das ist nur wichtig für die Firma: für Wartung und Verbesserung.

Deadline für die Abgabe der letzten Version ist 2004-02-15. Die beste bis 2004-01-12 abgegebene Version würde etwa 20% Bonuspunkte für die Klausur erreichen.

1.3 Lehrportal

Für die Veranstaltung wurde ein Lehrportal eingerichtet. Und zwar noch mit phproject, nicht das neue eStudyPortal. Direkter Link: http://despina.mni.fh-giessen.de/phproject/ http://despina.mni.fh-giessen.de/phproject/.

- Login: Nachname, wie er bei der ersten Hausübung per e-mail angegeben wurde. Erster Buchstabe groß!
- Passwort: Vorname. Erster Buchstabe groß!

Das Portal wurde im Wintersemester 2003/2004 von einem Tutor (Herr Stefan Lauwer) betreut. Es ist ein »Kollaborationsportal«, d.h. es macht nur Sinn, wenn die Studenten tatsächlich auch mitarbeiten. Das kann durchaus eigenes Interesse sein, denn während die Behaltensquote bei Frontal-Vorlesungen 5% beträgt, ist die Diskussion das effizienteste Lernmedium, auch besser als autodidaktische Lernen.

1.4 Klausur

Termin

Die Klausur findet vor den Semesterferien statt.

Hilfsmittel

Es dürfen keine Hilfsmittel verwendet werden. Dadurch sind recht einfache Fragen möglich. Wegen der multiple choice Fragen sollte man einen weichen Bleistift und ein Radiergummi mitbringen. Wegen der eventuellen frei zu beantwortenden Aufgabe sollte man außerdem Papier mitbringen. Verständnisfragen zum Text der Aufgaben können dem Dozenten während der Klausur gestellt werden.

Voraussetzungen

Zwei Hausübungen sind Klausurvoraussetzung, entsprechend der Prüfungsordnung. Wer einmal die Zulassungsvoraussetzungen zur Klausur erfüllt hat, braucht sie nicht ein zweites Mal erfüllen.

Art der Klausur

Die Klausur besteht aus 40-50 gleich gewichteten multiple choice Fragen. Bei wenigen Teilnehmern (also mit geringer Wahrscheinlichkeit) gibt es zusätzlich die Aufgabe, ein gegebenes Domänenmodell zu kritisieren. Verfahren bei multiple choice: für jedes richtige Kreuz ein Pluspunkt; für jedes falsche Kreuz ein Minuspunkt; kein negativer Übertrag zwischen den Aufgaben; die Aufgabenstellung informiert darüber, wieviele Alternativen anzukreuzen sind. Statistisch sind im Mittel 2-3 Antworten anzukreuzen. Die erreichten Bonuspunkte stehen erst nach Auswertung der Hausübungen fest, also nach 2004-02-15. Die in der Klausur erreichten Punkte wird Prof. Quibeldey-Cirkel jedoch auf seiner Homepage veröffentlichen. Dabei ist Note 1 nicht über 95% richtige Antworten definiert, sondern über das Spitzenfeld der Klausurteilnehmer. Unabhängig von etwaigen Bonuspunkten muss jeder die Klausur bestehen, d.h. 50% der Fragen richtig beantworten.

Hinweise zum Stoff

- Zu Beginn der Veranstaltung wurde ein Einstufungstest durchgeführt, der identisch mit der Klausur von Softwaretechnik I war. Die Klausur in Softwaretechnik III wird zu 40% Fragen dieses Einstufungstests enthalten, mit leicht veränderten Antwortalternativen. Das Lehrportal enthält ein Lösungsschema zu diesem Einstufungstest. Die Frage, welche Objektdiagramme nach einem gegebenen Klassendiagramm gültig sind, wird auf jeden Fall enthalten sein.
- Unbedingt klausurrelevanter Stoff:
  - Powerpoint-Präsentationen im Portal.
  - JUnit
  - Funktionalität von Eclipse 2.1 (z.B.: Welche Refactorings unterstützt Eclipse?)
  - Domänenmodellierung
  - Stoff entsprechend dem Dokument Klausurvorbereitung.ppt aus dem Portal, Folie 12.
  - Nur die im Skipt mit drei Sternen ausgezeichneten 21 Muster sind klausurrelevant. Sie werden im Dokument Klausurvorbereitung.ppt, Folie 12, aufgezählt. Man sollte ihre Namen und die zugehörigen Klassendiagramme kennen. Das Muster »Role Object« ist dabei wichtig, ebenso die Muster zu »Referring to Objects«; zu den Accounting Mustern werden jedoch nur wenige Fragen gestellt.
  - Der Stoff vom 2003-12-16 ist nicht klausurrelevant.
  - Zusicherungskonzept in UML: OCL und Constraints. Die Präsentation in der Vorlesung über OCL zeigt, was von OCL man beherrschen muss.
  - Modultests: der Test-First-Ansatz; JUnit und sein interner Aufbau.
  - Vor- und Nachteile des Muster-Ansatzes.

2 Einführung

Problemschema der Denkpsychologie Das Grundverständnis: »Entwerfen ist Problemlösen«. Die Denkpsychologie unterscheidet zwischen Problemen und Aufgaben:

- eine Aufgabe hat einen unbefriedigenden Anfangszustand s_α und einen befriedigenden Endzustand s_ω. Aufgaben löst man mit existierenden Regeln, Rezepten und Algorithmen. Es gibt einen geraden Weg zur Aufgabenlösung.
- ein Problem hat eine »kognitiv-technische Barriere«.

Lösen von Aufgaben ist reproduktiv: Fachwissen ist hinreichend. Problemlösen dagegen ist schöpferisch: Fachwissen ist notwendig, aber nicht hinreichend.

Problemtypen

- Interpolationsprobleme. Um sie (suboptimal) zu lösen, braucht man entsprechend große Rechenkapazitäten. Diese Probleme sind nicht Bestandteil der Softwaretechnik. Beispiele:
  - Constraint-Satisfaction-Probleme. Alle Probleme, bei denen es um Effizienz und Ressourcenverteilung geht. Dies sind NP-vollständige Probleme: sie sind nicht mit geschlossenen Algorithmen in endlicher Zeit lösbar.
  - Schach, Stundenplanung
- Sytheseprobleme. Diese Art Probleme wird in der Softwaretechnik behandelt.
  - Entwerfen mit Bausteinen
  - Alchimisten-Problem, Chip-Design
- dialektische Probleme. Auch diese Art Probleme werden in der Softwaretechnik behandelt.
  - Kundenaufträge (»As proposed by the project sponsor«). Die Kommunikation mit dem Kunden ist ein solches Problem: herauszufinden, was der Kunde will; Gesprächsführung.
  - exploratorisches Prototyping

Entwurfsmuster: aus einem Problem eine Aufgabe machen Das Grundprinzip aller Handlungsmuster ist »Versuch und Irrtum« - formuliert als »dialektischer Regelkreis«. Bisher brauchte man etliche Jahre Befrufserfahrung, um in dem Verfahren »Versuch und Irrtum« genug Erfahrung zu haben, um als Systemanalytiker werden zu können. Heute steht die Kompetenz dieser Experten in Form von Mustern zur Verfügung. Die Muster sind »Heuristiken«, die helfen, die »kognitive Barriere« zu überwinden. Das Problem ist jetzt, das richtige unter den mittlerweile etwa 10^3 Mustern zu finden.

Diese Veranstaltung beinhaltet nicht Strategien der Modellbildung, um die Realität im Computer abzubilden. Der heutige Stand der Softwaretechnik ist dagegen »Modellieren und Entwickeln mit Mustern«. In der Softwareentwicklung sind Muster: Erprobte, typische Lösungen für immer wiederkehrende Probleme in einem definierten Kontext. Solche Muster werden hier für Analyse, Architektur und Entwurf dargestellt. Arten von Mustern:

Analysemuster Wie findet man eine vernünftige Beschreibung eines Problembereichs, genannt »Fachmodell« bzw. »Geschäftsprozessmodell«. Dies sind Muster im Problembereich, verwendet im Arbeitsprozess der »Geschäftsprozessmodellierung«.

Architekturmuster Wie ist die Grobstruktur einer Software aufgebaut? Dies sind Muster im Lösungsbereich. Architekturmuster sind die erste wichtige Entwurfsentscheidung, angesiedelt am Übergang von einer fachlichen Lösung (»Lösungsbereich I«) zu einer technischen Lösung (»Lösungsbereich II«).

Entwurfsmuster Wie ist die Detailstruktur einer Software aufgabaut? Dies sind Muster im Lösungsbereich. Entwurfsmuster werden beim Übergang von Entwurf zu Implementierung verwendet.

Sprachidiome Lösungen für Probleme in bestimmten Programmiersprachen.

Prozessmuster Muster für den Softwareentwicklungsprozess

Organisationsmuster Muster zur Organisation von Softwarefirmen. Der »offizielle Industriestandard« ist »Rational Unified Model«.

Muster des Refactoring Muster zur Verbesserung von Software.

Antimuster Muster, die sagen, wie man es nicht machen soll.

Lernziele Den Schwerpunkt der Veranstaltung bilden Analysemuster, entsprechend der Bezeichnung »Systemanalyse« nach PO 1991.

- Objektorientierte Konzepte und Methoden für Analyse und Design kennen und anwenden können.
  - Was ist ein Objekt? Was ist ein Typ, eine Klasse, eine Schnittstelle? »Typ« ist ein anderes Wort für »Schnittstelle«: das äußere Erscheinungsbild, das die Interna verbirgt.
  - Wie wird das Zusicherungskonzept softwaretechnisch genutzt? In UML wird das durch die OCL (»object constraint language«) ausgedrückt. Um die Semantik des Kunden zu treffen, müssen bestimmte constraints verwendet werden.
- Entwurfsstandards beherrschen
  - Use-Case-Modellierung (Ivar Jacobson, Alistair Cockburn)
  - Software-Qualitätssicherung
    * Modultest (Kent Beck, Erich Gamma)
    * Refactoring (Martin Fowler). Betseht darin, im nachhinein aus Quick-and-Dirty-Programmierung ein gutes Design zu machen.
- Systemanalyse ist »Denkschule«: es ist »mehr als nur Softwaretechnik«. Es ist intellektuell herausfordernd.
  - »Generalist vs. Spezialist«
  - »Ingenieurkunst vs. Routine«
  - »Soft Skills vs. Methoden- und Faktenwissen«

Analyse vs. Design und Implementierung »The task of the software development team is to engineer the illusion of simplicity« (Grady Booch). Es geht darum, anhand des Begriffssystem des Kunden Spezifikationen für eine Lösung mit einfacher Schnittstelle zu erstellen. Diese wird dann von den Programmieren und Technikern erstellt.

3 Einführung in Entwurfsmuster

Das eigene Erfahrungswissen niederzuschreiben ist ein schwieriger Prozess - in der Informatik haben sich deshalb »Autoren-Werkstätten« etabliert.

3.1 Symmetrie und Musterbegriff

Symmetrie ist »das Invarianten, das Erhabene, das Schöne«. Spiegelbildlichkeit ist dagegen »bilaterale Symmetrie« - 95% aller Lebewesen sind so geartet. Aber: erst die Asymmetrie schafft aus etwas sonst symmetischem etwas individuelles, bedeutendes. Die Chaostheorie hat »selbstähnliche Strukturen« erkannt - »nichts ist chaotisch«.

Hardware ist ebenfalls »symmetrisch« - indem sie Symmetrie anwenden, werden Hardwaredesigner produktiv. Hardware hat eine »rhythmische Wiederholung ähnlicher Formen«.

Software als Bitmuster dagegen hat keine wiederkehrenden Strukturen. In anderen Darstellungen wie Assembler oder Sourcecode oder FPGA sind ebenfalls wenige Symmetrien erkennbar.

Um wiederkehrende Strukturen zu erkennen, kann man eine Matrix erstellen, in der in x- und y-Achse jeweils eine Zeile Sourcecode repräsentieren. Übereinstimmungen werden dann durch einen Punkt in dieser Matrix repräsentiert. Mit solchen Mustern kann man Symmetrien (und damit Redundanz) in der Software erkennen. Dieses Verfahren verwendet man eigentlich, um DNA-Code zu analysierern.

Ein ähnliches Verfahren ist es, in einer entsprechenden Matrix die Beziehungen zwischen Klassen mit Strichen darzustellen. Es entsteht ein unorganisiertes Strichmuster.

Die höchste Form der Abstraktion von Software sind Klassendiagramme.

Man versucht, das Invariante in einem System herauszufinden - das sei das Wichtigste. Bei Software sind dies die Muster in einem Klassendiagramm.

Wie alt sind »Muster«? Schon im 19. Jahrhundert gab es »Musterbücher« zu Erfindungen und Handwerksanleitungen. Das waren Handbücher des Entwerfens. Solche Handbücher gibt es auch im Maschinenbau, etwa zum Entwurf von Maschinenelementen. Nur in der Informatik gab es so etwas lange Zeit nicht.

Christopher Alexanders »A Pattern Language« (1977) war ein Musterbuch für die Architektur. Informatiker haben so etwas gelesen und das »Muster« auch in der Informatik eingeführt.

3.2 Einige Entwurfsmuster

Prinzip nahezu aller Muster ist es, die »feste Verdrahtung« von Strukturen in der Software durch Indirektionen aufzulösen, um mehr Flexibilität zu erhalten.

Abstraktion und Ausprägung: dies sind komplementäre Begriffe. Ausprägen ist, Semantik in »idiomatische Strukturen« niederer Sprachebenen umzusetzen. »Idiomatische Strukturen« sind typische Tokenfolgen mit typischer Semantik. Abstraktion ist ein Hilfsmittel, um die Semantik wie sie »im Kopf entsteht« möglichst unmittelbar auszudrücken. Das bedeutet auch, schneller und produktiver. So ist etwa der Produktivitätsgewinn von höheren Programmiersprachen gegenüber Assembler messber, er beträgt Faktor 10.

Es gibt ein musterorientiertes Buch »Projektmanagement« von Cockburn.

Die 80-zu-20-Regel: 80% der Ziele erreicht man mit 20% der Ressourcen. Es lohnt i.A. nicht, die restlichen 20% zu realisieren. Dasselbe ist der Gedanke der Muster: 80% der Architektur sollen mit 20% Aufwand implementiert werden - dazu ist Erfahrungswissen in Form von »Patterns« nötig.

Begriffe:

- Wiedergewinnung: Energie als Ressource
- Wiederverwendung: Information als Ressource
- Wiederverwertung: Materie als Ressource

Früher gab es »Literate Programming« von Donald Knuth: Kommentieren während des Programmierens. Heute wird »Literate Designing« vorgeschlagen. OSEFA ist eine Nachdokumentation eines komplexen Systems.

Was sind Autoren-Werkstätten? Ein Autor und ein Moderator sitzen in einem Kreis, lauter Kommentatoren in einem Kreis runderhum. Bis Patterns in Buchform veröffentlich werden, durchlaufen sie 2 bis 3 Zyklen in einer solchen Autorenwerkstatt. In entsprechender Art sollte man seine Diplomarbeiten lesen lassen: Feedback-Optimierung ist wichtig. Anschließend werden in Autoren-Werkstätten Kreativtechniken angewandt.

Ein »Stereotyp« ist in UML die Angabe von etwas, das in UML selbst nicht modelliert werden kann. In einer Legende muss angegeben werden, was die Stereotypen bedeuten.

3.3 Muster konkret

Möglichkeiten, Muster zu beschreiben:

- universelles Beschreibungsformat in strukturierter, knapper Prosa: Problem, Kontext, Kräfte, Lösung
- objektorientiert (»Gamma-Form«)

Was ist ein »pattern«: a solution to a problem in a context. Das heißt: es gibt keine »Universalpatterns«, sondern die Anwendung eines Patterns ist nur möglich, wenn man im selben Kontext ist. Patterns sind keine kodierten Lösungen, sondern strukturierte Lösungen - es ist noch Ausarbeitung notwendig, die durchaus unterschiedlich ausfallen kann.

RUP ist »rational unified process«. Entwickelt von der Firma Rational. Sie haben diesen Prozess definiert und mit einer mittlerweile populären Software unterstützt. RUP definieren ein Muster wie folgt: »a solution template (for structure, behavior) that has proven useful in at least one practical context, or, more likely, in several contexts, to be worthy of being called a pattern.«.

Eine Kollaboration in UML: es sind Zusammenarbeiten. So etwa sind Usecases Kollaborationen zwischen Akteuren und System. Kollaborationen werden durch Ellipsen dargestellt, die in Klassendiagrammen auftauchen können. Die Instanziierung einer Kollaboration (durch Wahl konkreter Parameter, d.i. durch Verwendung von Namen aus der Applikation statt Platzhaltern wie »Class 1« usw.) nennt die UML dann ein Muster.

3.4 Musterkategorien

Es gibt bisher keinen einheitlichen, weltweiten Index von Mustern. Das wird bisher von allen bedauert. Es gibt jedoch eine Einteilung der Muster nach ihrem Abstraktionsgrad:

- Analysemuster
- Architekturmuster. Schichten werden in UML durch packages ausgedrückt. Pakete in UML können »alles« enthalten.
- Design-Muster. Beispiel: Das Muster »Proxy« (engl. für »Stellvertreter«).
- Idiome. Beispiel: Das Muster »Singleton (C++)« ist ein idiomatisches Muster in C++, das das Design-Muster »Singleton« implementiert. Das Problem: »Sorge dafür, dass eine Klasse genau ein Exemplar besitzt, und stelle einen globalen Zugriffspunkt darauf bereit.«.

4 Analysemuster: Observations and Measurements

4.1 Associated Observation

Frage: Was für eine Aufgabe hat »Associated Observation«? Anscheinend hat sie zwei Aufgaben. Erstens, die Verbindung einer Diagnose (als Observation) zu einer Associative Function herzustellen, wozu die Diagnose als Associated Observation wiederholt und mit der Diagnose assoziiert wird. Zweitens, die Verbindung zu Observations am selben Patienten herzustellen, die gemäß der Associative Function geeignet sind, die Diagnose zu unterstützen. Diese Verbindung besteht *nicht* über die Associative Function, denn es gibt keine Abhängigkeiten der Wissensebene von der operationalen Ebene. Nur umgekehrt: jede Observation kann einem Observation Concept zugeordnet werden.

5 Analysemuster: Accounting

Enthalten in Fowlers überarbeitetem Kapitel von 2000. Zwischen Analyse- und Entwurfsmustern sind die sog. »Unterstützungsmuster« angesiedelt. Sie helfen zur Umwandlung.

Es gibt in Deutschland eine Buchführungspflicht, allerdings keine Formularpflicht. Ordnungsgemäße Buchführung hat vor Gericht Beweiskraft. Deshalb: alle Änderungen müssen nachvollziehbar sein, es darf keine Löschungen geben. Buchführung ist erst ab einem bestimmten Jahresumsatz nötig. Das »T« für Tabellen mit einer Gegenüberstellung von Soll und Haben ist »visueller Jargon« in der Buchhaltung.

Prinzipiell wäre es möglich, alle Buchhaltung von einem zentralen Prozessor erledigen zu lassen, der selbst keine Zustände hat, sondern (statische) Ereignisse in (statische) Buchhaltungseinträge umsetzt.

Gerichtete Assoziationen in UML: Assoziationen sind »Navigationsmöglichkeiten« zwischen Objekten. Assoziationen ohne Richtungsangabe bedeutet dabei, dass Navigationsmöglichkeiten in beide Richtungen zu implementieren sind. Nur eine Richtung fordert nur eine Navigationsmöglichkeit: Zeigt die Assoziation von A nach B, so muss die Navigation von A nach B möglich sein, d.h. Objekt A muss die assozierten Objekte von B kennen, in Gegenrichtung jedoch nicht.

Der Stereotyp «create» an einer gerichteten Assoziation bedeutet den Konstruktoraufruf. Gestrichelte Linien in UML bedeutet Abhängigkeiten. Solche sind auch mit Pfeilspitzen möglich. Die Pattern-Notation in UML verwendet auch diese Abhängigkeitskanten.

Prozedurale Zerlegung nach SA/SD versus Objektstrukturen: ein Paradigma zur objektorientierten Programmierung ist die »Navigation in Objektpopulationen über Assoziationen«. Die prozedurale Bedingungslogik arbeitet mit einem einzigen beliebig tief geschachtelten Algorithmus. OOAD dagegen arbeitet mit der Navigation in den kreierten Strukturen. Die Kombinationsmöglichkeiten sind dabei nicht im geschachtelten Algorithmus, sondern in der Assoziationsstruktur enthalten.

Zu »Accounting-Muster im Kontext, Alternative getrennt«: Zu jedem Objekt von AccountingEvent gibt es evtl. ein Objekt von Adjustment als Companion-Objekt. Dieses definiert, ob das AccountingEvent-Objekt noch gültig ist oder ggf. korrigiert ist. Es gibt im Laufzeitsystem also sowohl Objekte »nur von« AccountingEvent, als auch von Adjustment.

5.1 Event

Ereignisse kann man in der Objektorientierung unterschiedlich darstellen: als Methodenaufrufe, oder, wenn es sich wie hier um domänenrelevante Ereignisse handelt, die das System nicht vergessen darf, als eigene Objekte.

5.2 Accouting Entry

Beschreibungsklassen werden gebildet, indem man an den Klassennamen der zu beschreibenden Klasse »Specification« anhängt. »Descriptor« nach Greg Larmon und »Accounting Entry« nach Fowler stellen dasselbe Muster dar.

5.3 Posting Rule

Die Anwendung eines Patterns auf eine konkrete Domäne geschieht nicht mechanisch. Sondern es werden Rollen vergeben. Die Assoziationen zwischen den Klassen in der Domäne können einschränkender oder erweiternd sein gegenüber denen aus dem Pattern. Nur das Prinzip wird übertragen!

5.4 Account

5.5 Accounting Transaction

5.6 Reversal Adjustment

5.7 Difference Adjustment

5.8 Replacement Adjustment

6 Analysemuster: Referring to Objects

6.1 Exkurs: Kritische Aspekte

Der Systemanalytiker erstellt Typendiagramme, der Designer erstellt Klassendiagramme, und der Implementierer erstellt Quellcode. Weil es in UML keine dedizierten Typendiagramme gibt, werden die Ausdrucksmittel der Klassendiagramme dafür verwendet. Deshalb muss man darauf achten, beides nicht zu verwechseln: Typen und Klassen sind etwas völlig unterschiedliches.

- Typen sind Implementierungen von Begriffen des realen Informationssystems, d.h. von Begriffen der realen Welt. Sie existieren nicht im fertigen Software-Produkt, sondern als Konzepte in der Domäne; sie sind ein Domänen-Artefakt. Typen beschreiben nur die Struktur der realen Welt, nicht ihr Vehalten; d.h. Typen kann man als Schnittstellen auffassen.
- Klassen sind Implementierungen von Typen. Sie existieren in der fertigen Software, d.h. sie sind ein Software-Artefakt. Ein Typ kann durch ein Konstrukt aus mehreren Klassen implementiert werden.
- Assoziationen in Typendiagrammen sind Implementierungen von Beziehungen der realen Welt.
- Assoziationen in Klassendiagrammen sind Implementierungen von Assoziationen in Typendiagrammen.

Während die Klassendiagramme des Designers sich an den üblichen Fähigkeiten von Programmiersprachen orientieren, haben Typendiagramme überhaupt nichts mit Programmiersprachen zu tun. Sie orientieren sich an der realen Welt ohne jeden Bezug zur Implementierung in Software. Daher können Elemente wie dynamische (Um-)Klassifizierung (`«dynamic»`) und Mehrfachklassifizierung (`«incomplete»`) enthalten sein, diese beiden Elemente werden sogar gerne verwandt.

7 Application Facades

Typische mehrschichtige Software-Architektur in Anwendungen mit Application Facade:

- View Layer. Dies ist die Fassade der Anwendung, die »Klick-Oberfläche«.
- Application Layer. Zum Beispiel die Erfassung von Patientendaten. Hier ist die Logik enthalten, die nur für diese spezielle Applikation notwendig ist, nicht zur Domäne an sich gehört.

- Domain Layer. Das Domänenmodell der Anwendung, das Begriffssystem. Sie ist unabhängig von der Applikation und ihrer speziellen Logik, auch unabhängig von der Datenquelle.
- Infrastructure Layer. Eine Datenbank im Hintergrund.

Gründe für mehrschichtige Software-Architektur:

- Möglichkeit zur Arbeitsteilung im Team.
- Möglichkeit, sich auf einzelne Teile zu konzentrieren, wenn man allein arbeitet. Reduktion von Komplexität: ein Mensch kann immer nur einen Ausschnitt bilden und daran arbeiten. Nämlich 7 Elemente.
- Netzwerktransparenz der Schichten, verteilte Systeme. Keine Logik in der Präsentationsschicht.

7.1 Controller

Das Abbild des allgmeingültigen Mikroprozessors in der Softwaretechnik. Das Controllerobjekt kann als GoF-Muster »Fassade« realisiert werden. Ziel ist es, nur einige wenige Zugriffspunkte auf die Applikationslogik zur Verfügung zu stellen.

OOADP = object oriented analysis, design and programming

Aufgabenstellung der 2. Hausübung ist in der Präsentation »Analysemuster_5« enthalten. Die Oberfläche der Applikation sollte man einfallsreich gestalten. In der Hausübung müssen die Pakete entsprechend dem Paketdiagramm auftreten. Mit JUnit soll nur das Paket »application facade« getestet werden. Das Domänenmodell ist bereits fertig ausformuliert, muss nur noch abgeschrieben werden. Dabei sind die darin enthaltenen Analysemuster zu implementieren.

Das konkrete Problem, das mit dieser Hausübunggelöst werden soll: ein Informationssystem für das Gesundheitswesen, mitd em alle Patientendaten erfasst und neue eingefügt werden können. Es ist von Martin Fowler in Prosa (grüner Kasten) formuliert.

Als Artefakte müssen abgegeben werden:

- Formulierung des konkreten Problems
- Usecases und Domänenmodell als Analyse-Modell. Das Domänenmodell liegt bereits fertig vor von Martin Fowler. Außerdem gehört hierhin das Glossar.
- Im Designmodell: Design-Klassendiagramme (inkl. Design-Klassen wie Registratur, Datenstrukturen, Sortieralgorithmen, ...) und ein Sequenzdiagramm entsprechend dem Usecase-Diagramm. Ein Upper-Case-Werkzeug könnte aus dem Sequenzdiagramm Codefragmente generieren.
- Ausformulierung zu einem konkreten Informationssystem. Dabei werden JUnit und Refactoring verwendet. Refactoring ändert nichts an der Funktion, verbessert aber die Struktur. Man sollte eine Recherche über Refactoring im Internet durchführen.

Die Applikationslogik soll ohne die Oberfläche funktionieren.

8 Object Constraint Language

9 Übungsaufgaben

9.1 Testklausur

Hier: Besprechung der Testklausur. Die Sprache der multiple choice Fragen muss man beherrschen. Es gibt da einige Schikanen, auch in diesen SoftwareTk3-Klausuren. Der Fachbereich MNI hat multiple choice mit einer entsprechenden teuren Software eingeführt; der Student wird dieser Art Fragen also in Zukunft öfters begegnen.

Es ist dabei gewollt, dass Antworten ähnlich sind - damit man zeigen kann, ob man etwas tatsächlich weiß. Es gibt einige Ablenkungsantworten.

Die zukünftigen Klausuren werden Mischformen aus multiple choice und sonstigen Aufgaben sein. Wer den multiple-choice Test besteht, hat dabei nur eine 4.

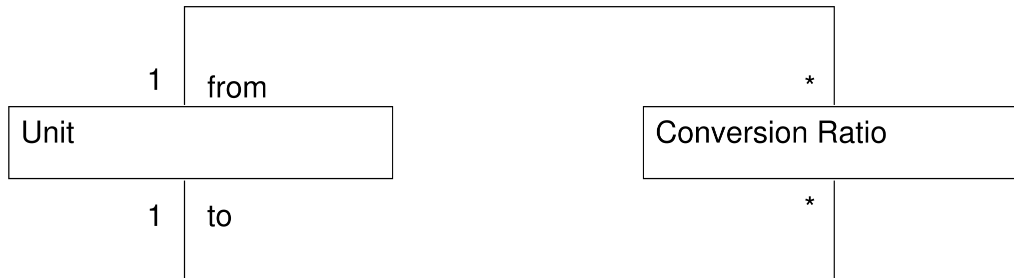

Abbildung 1: Das Muster »Conversion Ratio«

Bei der Testklausur wurde dasselbe Notenbild verwendet wie in der originalen SoftwareTk1-Klausur. Es sind 42% durchgefallen, 10 hatten 3.0 oder besser. Die Durchfallquote der SoftwareTk1-Klausur war bei 47%.

Bei zufälliger Beantwortung der multiple choice Klausuren fällt man erfahrungsgemäß stets durch.

Bei der Testklausur wurde erkannt: die meisten beherrschen die »Buchstaben« der UML, aber nicht die »Syntax«: wie man die Buchstaben zu sinnvollen Wörtern und Sätzen zusammensetzt.

Außerdem notwendig: die geforderten Fachbegriffe wiederholen.

9.1.1 Muster Conversion Ratio

Dieses Muster wurde nicht explizit in UML behandelt. (50% der Antworten waren richtig). Worte an einer Assoziation in der Nähe einer Klasse sind Rollenbezeichner dieser Klasse. Zahlen an diesen Stellen sind Multiplizitäten (dagegen meint »Kardinalitäten« dasselbe in der Datenbanktechnik).

Der »*« meint »0...*«, nicht »1...*«.

In diesem Muster beobachtbar: »Rolle« und ihre »inverse Rolle«.

9.1.2 Klinikinformationssystem

Das Muster »Measurement«. Entitäten, die mehr leisten als nur Attribut einer einzigen Klasse sein, sollten ausgegliedert werden.Es hilft oft, die inverse Rollenfunktion (entgegengesetzte Kardinalität) zu betrachten: man muss die Assoziation von beiden Seiten betrachten, um zu überprüfen, ob das Modell stimmt.

Man kann Kompositionen und Aggregationen und Vererbung als symbolische Abkürzungen für bestimmte Assoziationsarten (Assoziationsnamen) betrachten:

- Komposition (ausgemalte Raute) für »has as part«
- Aggregation (leere Raute) für »has«. Eine Beziehung von Ganzem und Teil. Das Ganze ist ein »Container«.
- Vererbung (Dreieck) für »is a«.

Komposition und Aggregation wird in der Analyse nur als »letztes Mittel der Wahl« verwendet: diese Dinge sind zu starke Abhängigkeiten (constraints), die in der Realität meist nicht gegeben sind. Beispiel: Patient als Container für Messungen zu verwenden (Aggregation) bedeutet, dass der Patient alle seine Messungen mitnimmt, wohin er auch geht.

»Measurement« gehört zur operationalen Ebene, »PhenomennType« gehört zur Wissensebene. PhenomenonType ist nicht die Einheit (diese steht schon in »Measurement«), sondern die Interpretation: etwa Übergewicht, Untergewicht, Normalgewicht.

Faustregel: Gibt es eine verbindende Klasse,

9.1.3 Analyse versus Design

Welche Unterschiede bestehen zwischen der Analyse und dem Design von Informationssystemen? Zwar sind die Übergänge zwischen Analyse und Design seit der Objektorientierung fließend, aber trotzdem gibt es wesentliche Unterschiede. Die richtigen Antworten:

- Analyse ist problemzentriert, Design lösungszentriert. Analyse ist der dialektische Prozess, das Problem des Kunden herauszufinden.
- Entdecken ist typisch für die Analyse, Erfinden ist typisch für das Design. (von Grady Booch)

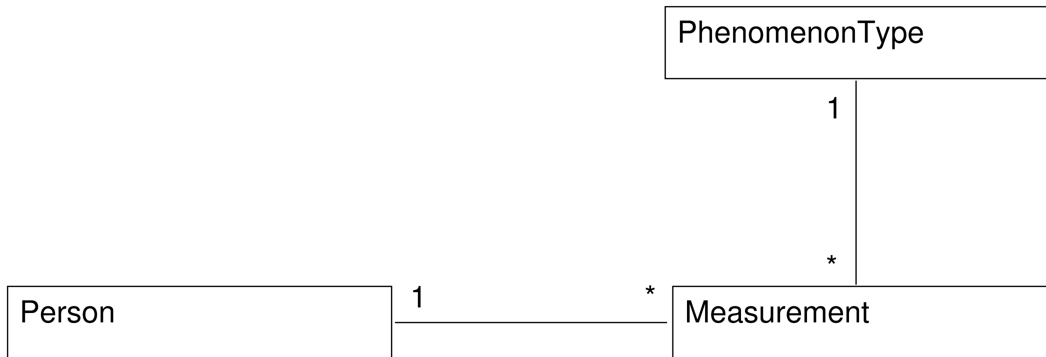

Abbildung 2: Zu einem Klinikinformationssystem

9.1.4 Attribut versus Assoziation I

»Objekteigenschaften können als Objektattribute oder Assoziationen zu Werteobjekten (value objects) modelliert werden. Welche der folgenden Eigenschaften sollten als Assoziationen modelliert werden?«

Es geht hier nicht um die Fähigkeiten der Programmiersprache, sondern um die Modellierung während der Analyse. Wenn also eine Programmiersprache »Datum« nicht als konkreter Datentyp (Datentyp, mit dem der Compiler arbeiten kann) in der Programmiersprache zur Verfügung steht, bedeutet das nicht, dass Datum als Assoziation modelliert werden muss. »Konkrete Datentypen« sind all die Datentypen, die ein Compiler direkt verarbeiten kann. »Abstrakte Datentypen« sind dagegen alle selbstdefinierten Datentypen.

Was ist das Entscheidungskriterium? Alle elementaren Dinge (d.h. alle konkreten Datentypen) werden als Attribute modelliert. Alle komplexen Dinge (d.h. alle abstrakten Datentypen) werden als Assoziationen modelliert. Je komplexer die Dinge sind, desto eher sollten sie ausgelagert werden. Die richtigen Antworten: Als Assoziationen sollten modelliert werden:

- SI-Einheiten. Weil keine Programmiersprache der Welt sie als konkrete Datentypen zur Verfügung stellt.
- Sozialversicherungsnummer. Weil sie eine innere Struktur hat.
- Im Zweifelsfall immer als Assoziation. Mit dieser Entscheidung ist man immer auf der richtigen Seite; sollte sich eine Entität als komplex erweisen, kann man dies noch berücksichtigen, wenn man sie als Assoziation modelliert hat.

9.1.5 Attribut versus Assoziation II

»Objekteigenschaften können als Objektattribute oder Assoziationen zu Werteobjekten (value objects) modelliert werden. Welche der folgenden Eigenschaften sollten als Attribute modelliert werden?« Die richtigen Antworten:

- Boolean
- Geburtsdatum. Hier wird vorausgesetzt, dass es ein Geburtsdatum als konkreten Datentyp in der verwendeten Programmiersprache gibt. Folgerichtig wurde in »Attribut versus Assoziation I« Datum auch nicht (!) als Assoziation modelliert.

In SoftwareTk3 gehe man immer davon aus, dass komplexe Probleme modelliert werden. Jede beobachtare Struktur sollte man dahr modellieren, sie ist relevant. Das ist die Begründung, warum »Name« nicht als String, nicht als konkreter Datentyp modelliert wird. Sondern als Assoziation, weil »Name« eine Struktur hat.

Man beachte, dass »Assoziation« aus dieser Frage in UML sowohl »Komposition« (ausgefüllte Raute) als auch »Aggregation« (leere Raute) sein kann.

Aus Sicht des Systemanalytikers ist die Modellierung als Komposition und als Attribut semantisch identisch. Attribute sind nur eine Anlehnung an existierende Programmiersprachen, als eigenständiges Konzept in UML aber unnötig.

9.1.6 CRC

Die richtigen Antworten:

- Die CRC-Methode zählt zu den Best-Practices-Methoden im XP-Vorgehensmodell.
- CRC steht für Class-Responsibility-Collaborator.
- Die CRC-Methode ist eine Brainstorming-Methode zur Klassenfindung.

9.1.7 Begriff-Klasse-Objekt

Die richtigen Antworten:

- Eine Klasse implementiert den Abstrakten Datentypen (ADT)
- Eine Klasse ist die Extension eines Begriffs
- Eine Klasse ist eine Menge (im mathematischen Sinne) von Objekten

9.2 Martin Fowlers Definition objektorientierter Analyse

Aufgabenstellung Wie definiert Martin Fowler die objektorientierte Analyse (OOA) im Artikel »Is there such a thing as object-oriented analysis?«? Der Artikel ist enthalten in [1].

Lösung

Was ist Analyse? Analyse modelliert die eigene Wahrnehmung und das eigene Verständnis der Realität, nicht die Realität selbst. Sie ist darum nicht rein deskriptiv, sondern konstruktiv: alle Abstraktionen, Vereinfachungen und Vereinheitlichungen sind Konstruktionen, mit deren Hilfe Realität verstehbar wird. Analyse ist, den Problembereich verstehen, Design ist, die Software verstehen, die den Problembereich unterstützt. Die Trennung ist unscharf. Analyse ist in Essenz die Definition der Geschäftsprozessobjekte und Geschäftslogik.

Was ist objektorientierte Analyse? Eine Analyse, die in der Sprache der Objektorientierung formuliert wird. Weil das für eine anschließende objektorientierte Implementierung nützlich ist. Damit ist OOA technologieabhängig. Die »Konstruktion« in der Analyse ist konkret bei OOA die »Objektfindung«.

9.3 Hausübung 1

Wichtig: In der Analyse formuliert man fast ausschließlich reine Assoziationen, ohne zu sagen, ob es Aggregationen oder Kompositionen sind. Auch mit Vererbung beschäftigt sich die Analyse kaum oder nicht.

Ausdrucksmittel in UML, die OCL tw. ersetzen können:

- beschränkt sichtbare Attribute und Methoden und Konstruktoren
- «abstract«
- property string
- Notiz
- Assoziationsklasse
- «ordered» bei Assoziationen
- mehrgliedrige Assoziationen
- Diskriminator bei Vererbung
- Aggregation, Komposition
- Interface, Interface-Realisierung, Interface-Erweiterung
- abstrakte Klassen, abstrakte Attribute, abstrakte Mathoden

- Mehrfachvererbung
- Datentypen von Attributen. Sie werden als eigene Klassen (ohne Verbindungen) aufgelistet und haben Stereotypen wir «primitive» oder «enumeration».
- Rollen und Sichtbarkeiten in Assoziationen
- qualifizierte Assoziationen; so können Assoziationen mit Eigenschaften versehen werden.

9.4 Hausübung 2

9.5 Organization Hierachy

Eine mögliche Frage in der Klausur: Gegeben ist ein Objektdiagramm oder Quellcode. Handelt es sich um das Muster »Organization Hierachy« oder nicht? Erkennungsmerkmal ist die rekursive Assoziation!

9.6 Vor- und Nachteile der Verwendung von Mustern

- Muster stellen stets eine weitere Indirektion dar; das bedeutet mehr Code, mehr Klassen und weniger Geschwindigkeit.
- Wesentlicher Vorteil ist die späte Bindung (»late binding«), d.h. Bindung zur Laufzeit, gegenüber der frühen Bindung (»early binding«), d.h. zur Übersetzungszeit. Besonders die Muster der GoF haben alle den Anspruch, mehr späte Bindung zu bieten. Fowlers Muster »Role Object« bietet ebenfalls späte Bindung.
- Ein Ziel der Muster wie auch der Weiterentwicklung von Programmiersprachen ist »maximale Polymorphie«: Operationen, deren Verhalten erst zur Laufzeit definiert ist.

9.7 Kritik eines Domänenmodells

Die Kriterien:

- Einfachheit ist ein Kriterium der Modellgüte. Man sollte die Dinge einfach machen, aber nicht zu einfach.
- Eine Faustregel: vermittelnde Klassen haben oft beiderseits beliebige Multiplizität.
- Domänenmodelle sind Analysemodelle. Analytiker sind sehr sparsam mit Kompositionen und lassen Multiplizitäten oft unspezifiziert. Alles andere ist meist ein schlechtes Analysemodell.

Literatur

[1] Prof. Dr. Quibeldey-Cirkel: Unterlagen zur Veranstaltung Softwaretechnik 3. Zur Verfügung gestellt als ein ZIP-Archiv. Quelle: Homepage von Prof. Dr. Quibeldey-Cirkel http://homepages.fh-giessen.de/~hg13345/ Passwort für die zip-Archive `SWTIII-KQCKQCKQC`.

[0] Prof. Dr. Burkhardt Renz: Folien zur Veranstaltung Softwaretechnik 3. Besteht aus vier Teilen in vier Dateien. Quelle: Homepage von Prof. Burkhardt Renz :: SoftwareTk3 http://homepages.fh-giessen.de/~hg11260/hp_v5_de.html

[2] Prof. Dr. Burkhardt Renz: Aufgabensammlung Softwaretechnik 3, Sommersemester 2003.

[3] Reza Rashidi: »Mitschrift der Vorlesung Systemanalyse Sommersemester 2001«. Sie enthält nur an einer Stelle einen fachlichen Fehler, ist sonst sehr lesenswert und enthält das, was Prof. Renz behandelt hat. Sie deckt jedoch nur noch 30% des Stoffes der aktuellen Veranstaltung ab. Sie ist erhältlich auf der Homepage von Reza Rashidi.

[4] Martin Fowler: »Analysis Patterns: Reusable Object Models«; Addison-Wesley. »Die Referenz für Analysemuster«. Dieses Buch wird als »einziges Buch für Analysemuster« beschrieben. Die Muster in dieser Vorlesung stammen aus diesem Buch. In einigen Exemplaren in der Bibliothek der FH Gießen-Friedberg enthalten. Auf der Homepage von Martin Fowler sind die wichtigsten Dinge in UML-Notation enthalten.

[5] Martin Fowler: »Architekturmuster«. Aus diesem Buch stammen die meisten Architekturmuster in der Veranstaltung »Softwaretechnik 3« bei Prof. Dr. Renz. Auf der Webseite von Sun gibt es die »J2EE-Patterns« in den »J2EE-Blueprints«; dies sind etwas speziellere Versionen der Architekturmuster von Fowler. In einigen Exemplaren in der Bibliothek der FH Gießen-Friedberg enthalten.

[6] Martin Fowler: »Refactoring«. In einigen Exemplaren in der Bibliothek der FH Gießen-Friedberg enthalten.

[7] »Unit Tests mit Java«. In einigen Exemplaren in der Bibliothek der FH Gießen-Friedberg enthalten.

[8] Klaus Quibeldey-Cirkel: »Entwurfsmuster«; Springer-Verlag. Dieses Buch enthält viel vom Stoff seiner Veranstaltung, ist jedoch keine Pflichtlektüre, um die Klausur bestehen zu können. Einige Auszüge daraus sind auf der Homepage von Prof. Dr. Quibeldey-Cirkel enthalten.

[9] Martin Hitz, Gerti Kappel: »UML @ Work: Von der Analyse zur Realisierung«; dpunkt-Verlag. Ein recht guter Überblick über UML.

[10] Heide Balzert: »Lehrbuch der Objektmodellierung: Analyse und Entwurf«; Spektrum Akademischer Verlag. Einige der in der Vorlesung enthaltenen Analysemuster sind hieraus entnommen.

[11] Peter Coad, Eric Lefebvre, Jeff De Luca: »Java Modeling in Color with UML: Enterprise Components ans Process«; Prentice Hall. Das hier in den ersten Kapiteln dargestellte Verfahren der farbigen UML-Klassendiagramme wird in der Vorlesung extensiv besprochen.

[12] Frank Buschmann, Regine Meinert, Hans Rohnert, Peter Sommerlad, Michael Stal:.

[13] Craig Larman »Applying UML and Patterns«; Prentice Hall. Das wohl beste Buch für objektorientierte Softwareentwicklung.

[14] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: »Design Patterns: Elements of Reusable Object-Oriented Software«. Addison-Wesley. Der beste Startpunkt für Entwurfsmuster.

[15] John Vlissides: »Pattern Hatching«, Addison-Wesley. Deutsche Fassung: »Entwurfsmuster anwenden«.

[16] UML-Spezifikation Version 1.5. http://www.omg.org/cgi-bin/doc?formal/03-03-01.

[17] Eclipse-IDE. http://www.eclipse.org