Einleitung

Datum:	11. Oktober 2007
Prüfung:	**Vertiefungsprüfung Softwarekonstruktion** • Softwaretechnik • Objektorientierte Softwarekonstruktion • Software-Qualitätssicherung und Projektmanagement • Software-Produktlinien Entwicklung
Prüfer:	**Professor Lichter**
Dauer:	ca. 45 Minuten
Note:	1,3

Die Prüfung verlief ruhig und entspannt, bei einigen Fragen war mir nicht direkt klar was Prof. Lichter meint, da er erstmal sehr allgemein fragt. Aus den Antworten sucht er sich dann etwas aus, und geht in die Tiefe. Zum Teil Produktlinien hat er mir keine Fragen gestellt, wohl aus Zeitgründen. Die Prüfung verlief generell so, dass zwischen den einzelnen Prüfungsthemen hin und her gewechselt wurde, das Protokoll lässt sich also leider nicht streng in die einzelnen Fächer unterteilen, sonst müsste ich die Fragen aus dem Zusammenhang herausnehmen.

Gedächtnisprotokoll

Lichter: Was ist denn Softwaretechnik oder Software Engineering?
Ich: Das ist ein Modell der Software-Entwicklung.

Ein Modell... ein Modell. Naja, gut ist auch ein Modell.
Softwaretechnik ist ein Ansatz die Software-Entwicklung ingenieurmäßig zu betreiben. In den 60er Jahren gab es die Software-Krise...

...Die gibt's immer noch. Aber was bedeutet denn „ingenieurmäßig"? Was ist da das Wichtigste?
Also bei den Ingenieurprinzipien geht es vor allem um Kostendenken, Qualitätsbewusstsein, Normen, sowas. Bei der Software-Entwicklung wollte man weg vom Künstler der die Software schreibt.

Ja es geht vor allem um die strukturierte Entwicklung. Was haben wir denn kennengelernt, um die Software-Entwicklung zu strukturieren.
Vorgehensmodelle, Prozessmodelle.

Was ist denn da der Unterschied?
Vorgehensmodelle beschreiben das Vorgehen während der Entwicklung, Prozessmodelle enthalten neben einem Vorgehensmodell noch Vorgaben für Projektmanagement, Qualitätssicherung, Konfigurationsmanagement.

Welche Prozessmodelle kennen Sie denn?
Phasenmodell, Unified Process speziell RUP, Extreme Programming.

Wenn wir uns den Unified Process mal ansehen. Was macht man denn da?
Der Unified Process ist erstmal ein Phasenmodell, und innerhalb der Phasen läuft die Entwicklung iterativ ab, und die Produkte entstehen inkrementell. Die 4 Phasen sind: Inception, ...

...Die brauchen Sie nicht aufzählen. Was macht man denn in den Phasen?
In den Phasen geht man verschiedenen Tätigkeiten nach, den Workflows oder Disciplines je nach Urheber.

Also das ist das Entscheidende. Man verfolgt gleichzeitig verschiedene Tätigkeiten.
Genau. Im reinen Phasemodell macht man die Tätigkeiten ja nacheinander in dafür vorgesehenen Phasen. Das war nur für die Objektorientierung nicht besonders zweckmäßig.

Wenn wir uns das Extreme Programming ansehen, was ist der große Unterschied zum RUP, wenn man sich die Anwendung ansieht?
Extreme Programming ist eher was für kleine Entwicklungsteams und überschaubare Projekte, RUP könnte ich mir auch bei Projekten mit 500 Entwicklern vorstellen, XP funktioniert da wahrscheinlich nicht.

So groß braucht man gar nicht werden. Wie entwickelt man denn beim XP?
Das läuft in kleinen Schritten ab, man entwickelt erst einen Test, dann implementiert man gegen den Test, und dann testet man, bis der Test keinen Fehler mehr findet.

OK. Wie kommt man denn beim XP an die Anforderungen?
Hmm. Ganz normal indem man den Kunden befragt?

Naja. Da gibt es schon eine spezielle Art.
Ach ja. Man hat in der Entwicklungsorganisation einen Kunden sitzen, der einem die Tests schreibt.

Wenn er es kann.
Wenn er es nicht kann dann muss er die Test halt prüfen und gucken ob das das ist was der Kunde will. Der Kunde muss quasi einen Vertreter für die Entwicklung abstellen.

Ja. Wissen Sie noch, wie man beim XP die Anforderungen notiert?
Hmm. Ganz normal natürlichsprachlich?

Ja sicher, aber da gibt es eine spezielle Art.
Puh. Weiss ich nicht mehr...

User Stories, sagt Ihnen das was?
Ja genau. Das sind kleine Geschichten von Benutzern.

Naja... heisst ja Stories...
Ja OK. Das sind sowas wie Anwendungsfälle. Einzelne Funktionalitäten aus Sicht eines Benutzers beschrieben.

Ja sowas wie Anwendungsfälle. Kommen wir mal generell zum Requirements Engineering. Wie erhebe ich denn Anforderungen?
Hmm.

Da gibt's so 3 Schritte, die man macht, erinnern Sie sich?
3 Schritte? Hmm... Also ich würde erstmal den Kunden befragen was er haben will.

Genau, erstmal nehmen sie die Wünsche des Kunden auf. Was kommt dann?
Dann würde ich sie modellieren, und dann eine Architektur entwerfen.

Da fehlt aber noch etwas. Was müssen Sie denn machen, bevor Sie die Architektur entwerfen?
Hmm. Die Anforderungen prüfen?

Genau. Warum?
Damit ich nichts falsches entwickle, da in einer frühen Phase begangene Fehler nachher richtig viel Geld kosten. Das muss man ja mit den echten Anforderungen vergleichen.

Sie müssen prüfen, ob das auch wirklich das ist was der Kunde will, genau. Stellen Sie sich mal vor sie hätten die Anforderungen vorliegen. Wie würden Sie die denn prüfen?
Naja... Ich würde Sie mit dem Kunden besprechen.

Nein. Wie prüfen Sie denn Anforderungen?
Ah, (klick), eine dynamische Prüfung würde sich da nicht anbieten, da man ja noch nichts in der Hand hat. Also würde ich eine statische Prüfung machen, und da haben wir speziell Review und einen linguistischen Ansatz mit sprachlichen Defekten wie Tilgung, Generalisierung und Verzerrung kennengelernt.

Also ein Review ist eine gute Idee. Sie machen also ein Review der Anforderungen. Warum machen Sie das?
Damit ich möglichst früh Fehler in den Anforderungen finde, da sich in einer frühen Phase der Entwicklung jeder Aufwand lohnt, der Fehler vermeidet. Die Fehler werden ja teurer, je später sie entdeckt werden, da haben wir ja z.B. die Badewannenkurve gesehen.

Gut. Sie haben jetzt also die Anforderungen geprüft. Jetzt wollen Sie einen Entwurf daraus machen. Wie gehen Sie da vor?
Ich entwerfe eine Architektur des Systems.

Was ist denn eine Architektur?
Das ist die Struktur des Systems, bei der man von den Details der Implementierung abstrahiert, und z.B. nur Schnittstellen und Module spezifiziert. Da gibt es verschiedene Sichten auf die Architektur.

Welche denn?
Die statische, dynamische und Systemsicht, bei der das System in seinem Kontext dargestellt wird.

Wofür ist denn die dynamische Sicht der Architektur?
Da sieht man das Zusammenspiel von Objekten, die sich Nachrichten schicken, z.B.

Objekte hat man ja in der Architektur gar nicht. Was könnte denn die dynamische Sicht sonst zeigen?
Dann wahrscheinlich die Benutzt-Beziehungen zwischen verschiedenen Einheiten.

Ja. Wie die Module sich untereinander benutzen zum Beispiel. Gehen wir mal zurück zur Modellierung. Wie modelliert man denn Anforderungen?
Zum Beispiel mit Use Cases.

Genau. Wie sieht so ein Use Case aus?
Der beschreibt eine funktionale Anforderung an das System von außen.

Wie beschreibt man denn nicht-funktionale Anforderungen?
Natürlichsprachlich, aber nicht als Use Case.

Use Cases beschreibt man ja auch natürlichsprachlich, aber Sie meinen die Diagrammdarstellung in UML. Was hat man denn da so für Elemente?
Den Use Case, Akteure, ein System.

Und was noch?
Hmm...

Man hat noch Beziehungen. Was gibt's denn für Beziehungen?
Extend-Beziehung, beschreibt einen optionalen Use Case, der ausgeführt werden kann, wenn eine bestimmte Bedingung erfüllt ist. Include, damit fügt man die Funktionalität eines Use Cases in einen anderen Use Case ein, und zwischen Akteuren kann man auch vererben.

Nur zwischen Akteuren? Nicht zwischen Use Cases?
Ich glaube nicht.

Doch kann man schon. Abstrakte Use Cases zum Beispiel.
Stimmt, ich erinnere mich.

Wann ist denn ein Entwurf gut?
Wenn er das System gut strukturiert... Naja... wenn sich die Einheiten nicht wild gegenseitig benutzten.

Ja schon richtig, aber was macht einen guten Entwurf aus? Was für Richtlinien gibt es da?
Mir fallen gerade nur Kriterien nach Meyer ein, aber das weiss ich nicht mehr so genau.

Die Entwurfskriterien könnte man da sicher auch anführen, aber ich meine eher möglichst kleine Schnittstellen, eindeutige Schnittstellen usw. Kennen Sie denn Entwurfsprinzipien?
Klar. Abstraktion, Kapselung, Information Hiding, Kohäsion und Kopplung, Open-Closed Prinzip.

Nehmen wir mal Kohäsion und Kopplung. Was bedeutet denn das?
Kohäsion ist der innere Zusammenhalt einer Klasse, also wie stark die Methoden und Variablen einen gemeinsamen Zweck haben. Wenn eine Klasse viele Methoden hat, die nichts miteinander zu tun haben, dann hat sie eine geringe Kohäsion. Und Kopplung ist die äußere Abhängigkeit einer Einheit von anderen Einheiten.

(Hier erinnere ich mich nicht mehr genau wie es weiterging)

Wenn wir hier mal rübergehen zum Messen. Da haben wir ja auch Metriken kennengelernt. Was ist denn eine Metrik?
Eine Abbildung einer Eigenschaft eines SW-Systems auf eine Skala.

Was kann ich denn da alles abbilden?
Alles was man möchte. Ergebnisse die das Programm liefert, einzelne Zahlen...

... Kann ich nur Produkteigenschaften abbilden?
Auch den Prozess.

Genau.
Eine Prozessmetrik wäre z.B. der Fertigstellungsgrad.

Richtig, ich kann also Produkteigenschaften und Prozesseigenschaften abbilden. Wenn wir uns Kohäsion und Kopplung nochmal ansehen, kennen Sie da Metriken?
Ja da gibt es die Metrik-Suite von C&K, die definiert für die Kopplung CBO, und für die Kohäsion LCOM. (Hab dann angefangen CBO und LCOM zu erklären).

So genau wollen wir das ja auch nicht wissen. Können Sie sich vorstellen, wie man LCOM verbessern könnte?
Hmm...

Was betrachtet man denn da?
Die Benutzung der Instanzvariablen.

Genau, aber da gibt es ja recht simple Beispiele, wo die Metrik fragwürdige Resultate liefert.
Hmm...

Wir haben ja in so einer Klasse nicht so viel. Die Variablen und die Methoden. Was könnte man da noch betrachten?
Die Benutzung der Methoden untereinander würde sicher etwas verbessern, wenn man z.B. einen Getter auf eine Variable hat, den alle Methoden verwenden. Dann würde die Metrik nur eine Methode registrieren, die die Variable verwendet, und dann sagen dass die Klasse eine geringe Kohäsion hat, obwohl sie eigentlich stark zusammenhängt.

Genau. Damit hätte man Probleme mit Gettern und Settern beseitig. Wenn wir uns die Objektorientierung genauer ansehen, dann gibt es da ja z.B. Klassen. Was ist denn eine Klasse?
Klassen sind die Begriffe der Anwendung. Sie fassen Ähnlichkeiten verschiedener Gegenstände in einem Begriff zusammen.

Ja. Dann haben wir da noch Objekte, und Vererbung. Was macht man denn mit Vererbung?
Damit modelliert man Generalisierung und Spezialisierung, und bildet Hierarchien der Begriffe der Anwendung.

Wie kann man denn vererben?
Man kann Erweitern, Definieren und Redefinieren. Redefinieren kann man bei strikter Vererbung nicht. Wenn man ein abstrakte Methode deklariert, und die Unterklasse implementiert die Methode, dann ist das Definieren.

Ja. Kann man Vererbung denn nur sinnvoll einsetzen?
Wir haben das Substitutionsprinzip kennengelernt, das einem eine Hilfe gibt wann man sinnvoll Vererbung einsetzt, aber man kann Vererbung auch total sinnlos einsetzen.

Wie zum Beispiel?

Wenn man sich wahllos Code zusammenkopiert, den man gerade braucht. Die Implementation Inheritance.

Was ist denn daran schlecht?
Das bringt zwar kurzfristig einen Zeitgewinn, weil man den Code nicht neu schreiben braucht, aber später kriegt man Probleme, wenn man etwas ändern möchte. Dann wirkt sich eine Änderung an einer Klasse auf Klassen aus, von denen man es nicht erwartet hätte.

Genau. Damit erspart man sich später Kosten in der Wartung. Wenn wir uns da noch die analytischen Maßnahmen angucken, hatten wir ja das Testen. Wie testet man denn ein System?
Es gibt verschiedene Testverfahren, Blackbox-Test, Whitebox-Test zum Beispiel.

Nehmen wir an, Sie werden beauftragt ein System zu testen, wie machen Sie das?
Dann brauche ich erstmal die Soll-Resultate, die möglichen Eingaben, dann kann ich z.B. einen Blackbox-Test machen.

Alle Eingaben können Sie aber schlecht testen. Wie kann man das denn eingrenzen?
Da haben wir Äquivalenzklassenbildung besprochen.

Was ist das?
In Äquivalenzklasse fasst man Eingaben zusammen, die gleiche Fehler finden. Dann testet man nicht alle Eingaben, sondern nur aus jeder Äquivalenzklasse einen Repräsentanten.

Damit schränken Sie die Eingaben schonmal ein. Wie testen Sie denn jetzt das System?
Gegen die Anforderungen.

(Hier ging es noch ein bisschen weiter um Tests)

Kommen wir zum Projektmanagement. Was macht man da so?
Vor allem planen. Kosten, Leistungen, Ressourcen, Termine. Man startet und beendet ein Projekt.

Was ist denn ein Projekt?
Eine einmalige, zeitlich begrenzte Tätigkeit mit dem Ziel, Kundenwünsche in ein Software-Produkt umzusetzen.

Stellen Sie sich vor, sie sind Projektleiter. Was sind da Ihre Aufgaben?
Vor allem Kommunikation, Planung und Kontrolle. Ein Projektleiter muss das Projekt vor Einflüssen des Umfelds schützen, planen und das geplante kontrollieren, z.b. anhand von Metriken, die mir Aussagen über den Fortschritt und die Leistung des Projekts geben.

Bei der Terminplanung hatten wir uns Balkenpläne angesehen. Was ist denn das?
Das ist ein Plan, der einzelne Arbeitspakete auf einer Zeitleiste zeigt, und die Arbeitspakete können mit verschiedenen Abhängigkeiten zueinander stehen. Jedes Paket hat einen Anfangs- und einen Endzeitpunkt. Z.B ein Gantt-Chart.

Ja. Welche Information bekomme ich denn aus so einem Balkenplan?
Eine Übersicht der Startzeiten, Längen der AP.

Und was noch?
Ich kann auch noch kritische Arbeitspakete identifizieren.

Was ist denn ein kritisches AP?
Das ist ein AP mit einem Gesamtpuffer von 0. Da sind also frühester Startzeitpunkt und spätester Startzeitpunkt identisch, genauso die Endzeitpunkte. Ich kann es also nicht verschieben. Dann kann ich noch einen Kritischen Weg sehen.

Oder mehrere. Was ist das?
Eine Folge von ausschliesslich kritischen Arbeitspaketen.

Dankeschön. Warten Sie bitte kurz draußen.

Literatur

Bis auf Softwaretechnik sind die Folien zu den Vorlesungen perfekt. Die Folien zur Vorlesung Softwaretechnik von Prof. Nagl haben mich eher verwirrt, als dass sie mir geholfen hätten. Vieles aus der Vorlesung findet man auch in den Folien zu OOSK und SQS, und im Buch Software Engineering (s.u.). Neben den Folien, Prüfungsprotokollen, Wikipedia, Google, und anderen Internetseiten habe ich folgende Bücher benutzt:

***Software Engineering*; J. Ludewig, H. Lichter**
: Sehr empfehlenswertes Buch, besonders als Nachschlagewerk zur Vertiefung von einzelnen Themen, und als Ersatz für das schlechte Material zur Softwaretechnik-Vorlesung.

***Software-Qualitätsmanagement in der Praxis*; E. Wallmüller**
: Empfehlenswert für den Teil Software-Projektmanagement, den Rest findet man auch im Buch *Software Engineering.*

***Entwurfsmuster*; E. Gamma, u.a.**

Die Erklärungen zu manchen Entwurfsmustern in den Folien fand ich verwirrend, hier findet man Klarheit, mit Beispielen und gut strukturiert.

Weitere Bücher, die zwar bei den Prüfungsthemen nicht weiterhelfen, dafür aber beim Lernen generell sehr hilfreich sind:

***Effektives Lernen*; H. Dahmer, J. Dahmer**

Thema des Buchs ist zu lernen wie man lernt. Welche Formen des Lernens es gibt, wie gut man mit den verschiedenen Formen Gelerntes behält, wie man sich motiviert, und viel psychologisches Grundwissen zum Thema Lernen. Die Autoren stellen das Zettelkastenprinzip vor, was besonders gut ist zum Vokabellernen, also nicht unbedingt für diese Prüfung, aber auch bei den Prüfungsthemen gibt es „Vokabeln“, die man so sehr gut lernen kann.

***Keine Angst vor Prüfungsangst*; H. Knigge-Illner**

Thema des Buchs sind Strategien zur optimalen Prüfungsvorbereitung. Diese Strategien helfen auch Menschen ohne Prüfungsangst, also nicht vom Titel abschrecken lassen. Besonders empfehlenswert sind die Teile Arbeitsplanung, Zeitmanagement und Strukturierendes Lernen. Ausserdem enthält das Buch ein komplettes Kapitel über Training für mündliche Prüfungen.