Thema, Ziele: Software-Engineering, Softwarefehler, Vorgehensmodelle

Aufgabe 1: Verständnisfragen

a) Bei einer Diskussion über Software-Entwicklung, behauptet ein Informatik-Student, dass bei den Vorgehensmodellen (Methoden) bei denen die UML verwendet wird, der Übergang zwischen Analyse und Design fliessend sei, da bei beiden Tätigkeiten die gleiche Notation (Klassendiagramme etc.) verwendet würde.

Ist diese Behauptung richtig? Begründen Sie Ihre Antwort.

Diese Behauptung ist falsch, da die Ziele verschieden sind. Ziel der (Anforderungs-) Analyse ist Erkenntnisgewinn – WAS soll gebaut werden. Ziel des Design ist hingegen ein Konzept zu entwerfen, WIE das Softwaresysten gebaut werden soll.

b) Eine berühmte Aussage bezüglich Software-Projekten ist:

"adding manpower to a late software project makes it later" (Fred Brooks, 1975)

Erklären Sie diese Aussage, Brooks's law genannt, und begründen Sie, warum sie auch heute, 35 Jahre später für die Softwareentwicklung immer noch richtig ist.

Wenn Leute zu einem Softwareprojekt neu hinzukommen, müssen sie in der Regel zuerst darüber in Kenntnis gesetzt werden. Das hat einen erheblichen Kommunikations- und Koordinationsaufwand zur Folge. Dadurch werden die bisherigen Projektmitarbeiter zusätzlich belastet, weshalb sie weniger Zeit für ihre restlichen Aufgaben zur Verfügung haben, als ohne die neuen Mitarbeiter.

Erklären Sie allgemein bei welcher Art von Projekten diese Regel nicht gilt und geben Sie ein konkretes Beispiel für ein solches Projekt an.

Bei allen Projekten mit geringem Kommunikations- und Koordinationsaufwand (geringe Komplexität). Zu Beispiel beim Bau einer Häusersiedlung können problemlos so viele Maler oder Elektriker eingesetzt werden, wie Häuser gleichzeitig zu bauen sind, solange sie sich nicht gegenseitig behindern.

c) Eine weitere berühmte Aussage von Fred Brooks bezüglich Software-Projekten lautet:

"there is no silver bullet" (Fred Brooks, 1986)

Erklären Sie auch diese Aussage. Was meint man hier mit "silver bullet"?

"Silver Bullet" kann hier mit "Wunderwaffe" übersetzt werden. Eine Wunderwaffe, die einem ermöglicht auch hochkomplexe Systeme und Projekte, wie sie häufig bei der Softwareentwicklung vorkommen, zu erstellen.

d) Welche Arten von Komplexität unterscheidet Brooks? Erklären Sie diese. Welche kann bzw. können beeinflusst werden?

Komplexität verursacht durch essentielle und kontingente (engl. accidential) Eigenschaften. Essentielle Eigenschaften sind gegeben durch das Problem selbst. Sie können nicht beliebig reduziert werden. Kontingente Eigenschaften werden durch uns selber verursacht. Durch gute Arbeitsmethodik und geschickt gewählte Werkzeuge kann deren Komplexität reduziert werden.

e) Ihr Chef, der alles besser weiss, obwohl er noch nie ein eigenes Programm geschrieben hat, verlangt, dass beim nächsten Software-Projekt das Programm für PC's in Assembler geschrieben wird, weil er gehört hat, dass Assembler-Programme schneller seien als z.B. in C++ geschriebene.

Welche Art der Komplexität gemäss Brooks's wird dadurch erhöht? Warum wird diese erhöht?

Es wird die kontingente Komplexität erheblich erhöht. Die Mitarbeiter müssen sich nun noch mit den Tücken der Assemblerprogrammierung befassen, wodurch die Produktivität erheblich sinkt.

Was tun Sie vernünftigerweise?

Mit dem Chef das Gespräch suchen. Falls er bei seiner Idee bleibt und damit zu rechnen ist, dass er noch längere Zeit auf seinem Platz bleibt, nach einem anderen Arbeitsplatz Ausschau halten.

f) Erklären Sie, warum das Original-Wasserfallmodell in der Praxis nicht funktioniert.

Weil alles beim ersten Mal richtig gemacht werden muss!

In welchen speziellen Fällen kann es och funktionieren?

Bei einfachen Projekten, bei denen mal alles zum Voraus kennt. Oder bei (kleinen) Projekten, bei denen man ebenfalls alles schon kennt, weil man sie jetzt zum zweiten Mal macht, nur diesmal richtig!

g) Erklären Sie den/die Unterschied(e) zwischen dem modifizierten Wasserfallmodell (mit Rückweg zur vorhergehenden Phase) und den anderen iterativen und inkrementellen Vorgehensmodellen.

Unterschied 1: Es wird (auch) beim modifizierten Wasserfallmodell davon ausgegangen, dass beim ersten Mal, alles richtig gemacht wird. Bei den anderen Modellen ist das nicht der Fall.

Unterschied 2: Beim modifizierten Wasserfallmodell kann nur jeweils zur vorhergehenden Phase zurückgegangen werden (Korrekturpfad). Bei den anderen Modellen wird nie rückwärts, immer nur vorwärts gegangen. Evt. aufgetauchte Probleme werden dann beim "nächsten Umgang" behoben.

Beispiel: 4 Tätigkeiten a, b, c, d. Wasserfall: abcd, ababcd, abcdcd, … → Rücksprünge
Andere Modelle: abcd abcd abcd … → Zyklen.

h) Worin liegt das Hauptproblem bei der Anwendung des Wasserfallmodells mit externen Kunden als Auftraggebern?

Die in der Analyse-Phase erstellten Pflichtenhefte (Anforderungsspezifikationen) sind umfangreich, detailliert, langweilig und für den Kunden oft unverständlich.

Folgen: Kunde muss Pflichtenheft abzeichnen, welches er nicht versteht und sieht das Produkt erst wenn es fertig ist.

Typische Reaktion: "I know, this is what i asked for, but it's not, what I really wanted."

i) Bei welchen Vorgehensmodellen kann die objektorientierte Software-Entwicklung eingesetzt werden?

Eigentlich bei allen. Ob objektorientiert oder nicht objektorientiert programmiert wird, hat grundsätzlich nicht mit dem Vorgehensmodell zu tun.

Aufgabe 2: Berühmte Softwarefehler … oder aus Fehler lernen

Obwohl in der Raumfahrt bei der Software-Entwicklung konsequent Software-Engineering-Methoden eingesetzt werden, kann es auch hier zu spektakulären Unfällen kommen.

2.1 "The Explosion of the Ariane 5" – 1996

Einen solchen Unfall beschreibt der Artikel *"The Explosion of the Ariane 5"* auf der nächsten Seite.
Quelle: "http://www.ima.umn.edu/~arnold/disasters/ariane.html".

Lesen Sie diesen Artikel und beantworten Sie anschliessend die untenstehenden Fragen.

Beachten Sie, dass aus diesem Artikel nicht hervorgeht, dass das bei der Ariane 5 für das INS ("Inertiales Navigationssystem") verwendete Softwaremodul weitgehend von der Ariane 4 übernommen wurde. Dort lief diese Software lange Zeit erfolgreich und wurde deshalb auch als sicher eingestuft.
(Details siehe "http://de.wikipedia.org/wiki/Ariane_V88")

Fragen:

1. Was war der Fehler der zum Unfall mit der Ariane 5 führte?

   Eine 64-Bit Gleitkommazahl wurde in eine 16 Bit int-Zahl konvertiert. Da dabei der int-Wertebereich überschritten wurde, löste dies eine System-Exception aus, welche zum Ausfall des Softwaresystems führte, was schlussendlich zum Absturz führte. (Details siehe "http://de.wikipedia.org/wiki/Ariane_V88".)

2. Was war Ihrer Meinung nach die Ursache für diesen Fehler?
   Die zuwenig sorgfältige geprüfte Übernahme der Ariane 4 Softwaresystems für die Ariane 5. Die Ariane 5 war wesentlich "kräftiger gebaut" und dadurch schneller als die Ariane 4.

3. Wie hätte sich dieser Fehler Ihrer Meinung vermeiden lassen können?
   Die unterschiedlichen Betriebsbedingungen von Ariane 4 und 5 hätten besser untersucht und berücksichtigt werden sollen.

4. Welche Schlüsse ziehen Sie aus diesem Fehler in Bezug auf die Entwicklung von Software im Allgemeinen und das Software Engineering im speziellen (Entwicklung, Testen, Fehler, Wartung, etc.)?
   Grundsätzlich: Man kann nie vorsichtig genug sein. Bei der Wiederverwendung von Software müssen die Rahmenbedingungen klar spezifiziert und die Software muss entsprechend geprüft und getestet werden.

5. In welcher Projektphase trat dieser Fehler Ihrer Meinung nach ein?
   Ein Entscheid, dass eine bestehende Software wiederverwendet wird, erfolgt grundsätzlich in der Grob-Design-Phase. Vermutlich ist der Fehler aber bereits in der Analyse-Phase passiert, weil man (fälschlicherweise) angenommen hat, dass es sich um das exakt gleiche Teilproblem wie bei der Ariane 4 handelt. Damit wurde die Design-Entscheidung erheblich "vorgespurt". – Grundsätzlich wurden den Randbedingungen unter denen die für Ariane 4 geschriebene Software korrekt funktioniert zuwenig Beachtung geschenkt wurde.

6. Was hat dieser Fehler gekostet? (nur direkte Kosten, ohne Entwicklungs- und Folgekosten)
   Gemäss oben erwähntem Artikel rund 500 Millionen USD. Gemäss deutschem Wikipedia ca. 290 Millionen Euro und gemäss englischem Wikipedia mehr als 370 Millionen USD.

2.2 Verlust der NASA-Sonde "Mars Climate Orbiter" (MCO) – 1999

Lesen Sie den folgenden Wikipedia-Beitrag "http://de.wikipedia.org/wiki/Mars_Climate_Orbiter" und beantworten Sie anschliessend folgende Fragen.

1. Wie lange dauerte der Flug dieser NASA-Sonde bis der Fehler sich bemerkbar machte?
   Vom 11. Dezember 1998 (Start) bis am 23. September 1999 (Mars erreicht). Also gut 9 Monate.

2. Was war der Fehler der zum Verlust führte?
   Die NASA und der Lieferant der Navigationssoftware (Lockheed Martin) verwendeten bei der Entwicklung der Software – ohne es zu merken(!) – für die physikalische Grösse der Kraft unterschiedliche Masseinheiten. Die NASA arbeitete mit der SI-Einheit "Newton", Lockheed Martin mit der angloamerikanischen Masseinheit "pound-force". Da ein pound-force ca. 4.45 Newton ist, wurde die Sonde zu Nahe an den Mars geführt, was zu deren Absturz führte.

3. Angenommen Sie würden die Fehlerursache nicht kennen. Hätten Sie sich dann vorstellen können, dass in der Raumfahrt solch "dumme" Fehler vorkommen können?
   Eigentlich nicht, aber "was schiefgehen kann, geht schief" (Murphy's Law).

4. In welcher Projektphase trat dieser Fehler Ihrer Meinung nach ein?
   In der Design-Phase. Beim Festlegen WIE die Aufgabe gelöst werden soll, müssen auch die zur Anwendung gelangenden Masseinheiten festgelegt werden.

5. Was hat dieser Fehler gekostet?
   Wikipedia: "The cost of the mission was $327.6 million total for both orbiter and lander, $193.1 million for spacecraft development, $91.7 million for launching it, and $42.8 million for mission operations.".