Gaida, Lässer, Schwendinger

Lerntagebuch

LV: Softwareprozesse und Softwarequalität

University of Applied Sciences: Vorarlberg

Department of Computer Science
Dipl. Inform. Bernd G. Wenzel

Dornbirn, 2014

Abstract

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Inhaltsverzeichnis

Listings

Abbildungsverzeichnis

Todo list

1 LV 1 am 21.02.2014

1.1 Organisatorisches

Zu Beginn dieser Vorlesung wurden zuerst die organisatorischen Randbedingungen für diese Lehrveranstaltung geklärt.
Dabei wurde entschieden, dass anstelle einer Prüfung ein Lerntagebuch in Gruppen zu max. drei Personen erstellt wird. Das Lerntagebuch enthält zusätzlich ein Glossar mit den wichtigsten Begriffsdefinitionen rundum das Thema Softwareprozesse und Softwarequalität.

1.2 Motivation des Software-Engineering

Als Einstieg in das Thema dienten ein paar nicht ganz ernst gemeinte "Gesetze". Diese Gesetze sollten uns verdeutlichen, dass Software-Engineering nicht so trivial und einfach ist, wie es oft den Anschein hat. Wie schon die Folgerung von Kleinbrunner besagt

> "Wenn eine Programmieraufgabe leicht aussieht, ist sie schwer."

Es wurden unter anderem die "Gesetze"von Gutterson, Farvour, Brunk, Munbright und vielen anderen vorgestellt.

Um die Bedeutung von qualitativ hochwertiger Software zu verdeutlichen, stellte der Vortragende einige Probleme in bekannten Softwareprojekten wie beispielsweise der Verlust einer Segelyacht mit Crew vor der mexikanischen Pazifikküste aufgrund eines Softwarequalitätsfehlers.

Abschließend erhielten wir die Aufgabe bis zur nächsten Vorlesung ein Programm zu entwickeln zur Berechnung der reellen Lösungen einer quadratischen Gleichung. Ziel bei dieser Aufgabe ist es auf die Qualität des Codes zu achten.

2 LV 2 am 07.03.2014

Zu Beginn wurden kurz die Lösungen der Hausaufgabe besprochen. Es wurde verdeutlicht, dass sogar so eine einfache Aufgabenstellung von einigen Studenten anders implementiert wurde, als ursprünglich gedacht.

Aufbauend auf LV 1 wurden wichtige Begriffe wie Software, Software-Engineering, Softwareprozess usw. definiert. Die wichtigsten Definitionen, auch von zukünftigen Vorträgen, sind im Glossar angeführt. Dabei wurde auch das Vorgehensmodell und die Methode des Software-Engineerings vorgestellt.

Bei der Methode des Software-Engineerings geht es um einen strukturierten Ansatz für die Software-Entwicklung mit Ziel eine möglichst hohe Qualität der Software bei gleichzitig geringen Entwicklungskosten zu erreichen.

Das Vorgehensmodell definiert wie der Softwareentwicklungsprozess abläuft. Bekannte Modelle sind dabei Scrum, Wasserfall oder das V-Modell. Dabei stellt sich heute oft die Frage, ob man sich für einen klassisches oder agiles Vorgehensmodell entscheidet. Der größte Unterschied zwischen agil und klassisch ist, dass beim Agilen Ansatz der Kunden viel mehr involviert ist und öfters das Feedback von ihm in die Entwicklung mit einfließt. Auch sind die Iterationen beim Agilen Vorgehen meist kürzer.

Anschließend behandelten wir das Thema der soziotechnischen Systeme. Dabei wurde der Begriff genauestens definiert, die wesentlichen Eigenschaften und der Lebenszyklus sowie die Umfeldfaktoren vorgestellt.

Nach den soziotechnischen Systemen behandelten wir das Thema der kritischen Systeme. Dabei geht es um Systeme dessen Ausfall oder Fehlfunktion sich auf die Umgebung in Form von z.B. wirtschaftlichen Verlusten, Schäden an der Umwelt auswirken. Wichtige Faktoren dabei sind die Verfügbarkeit, Zuverlässigkeit, Betriebssicherheit und Systemsicherheit. Der Sammelbegriff für diese Punkte heißt Verlässlichkeit.

Zum Schluss der Lehrveranstaltung wurden die ethnischen Herausforderungen beim Software-Engineering behandelt. Dabei geht es um Vertraulichkeit gegenüber dem Arbeitgebern und Kunden, um den Schutz von geistigem Eigentum und den Schutz vor Computermissbrauch. Es gibt auch einen Knigge für das professionelle Verhalten des Software-

Engineering definiert durch ACM/IEEE.

2.1 Anforderungsanalyse

Bei der Anforderungsanalyse ist es wichtig die Projektbeteiligten eindeutig zu identifizieren. Hierbei ist es auch wichtig deren Erwartungen festzustellen. Sprachbarrieren zu beseitigen sowie die Unterscheide im Bezug auf Anforderungen zu aufzuklären. Anbei sehen sie einige Beispiele solcher Beteiligten.

- Endanwender
- Kooperationspartner
- Systemadministratoren
- Sicherheitsbeauftragte
- Marketingabteilung andere betroffene Mitarbeiter
- Hard- und Softwareentwickler Manager
- Technologie-Experten
- Anwendungsexperten
- Betriebsratsmitglieder
- Behörden

3 LV 3 am 21.03.2014

3.1 Anforderungsanalyse

3.1.1 Sammeln von Anforderungen

Um Anforderungen zu sammeln gibt es verschiedene Möglichkeiten:

- Fragebogen
- geschlossene Interviews
- offene Interviews

Sehr gut geeignet ist eine Mischung aus allen drei Varianten. Wichtig ist, dass man die Leute interagieren lässt - sprich man benötigt ein gewisses Maß an Flexibilität und muss sehr gut zuhören können.

Bei der Sammlung von Anforderungen treten einige Probleme auf. Eine sehr große Hürde stellt der Jargon und das implizite Wissen dar. Die Leute gehen davon aus, dass jeder ihren Jargon (mit allen Begrifflichkeiten) kennt und versteht. Das implizite Wissen ist den Leuten meistens nicht bewusst (z.B. aus Gewohnheit) und wird von ihnen zurückgehalten. Beim nicht expliziten Wissen, wissen die Leute, dass sie es haben und gehen davon aus, dass jeder dieses Wissen hat, da es für sie selbstverständlich ist.

Wichtig ist, dass man nicht sofort mit Interviews anfängt, sondern sich erst in die Domäne einliest. Andernfalls ist das Ergebnis von den Interviews nicht zu gebrauchen, da einfach das Verständnis für die Domäne nicht vorhanden ist.

Szenarien stellen eine weitere Technik zur Sammlung von Anforderungen dar. Es ist zu empfehlen bei der Erstellung von Szenarien Ad-hoc vorzugehen und zur Dokumentation UML-Diagramme zu verwenden. Wichtig dabei ist es auch auf die Details zu achten, um das Gesamtbild besser verstehen zu können.

Eine weitere Technik zur Sammlung von Anforderungen ist die Ethnografische Methoden. Durch diese Variante kann man durch Beobachten von Personen Anforderungen herausgefunden. Dabei wird diese Person neutral und ohne jegliche Interaktion mit dieser

beobachtet. Es müssen stets stillschweigend Notizen gemacht werden und am Ende wird mit einem Experten das Beobachtete diskutiert. Es ist von Vorteil dies mit der jeweiligen beobachteten Person zu machen und nicht mit deren Chef, da sonst weitere Beobachtungen durch wenig Kooperation der Mitarbeiter behindert werden könnten.
Durch diese Variante kann man sehr gut implizite, nicht explizite Anforderungen und Abweichungen erkennen. Auch die Anforderungen der Benutzer können dadurch sehr gut ermittelt werden.

3.2 Klassifizierung von Anforderungen

Bei der Klassifizierung hat man die Aufgabe die Struktur der Anforderungen aus den unterschiedlichen Quellen zu definieren. Es ist wichtig Duplikate sowie Synonyme zu erkennen, die Beziehungen zwischen den Anforderungen zu definieren und eine Gruppierung der Anforderungen durchzuführen.

3.3 Validierung von Anforderungen

Es gilt zu prüfen, ob die Anforderungen gültig (filtern von falschen und unnötigen Anforderungen), konsistent, vollständig (sehr schwer zu prüfen), realisierbar (genügt das Budget, Zeit? ist die Technologie geeignet?) und verifizierbar sind.

Um dies Überprüfung durchzuführen kann man ein Anforderungsreview machen, einen Prototypen entwickeln oder Testfälle erzeugen.

Für ein Anforderungsreview benötigen wir aus allen wichtigen Gruppen der Stakeholder einen Vertreter, einen Systemarchitekten und einen Vertreter der Softwareentwickler (diese müssen am Ende ja wissen, was sie entwickeln sollen). Die Durchführung erfolgt durch den jeweiligen Anwendervertretern. Es werden dabei alle Anforderungen diskutiert. Durch das Anforderungsreview können Konflikte, Widersprüche, Fehler und Versäumnisse ermittelt werden. Sie sollten in einem Review-Bericht festgehalten werden. Die Prüfung auf Konsistenz und Vollständigkeit fallen unter die Kategorie der Notwendigen Prüfungen. Prüfungen bezüglich der Verifizierbarkeit, Verständlichkeit sowie Anpassungsfähigkeit sind Optionale Prüfungen.

3.4 Priorisierung von Anforderungen

Die Priorisierung ist speziell bei einer agilen Vorgehensweise sehr wichtig.

Der Grundgedanke ist durch Phasenweise Implementierung das Leben der Anwender durch kontinuierliche Softwareupdates zu erleichtern. Es minimiert auch das Risiko der Einführung der Software.

Die Prioritätssteuerung erfolgt über eine Bewertungsformel. Es wird jeweils eine Bewertung zu jeder einzelnen Anforderung durch das Nutzwertanalyseteam und das Kostenanalyseteam erstellt. Beide Teams bewerten völlig unabhängig voneinander und ohne Kommunikation zwischen den Teams. Nach der Auswertung erhält man eine Liste der Anforderungen welche für die nächste Ausbaustufe abgearbeitet werden sollten.

3.5 Dokumentation und Verwaltung

3.5.1 Modelle

Bieten eine detaillierte und formalisierte Dokumentation über den Ist-Zustands und Soll-Zustands. Es werden auch Sichten von Kunden oder anderen Abteilungen auf beide Zustände festgehalten.

Kontextmodelle

Mit diesem Modell werden die Systemgrenzen hinsichtlich des Gesamtsystems definiert. Hier ist es wichtig den Kontext zu definieren. Neben dem Gesamtsystem werden die technischen Systeme betrachtet. Bei den technischen Systemen ist die Definition des Scopes besonders wichtig. Wichtig bei der Erstellung eines Kontextmodelles die die Auswahl der Verfügbaren Notationen (UseCase-Diagramm, Architekturdiagramm, etc.).

Verhaltensmodelle

Es werden Daten- und Ereignis-orientiert die Abläufe im System definiert. Es gibt auch Mischformen (z.B. eine Banküberweisung). Zur Veranschaulichung dienen Diagramme wie Datenflussdiagramme, Prozessdiagramme, Zustandsdiagramme, Sequenzdiagramme oder Szenarios.

Datenmodelle

Für die Darstellung der Datenmodelle gibt es ER-Diagramme, UML-Klassendiagramme, XML-Schema- OWL und diverse andere Diagrammtypen. Sie dienen zur logischen Definition der Datenstrukturen hinsichtlich transients und persistents. Auch hier ist die Auswahl der verfügbaren Notation von großer Wichtigkeit.

Ein wichtiges Werkzeug bei den Datenmodellen ist das Data Dictionary. Hier speichert man sich die Klassen und Attribute. Man speichert sich zu den Entitäten gewisse Eigenschaften wie zum Beispiel den Namen, Beschreibung und Typ.

Objektorientierte Modellierung

Bei der Objektorientierten Modellierung geht es um die Vereinigung der Funktionalität von Daten und Verhaltensmodellierung. Hier ist die Rede von Daten, Datenflüssen sowie Ereignissen. Als Standard Notation wird hierbei UML verwendet. Bei der Modellierung werden folgende Highlights verwendet.

- Klassifikationshierarchien
- Aggregationsbeziehungen
- Operationen (Objektverhalten)

Strukturierte Methoden

Die strukturierte Methode beschreibt eine detailliert definierte Vorgehensweise bei der Software-Entwicklung. Normalerweise basierend auf einem Satz von Diagrammtypen in Verbindung mit Regeln und Richtlinien. Beispiele für Strukturierte Methoden sind:

- JSP
- V-Modell
- RUP

Durch den Einsatz von strukturierten Methoden wird gewährleistet, dass immer eine gewisse Qualität erreicht wird.

Die Nachteile sind mangelnde Unterstützung nicht-funktionaler Anforderungen und die Anwendbarkeit für konkrete Probleme. Letzteres benötigt sehr viel Erfahrung um die Entscheidung *Wann verwende ich welche Methode* korrekt treffen zu können. Es muss auch beachtet werden, dass die Anpassung einer Problemklasse sehr schwierig ist und dass ein enormer Dokumentationsaufwand oft das Wesentliche verdeckt. Eine weitere Gefahr bei

dieser Methoden ist der hohe Detaillierungsgrad, welcher das Verständnis für das System erschwert.

3.5.2 Anforderungsmanagementsystem

Erfahrungsgemäß sind die Anforderungen am Anfang eines Projektes stets unvollständig und sie ändern sich im laufe eines Projektes fortlaufend. Die Anforderungen helfen allerdings dabei das Problem bzw. die Probleme besser zu verstehen. Deshalb ist der Umgang mit Anforderungen und ihren Änderungen von enormer Bedeutung. Hier ist es wichtig auf eine gute Dokumentation zu achten und die richtigen Werkzeuge zur Unterstützung einzusetzen. Vor allem die Aufgabenübersicht (Anforderungssammlung, Beziehungsmanagement, etc) ist nicht zu vernachlässigen.

Man muss bei den Anforderungen zwischen verschiedenen Typen unterscheiden. Es gibt dauerhafte Anforderungen, die relativ stabil sind und mit dem Kernbereich des Systems verbunden. Sie können aus standardisierten Modellen des Anwendungsbereichs abgeleitet werden. Daneben gibt es veränderliche Anforderungen mit einer sehr hohen Änderungswahrscheinlichkeit. Es müssen wirtschaftliche, technische und gesetzliche Randbedingungen beachtet werden.

4 LV 4 am 25.04.2014

4.1 Dokumentation und Verwaltung

4.1.1 Pflichtenheft

Das Pflichtenheft dient zur Zusammenstellung der vollständigen und detaillierten Benutzer- und Systemanforderungen. Wichtig ist, dass das Pflichtenheft vollständig ist, wobei dieses Ziel nur sehr schwer zu erreichen ist. Bei Projekte der Europäischen Union ist es sogar nicht mehr erlaubt Änderungen am Pflichtenheft nach Vertragsabschluss durchzuführen. Dies ist durch ein Wettbewerbsgesetz geregelt.
Ein Pflichtenheft kann aus mehr als einem Dokument bestehen und wird für Kunden, Manager, Entwickler, Tester, Warter und die Juristen geschrieben.
Ein beliebter Trick in diversen Ländern ist es, das Pflichtenheft via E-Mail abzuändern. Wenn diese Änderung allerdings nicht innerhalb von 14 Tagen widerrufen wird, dann ist sie Bestandteil vom Vertrag und muss entwickelt werden. Das gleiche gilt für Besprechungen, die schriftlich Protokolliert werden. Sollten diese Änderungen dann nicht berücksichtigt werden, kann ein Rechtsstreit eintreten.
Im Pflichtenheft erfolgt eine Klassifizierungen der Informationen hinsichtlich der Anforderungen (mandatory requirements), Wünsche (optional requirements), informative Bestandteile (dienen zum besseren Verständnis, sind unverbindliche Informationen) und Warnungen (Abbildungen, Tabellen, Anhänge als informative Bestandteile). Um informative Bestandteile wie ein Anhang zu einem vertraglicher Bestandteil zu machen, muss es mit Normative gekennzeichnet werden.
Der Aufbau vom Pflichtenheft kann auch nach vordefinierten Normen und Standards erfolgen. Ein Beispiel für eine solche Norm ist die IEEE/ANSI 830-1998 Norm. Wichtig ist das auch die Normen und Standards von Referenzen zum Bestandteil des Vertrages werden.

4.2 Spezifikation kritischer Systeme

Kritische Systeme sind jene, bei deren Benutzung durch das Auftreten eines Fehlers Gefahr besteht - zum Beispiel ein GPS-System oder ein Defibrillator in der Medizintechnik.

4.2.1 Anforderungen an kritische Systeme

Funktionalen Anforderungen unterstützen Definitionen von Funktionen zur Fehlerprüfung, Funktionen zur Wiederherstellung im Fehlerfall und Aspekte zum Schutz gegen Systemausfälle. Zu den nicht funktionale Anforderungen zählen die Systemzuverlässigkeit und die Systemverfügbarkeit. Es werden auch Negativanforderungen festgelegt, welche jene Verhalten beschreiben, das ein System auf keinen Fall zeigen darf beschreiben. Eine Negativanforderung wäre zum Beispiel das ein Flugzeug nicht schneller als 800km/h fliegen darf.

4.2.2 Risikomanagement

Beim Risikomanagement werden Gefahren bestimmt, klassifiziert und versucht deren Risiko zu verringern.
Prinzipiell gelten Murphy's Gesetze: Älles, was schiefgehen kann, wird irgendwann einmal schiefgehen.Alles, was schiefgehen kann, wird genau dann schiefgehen, wenn es den maximalen Schaden verursacht.öder Wenzels Korollar dazu "Murphy war ein Optimist."
Das Ziel beim Risikomanagement ist es, Risiken zu erkennen und dessen Auswirkungen zu minimieren. Risiken entstehen prinzipiell dort, wo Systemkomponenten aufeinandertreffen oder die Umwelt Einflüsse auf das System hat. Entgegenwirken kann man durch erfahrene Entwickler, Berater oder Fachleute des Anwendungsgebiets (siehe witziges Beispiel Ölschraube bei Scania).
Bei der Risikoanalyse und der Risikoklassifizierung spielen die Begriffe Unfallwahrscheinlichkeit, Schadenswahrscheinlichkeit und Schadenshöhe eine wichtige Rolle. Durch die Klassifizierung kann man abschätzen, ob ein Risiko vernachlässigbar ist oder ob es unannehmbar ist. Unannehmbare Risiken verursachen große Schaden, haben eine hohe Schadenswahrscheinlichkeit und sollten unter allen Umständen versucht werden zu vermeiden.
Um herauszufinden woher ein Risiko kommt wird es zerlegt um die Ursache für das Risiko zu entdecken. Hierfür gibt es viele verschiedene Techniken wie Reviews, Checklisten, Petri-Netze, Fehlerbaum sowie einige andere Techniken.
Um die Risiken minimieren zu können, werden in der Praxis die Techniken zur Risikovermeidung-, Risiko-Erkennung- und die Risikobeseitigungen- sowie die Schadensbegrenzungsstrategie kombiniert.

4.2.3 Betriebssicherheit

Wichtig ist es das System in Hinsicht auf dessen Betrieb zu sichern. Vor allem Schutzmechanismen und Ausfallsysteme sind von großer Bedeutung. Auch die Betriebssicherheit

im Laufe des Produktlebenszyklus ist zu beachten. Um Fehler in diesem Bereich klassifizieren zu können, gibt es verschiedene Klassen. Klasse 1 bis Klasse 5 werden dabei in der Software Industrie als Standard angesehen.

- Klasse 1: Fehler die andere Anwendungen betreffen können.
- Klasse 2: Fehler die Benutzer der selben Anwendung betreffen können.
- Klasse 3: Funktion steht ohne Workaround nicht zur Verfügung.
- Klasse 4: Funktion steht nur mittels Workaround zur Verfügung.
- Klasse 5: Verbesserungsvorschlag

4.2.4 Systemsicherheit

Bei der Systemsicherheit spielen vor allem die Bedrohungen eine wichtige Rolle. Die Bedrohungen können dabei in Kategorien unterteilt werden. (Direkte Bedrohungen, Indirekte Bedrohungen, etc.). Vor allem die Monopolartige Stellung einiger Unternehmen führt dazu das Sicherheitslücken im großen Stil auftreten können. Zudem gibt es keine 100

4.2.5 Zuverlässigkeit

Damit ein System zuverlässig ist, müssen sowohl die Hardware als auch die Software und die Benutzer zuverlässig sein. Die Zuverlässigkeit einer Software ist messbar. Dabei werden Messwerte bezüglich Systemfehler, Zeit zwischen Systemfehlern und ähnlichem verwendet. Vor allem die Zeit zwischen Systemfehler und Wiederverfügbarkeit des Systems ist wichtig für die Aussage über die Zuverlässigkeit eines Systems.
Die Vorgehensweise bei der Zuverlässigkeitsspezifikation ist es die möglichen Systemfehler zu ermitteln, deren Auswirkungen zu analysieren und mögliche Vorkehrungen zur Beseitigung dieser Risiken zu realisieren. Besonders wichtig ist es allerdings zu berücksichtigen, dass extreme Zuverlässigkeit nicht testbar ist.

4.2.6 Formale Spezifikationsmethoden

In die formale Spezifikationsmethoden wird schon seit langer Zeit gearbeitet. Der Durchbruch dieser Technik blieb allerdings bis heute aus. Die Gründe dafür sind neben der Entstehung von anderen Software-Engineering-Methoden auch die Marktveränderungen in diesem Bereich und die beschränkte Einsetzbarkeit der Methode. Allerdings ist diese Methode vor allem in den Bereichen des Luftverkehrs, der Raumfahrt und in der Medizintechnik weit verbreitet. Die Nutzung in Softwareprozessen kann entweder mittels

einem Linearen Prozess oder einem Parallelen Prozess erfolgen. Die Spezifikation von System-Schnittstellen kann mit unter allerdings recht Umfangreich werden. Auf Grund dieses Problems entsteht hier eine Chance für den Model Driven Architekuture Ansatz. Formale Spezifikationstechniken können als gute Ergänzungen zu informellen Techniken betrachtet werden. Vor allem im Kostenvergleich können sie überzeugen.

5 LV 5 am 23.05.2014

5.1 Besprechung Hausaufgabe - Anforderungen

5.1.1 Funktionale Anforderungen

- Ergebnis x element R : $ax^2 + bx + c = 0$
- kein Absturz bei komplexer Lösung
- vernünftige Reaktion auf Eingabefehler
  - a = 0
  - unvollständige Eingaben
  - $4ac > b^2$
- relativer max. Rundungsfehler von 0,1% mit Beweis (für die wirkliche Realisierung darf die Formel nicht 1 zu 1 programmiert werden. In der Realität wird eher eine Newton-Annäherung implementiert).
- es gibt immer ein definiertes Ergebnis (Ausnahmebehandlungen)
- keine Endlosschleife (wenn Newton angewendet wird, dann braucht es eine vernünftige Abbruchbedingung)
- Zeitbedarf max 1 msec
  - CPU der Ver
  - Rechner/Prozessor
  - max. 2x SQRT der Standardimplementierung
- Signatur: Name, Parameter, Ergebnistyp

Jetzt soll jeder seine Lösung anhand dieser Anforderungen bewerten. Die Lösung darf auch dahingehend verbessert werden, dass alle Anforderungen erfüllt werden.

5.2 Referenz-Architekturen

- ISO-OSI-Referenzmodell
- ECMA-Toaster (für CASE-Umgebungen)
- Schichtenarchitektur

Verteilte Systeme: Schichtenarchitektur, C/S ...

Echtzeit-Software etc wurde übersprungen.

5.3 Entwicklung - Entwicklung kritischer Systeme

Bei der Entwicklung kritischer Systeme, gibt es verschiedene (komplementäre) Strategien.

- Fehlervermeidung: zuerst wird überprüft und nicht erst gerechnet, wenn ein Fehler passieren kann
- Fehlerentdeckung: ??
- Fehlertoleranz: eine Situation, in der z.B. die Berechnung beginnt und während der Berechnung wird eine Fehlersituation erkannt und automatisch behandelt.

Je mehr Fehler behoben werden, umso teurer wird die Software.

Um Fehler zu erkennen, stehen folgende Techniken zur Verfügung:

- verlässliche Softwareprozesse
- Qualitätsmanagement
- formale Spezifikation
- statische Verifikation: mehrere Personen verifizieren
- starke Typisierung: Vorteil durch Unterstützung des Compilers
- sichere Programmierung: z.B. keine GoTos
- Ausnahmebehandlung: nur Sprachen verwenden, die das unterstützen
- geschützte Informationen

5.3.1 Fehlervermeidung

Die Fehlervermeidung beginnt bereits bei den Anforderungen. Diese müssen genau inspiziert werden (sind sie widersprüchlich?). Auch ein Anforderungsmanagement ist notwendig, da sich Anforderungen ändern.

Ebenfalls wichtig ist, dass mindestens ein Vier-Augen-Prinzip beim Code und Entwurf verfolgt wird.

Als nächsten Schritt gilt es Tests zu planen und zu managen.

Auch welche Konfigurationen erstellt und ausgeliefert werden, muss klar sein.

Sichere Programmierung

Im Rahmen der sicheren Programmierung, sind mögliche Gefahrenquellen Gotos, Rundungsfehler von Gleitkommazahlen, Pointer, dynamische Speicherallokierung, Parallelität, Rekursionen, Interrupts, Vererbung, Aliasing, keine Überprüfung von Arraygrenzen und Konfigurationsdaten.

Ausnahmebehandlung

Es sollten prinzipiell nur Sprachen verwendet werden, die eine vernünftige Ausnahmebehandlung bieten. Der Code wird dadurch wartbarer und Fehler lassen sich vermeiden.

5.3.2 Fehlertoleranz

Fehlertolerante Architekturen

Eine Variante ist die Verwendung von redundanter Hardware. Dabei ist es wichtig, dass auf jeder Hardware Software von unterschiedlichen Entwicklern laufen, um die Ergebnisse vergleichen zu können um so fehlertoleranter zu werden.

Eine andere Variante ist die Verwendung von sogenannten Wiederherstellungsblöcken.

5.4 Weiterentwicklung

5.4.1 Ursachen der Wartung

Wartung bedeutet nicht gleich Fehlerbehebung. Sie macht nur 17% bei der Wartung aus. Circa 65% machen Erweiterungen aus und weitere 18% dies Softwareanpassung an z.B. neue Hardware.

5.4.2 Wartungsaufwand

Oftmals wird in der Praxis das Entwicklerteam nach dem Release auf ein neues Projekt angesetzt und ein anderes Team liest sich in die Software ein und wartet sie dann bzw. die Wartung wird an eine andere Firma übertragen. Das ist unter anderem ein Grund für hohen Wartungsaufwand. Oft spielen auch die Fähigkeiten (z.B. mangelnde Erfahrungen im Anwendungsgebiet, der eingesetzten Technologie) der Entwickler, veraltete/fehlerhafte/nicht vorhandene Dokumentationen oder fehlendes Konfigurationsmanagement eine entscheidende Rolle.

Ebenfalls wichtige Einflussgrößen sind die Anzahl und Komplexität der Schnittstellen, die Anzahl der veränderlichen Anforderungen und die Geschäftsprozesse, in denen das system verwendet wird.

5.4.3 Weiterentwicklungsprozesse

Seite 35-39 (excl. Reengineering)

6 Glossar

- Software
- Software-Engineering
- Informatik
- Softwareprozess
- Software-Validierung (o Ob die Software für den Kunden einen Wert hat)
- Software-Verifizierung (o Übereinstimmung der Software mit den Spezifikationen)
- Qualität
- Technisches computer-basiertes System
- Soziotechnisches System
- Funktionale Anforderungen
- Nicht-Funktionale Anforderungen
- Technical Debt
- Verifikation
- Validierung
- ESSENCE Kernel
- Data Dictionary
- Scope: was man wirklich implementieren will
- Kontext: das ganze um den Scope herum.
- Domäne
- Szenarien
- Anforderung
- Anforderungsreview
- Stakeholder

- Systemgrenzen
- Kontext
- Scope
- Usecase
- UML
- Architekturdiagramm
- Datenflussdiagramm
- Prozessdiagramm
- Zustandsdiagramm
- Sequenzdiagramm
- ER-Diagramm
- XML
- OWL
- transients
- persistents
- Objektorientierten Modellierung
- Datenfluss
- Ereignis
- Aggregationsbeziehungen
- JSP
- V-Modell
- RUP
- Qualität
- mandatory requirements was muss das fertige System können
- Qualitative Anforderung: es hat die Eigenschaft oder hat sie nicht
- Quantitative Anforderung: ist eine messbare Eigenschaft z.B. die Antwortzeit beträgt max. 1 Sekunde
- Pflichtenheft

- Kunden
- Manager
- Entwickler
- Tester
- Klassifizierungen
- Normen
- Standards
- Systemzuverlässigkeit
- Systemwiederherstellungszeit
- Systemverfügbarkeit
- Risikomanagement
- Workaround
- Sicherheitslücke
- Hardware

7 Zusammenfassung

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.