Todo list

1 Einführung

Das Hochhalten von Softwarequalität ist eine sehr anspruchsvolle Aufgabe, welche eine gewisse Planung vorraussetzt. Gerade bei Projekten mit hohen Budgets und materiellen Einsätzen werden hohe Anforderungen an das Qualitätsmanagement gesetzt.

Einige bekannte Fehlschläge in der Softwareentwicklung hätten wahrscheinlich mit besseren Qualitätssicherungsmaßnahmen verhindert werden können:

- Pioneer 4 verfehlte den Mond
- unnötige Mahnungen durch die französische Finanzverwaltung
- das Herausgeben von faulen Krediten, was schlussendlich zum Bankencrash geführt hat
- Verlust einer Segelyacht im Pazifik

1.1 Verifikation und Validierung

Die Begriffe sollten klar getrennt werden, da sie nicht synonym verwendet werden können, und bei Softwareprozessen und Softwarequalität eine gewichtige Rolle spielen.

Die Verifikation stellt fest ob die Software mit der vorhandenen Spezifikation übereinstimmt, wohingegen die Validierung bestimmt ob die Software für den Kunden auch wirklich nützlich ist.

1.2 Technical Debt

Technical Debts sind ein wichtiges Thema in der Qualitätssicherung. Der Begriff wurde aus dem Finanzwesen übernommen (Debt = Schulden). In der Softwareentwicklungen ist damit gemeint, dass ungeschickte Lösungen irgendwann gefixt werden müssen.

Diese technische Schulden kann man bewusst eingehen, um Termine zu halten. Allerdings muss man immer im Hinterkopf behalten, dass man diese Schulden zu einem späteren Zeitpunkt auch wieder bezahlen muss.

Einer der größten Unterschiede zwischen klassischen und agilen Projektmanagementmethoden ist der Umgang mit diesen technischen Schulden. Bei agilen Methoden werden diese Schulden bewusst eingegangen, um ein schnelleres iteratives Vorgehen zu gewährleisten. Bei diesen Methodiken ist allerdings auch die Gefahr einer Überschuldung ungleich höher als bei den klassichen.

Allerdings werden in der Regel auch bei klassischen Methoden technische Schulden verursacht, nämlich in Form von Änderungen in den Anforderungen, auf welche in agilen Methoden besser reagiert werden kann.

2 Hausaufgabe Besprechnung

Die Aufgabe war ein Programm zu schreiben, welches in der Lage ist nach Eingabe der Koeffezienten eine quadratische Gleichung zu lösen.

Bei der Besprechung der Hausaufgabe wurde auf folgende Punkte aufmerksam gemacht:

- Klare Definition für Verhalten in Sonderfällen (z.B. komplexe Zahlen da in Wurzel negative Zahl)
- Gleitkommazahlen sollten auf Grund von Rundungsfehlern vermieden werden
- Statt der Standardformel ein eigenes Näherungsverfahren implementieren, um den Rundungsfehler zu minimieren

3 Wichtige Begriffe

- Softwareprozesse (Abfolge von Tätigkeiten, durch die ein Software-Produkt entsteht)
- Vorgehensmodell (Vereinfachte Beschreibung eines Softwareprozesses)
- Methode (Strukturierter Ansatz für die Software-Entwicklung)

3.1 Software-Engineering

Software-Engineering ist eine technische Disziplin welche eine Lösung für den Anwender bieten will unter Einsatz von Theorien, Methoden und Werkzeugen und Berücksichtigung von Organisation, Management und Entwicklung.

3.2 Informatik

Grundlage der Wissenschaft für Software-Ingeneure (Theorie, Konzepte, ...).

3.3 ESSENCE Kernel Overview

In diesem Model gibt es verschiedene Zustände für die Anforderungen und jeder dieser Zustände hat selbst wieder Kriterien welche einem dabei helfen in welchem Zustein ein Projekt ist.

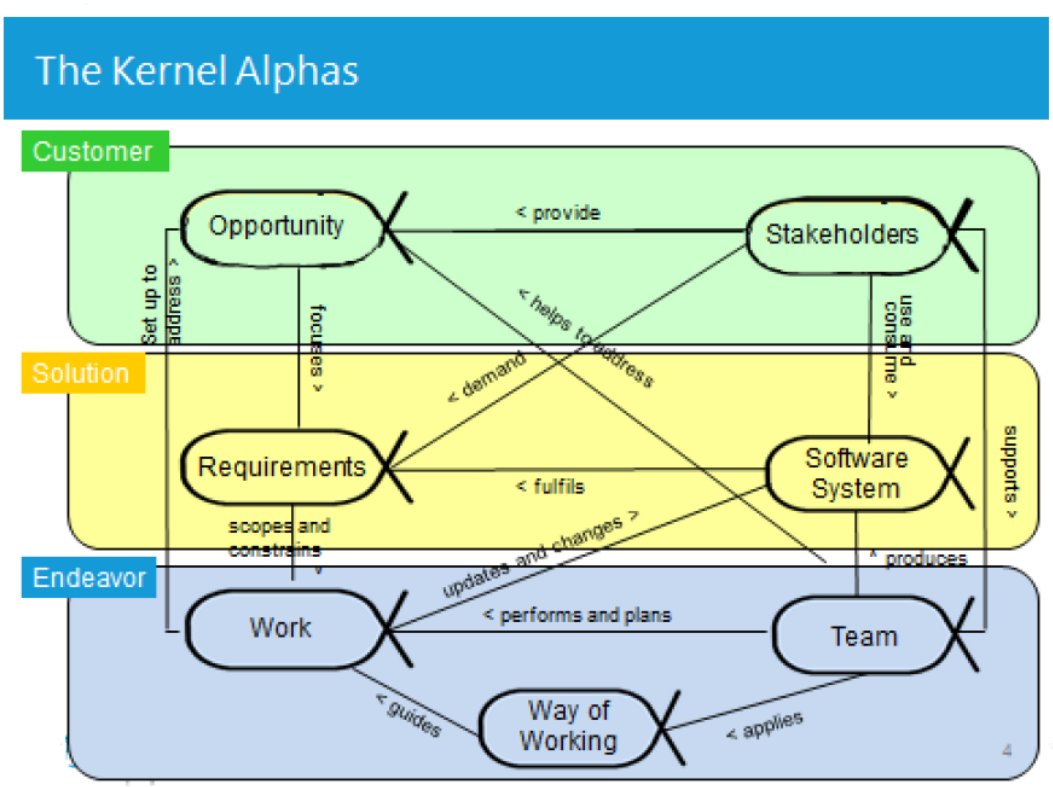

Abbildung 3.1: ESSENCE Kernel Overview

Zustände:

- Conceived
- Bounded
- Coherent
- Accepted
- Adressed
- Fulfilled

3.4 Qualität

Grad in dem die inhärenten Eigenschaften des Produkts Anforderungen erfüllen.

- **Explizite Anforderungen**
- **Implizite Anforderungen** - Anforderungen welche existieren aber den Stakeholdern nicht bewusst sind.
- **Nicht explizite Anforderungen** - Stakeholer wissen, dass sie diese Anforderungen gibt aber sie teilen diese nicht mit (sind quasis eh klar).
- **Objektiv** - Vollkommen klar, dass man etwas braucht.
- **Subjektiv** - Vermeintliche Anforderungen, welche nicht wirklich wichtig sind.
- alle betroffenen / interessierten Personen

3.4.1 Begriffsabgrenzung

Technisches computer-basiertes System System welches ausschließlich aus Soft- und Hardware-Komponenten besteht.

Soziotechnisches System System bestehend aus einem oder mehreren technischen Systemen, den Menschen die es bedienen, den notwendingen Arbeitsprozessen, organisatorischen Richtlinien, usw.

Systeme:

- haben systemspezifische Eigenschaften die nur dem System als Ganzem zugeordnet werden können
- sind häufig nicht deterministisch
- hängen von organisatorischen Zielen ab

3.5 Kritische Systeme

Relevante Systemeigenschaften für kritische Systeme sind:

- Reparierfähigkeit
- Wartbarkeit
- Überlebensfähigkeit
- Fehlertoleranz

Kritisches System Systeme, bei dessen Ausfall oder Fehlfunktion großen Schaden anrichten kann (wirtschaftliche Verluste, physische Schäden, Gefahr für Gesundheit und Leben von Menschen).

Sicherheitskritisches System Schäden an der Umwelt und/oder Gefahr für Gesundheit und Leben von Menschen (Bohrinsel im Golf von Mexiko).

Aufgabenkritisches System Aufgaben die ein System erledigen solle werden nicht durchgeführt (z.B. Bank).

Geschäftskritisches System Extrem hohe Kosten bzw. signifikante Gewinnausfälle können die Folge eines Systemausfalls sein.

4 Anforderungen

4.1 Sammlung von Anforderungen

4.1.1 Interviews

Technik

- Interviews mit Fragebogen
- Geschlossene Interviews vorgegebener Fragebogen abarbeiten
- Offene Interviews mit einer Diskussion zwischen Analyseteam und Beteiligten

Durchführungsempfehlung

- Mischung aller Methoden
- Vorbereitung aller Diskussionen als geschlossenes Interview
- gut und interessiert zuhören
- gezielt Fragen

Probleme

- Jargon des Anwendungsgebiets
- Implizites Wissen

Eignung

- Verständnis der Benutzeranforderungen
- Ergänzung zu anderen Informationsquellen mit zum Beispiel Dokumentationen oder Beobachtungen

Szenarien

- Grundbestandteile
  - Ausgangssituation
  - Ereignisablauf
  - Ausnahmen und ihre Behandlung
- Varianten
  - Ad-hoc
  - Formell

4.1.2 Ethnografische Methode (Völkerkunde)

Grundidee: Beobachten ohne einzugreifen

- Beobachten der alltäglichen Arbeit im normalen Umfeld
- Notieren von Auffälligen
- Diskussion mit Experten
- Ableitung der Anforderungen

Stärken

- Erkennen von impliziten Anforderungen
- Erkennen von nicht explizierten Anforderungen
- Erkennen von Abweichungen

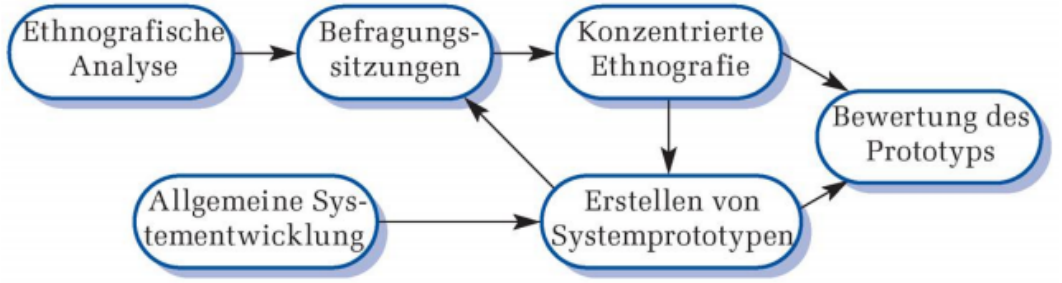

Abbildung 4.1: Verknüpfung von Ethografie und Prototypen nach Sommerville (Copyright Pearson Studium 2007)

4.2 Klassifizierung von Anforderungen

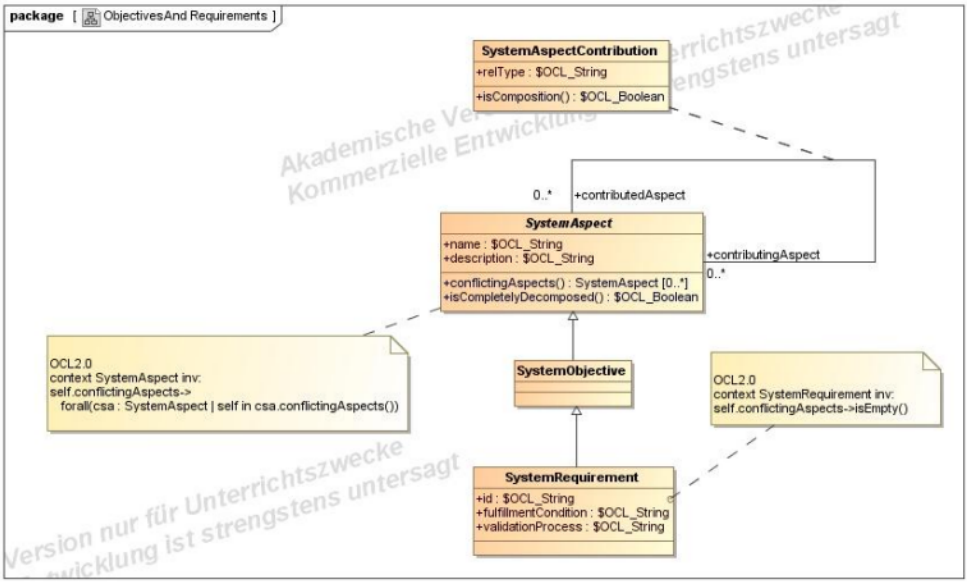

Abbildung 4.2: Metamodell

Aufgaben

- Erkennen von Duplikaten / Synonymen
- Beziehungen zwischen Anforderungen
- Gruppierung der Anforderungen

4.3 Validierung von Anforderungen

Wichtige Prüfungen sind:

- Gültigkeitsprüfung
- Konsistenzprüfung
- Vollständigkeitsprüfung (schwierig, aber mit etwas Bauchgefühl machbar)
- Realisierbarkeitsprüfung
- Verifizierbarkeitsprüfung

4.3.1 Techniken

Prototypen erstellen und Testfälle entwickeln. Falls das nicht möglich sind die Anforderugnen schlecht definiert.

4.3.2 Review

Teamzusammensetzung Es sollten Vertreter aller am Projekt beteiligten bzw. vom Projekt betroffenen Gruppen auf Anwenderseite, ausserdem Systemarchitekten und Vertreter der Softwareentwickler.

Durchführung Die Führung der Analyse liegt bei den jeweiligen Anwendervertretern. Diskutiert werden sollten alle Anforderungen. Dadurch können Fehler, Konflikte und Wiedersprüche aufgedekt werden. Das Ergebnis ist ein Review Bericht.

Prüfungen Die Konsistenz der Anforderungen sollten am Ende geprüft werden. Ausserdem sollte eine Vollständigkeitsprüfung durchgeführt werden.

4.3.3 Priorisierung von Anforderungen

Grundgedanke Phasenweise Implementierung der Software.

Prioritätsfestlegung

Teams festlegen, die nicht miteinander kommunizieren sollten. Ausserdem wird eine Bewertungsformel festgelegt.

Nutzwertanalyseteam Bewertet den Nutzen für die Anwender.

Kostenanalyseteam Schätzt die Kosten jeder Anforderung

Auswertung

Einflussgrößen

- Nutzwert jeder Anforderung
- Kosten jeder Anforderung
- Obergrenze des Aufwands für Stufe 1
- Abhängigkeiten

Ergebnis Liste der Anforderungen für nächste (bzw. erste) Ausbaustufe

4.4 Systemmodelle

Kontextmodelle Definiert die Systemgrenzen des Gesamtsystems und des technischen Systems. Das Gesamtsystem umfasst das Technische System und die Menschliche Komponente und definiert den Kontext. Das Technische System definiert den Scope.

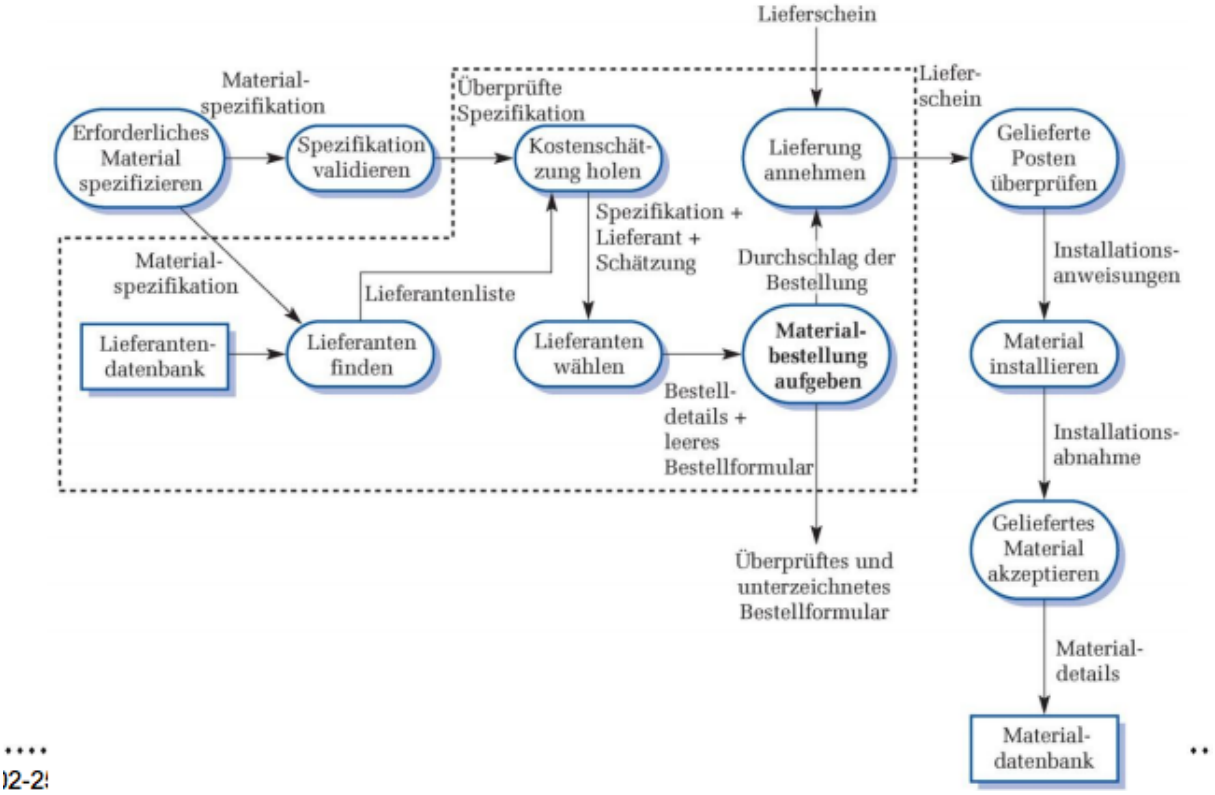

Abbildung 4.3: Kontextmodelle

Verhaltensmodelle Definiert die Abläufe in einem System. Sie können Datenfluss (Datenfocus) - oder Ereignis (Ereignisfocus) - Orientiert geschehen. Aussderm gibt es Mischformen davon. Dafür können Datenflussdiagramme (Abbildung 4.4), Zustandstabelld (Abbildung 4.5) und ähnliches verwendet werden.

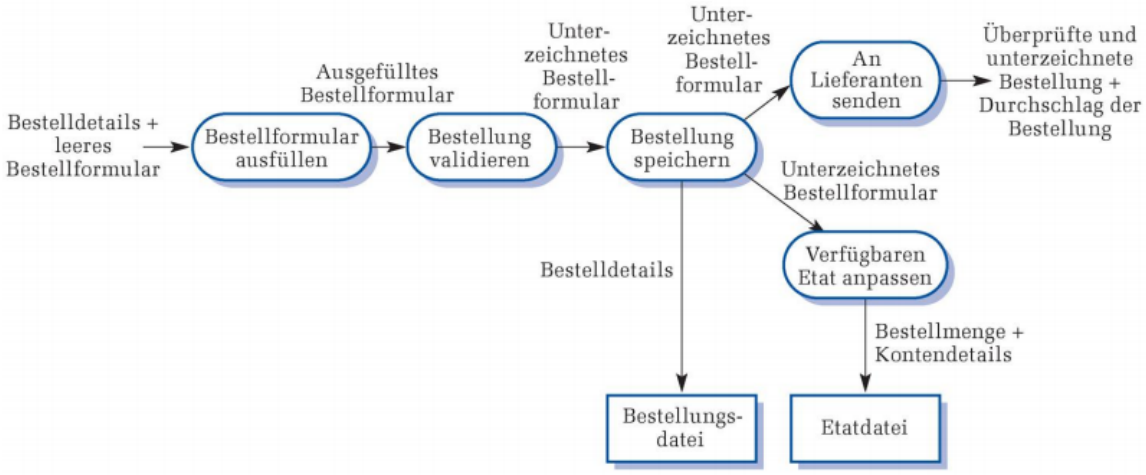

Abbildung 4.4: Datenflussdiagramm

Datenmodelle Definiert die logische und persistente (lokal, übergreifend mttels Datenbank und Datenaustausch) Datenstruktur. Dazu kann ER-, UML-Diagramme, OWL oder andere Ontologie-Darstellungen und andere.

Zustand	Beschreibung
Warten	Das Gerät wartet auf eine Eingabe. Die Anzeige zeigt die aktuelle Zeit an.
Halbe Leistung	Die Geräteleistung wird auf 300 Watt gesetzt. Die Anzeige zeigt „Halbe Leistung" an.
Volle Leistung	Die Geräteleistung wird auf 600 Watt gesetzt. Die Anzeige zeigt „Volle Leistung" an.
Zeiteinstellung	Die Garzeit wird auf die Eingabe des Benutzers gesetzt. Die Anzeige zeigt die gewählte Garzeit an und wird bei der Einstellung der Zeit aktualisiert.
Deaktiviert	Der Betrieb des Geräts ist aus Sicherheitsgründen deaktiviert. Das Licht im Gerät ist eingeschaltet. Die Anzeige zeigt „Nicht bereit" an.
Aktiviert	Der Betrieb des Geräts ist aktiviert. Das Licht im Gerät ist ausgeschaltet. Die Anzeige zeigt „Bereit" an.
Betrieb	Das Gerät ist in Betrieb. Das Licht im Gerät ist eingeschaltet. Die Anzeige zeigt den Countdown der Garzeit an. Nach dem Ende der Garzeit ertönt fünf Sekunden lang ein Signal. Das Licht im Gerät ist eingeschaltet. Die Anzeige zeigt „Kochvorgang beendet" an, während das Signal ertönt.
Stimulus	**Beschreibung**
Halbe Leistung	Der Benutzer hat die Taste „Halbe Leistung" betätigt.
Volle Leistung	Der Benutzer hat die Taste „Volle Leistung" betätigt.
Timer	Der Benutzer hat eine der Tasten für die Zeitschaltuhr betätigt.
Zahl	Der Benutzer hat eine Zifferntaste betätigt.
Tür offen	Die Gerätetür ist nicht geschlossen.
Tür geschlossen	Die Gerätetür ist geschlossen.
Start	Der Benutzer hat die Starttaste betätigt.
Abbruch	Der Benutzer hat die Abbruchtaste betätigt.

Abbildung 4.5: Zustandtabelle

Objektorientierte Modellierung Vereinigt die Funktionalität von Daten- und Verhaltensmodelle und können Daten, Datenflüsse, Datenstrukturen und Erreignise erfassen. Als Notation wird meist UML verwendet.

Sind die Klassen in dem Diagramm Abbildung 4.6 wirklich **Klassen oder Rollen**? Nein das sind eindeutig Rollen.

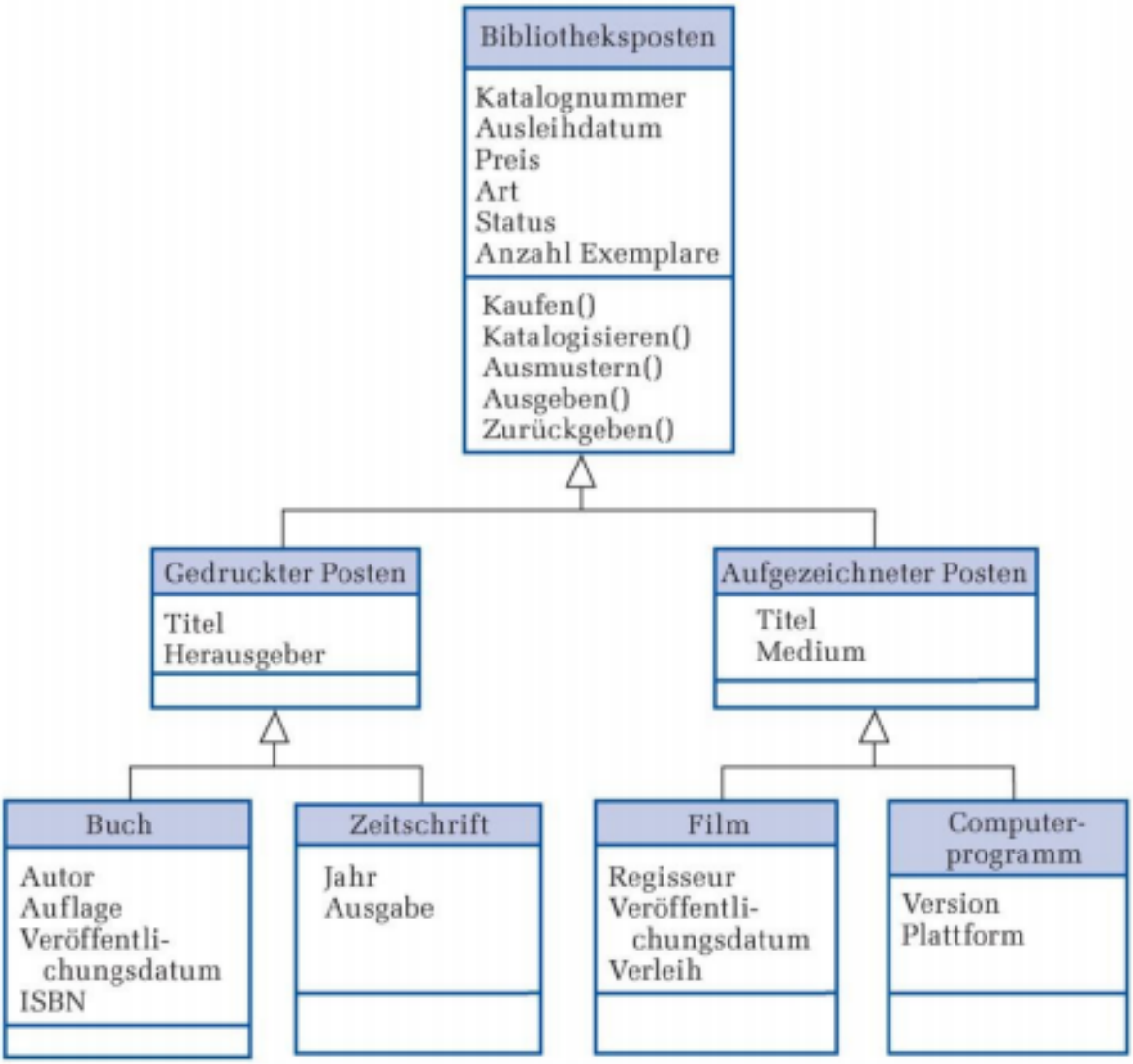

Abbildung 4.6: Objektorientierte Modellierung

Strukturierte Methoden Detailliert definierte Vorgehensweise bei der SW-Entwicklung. Normalerweise basierend auf einem Satz von Diagrammtypen. Definiert zusätzliche Regeln und Richtlinien. Beispiele dafür sind JSP, V-Modell oder RUP. Nachteile davon sind Mangelnde Unterstützung nicht-funktionaler Anforderungen und Anwendbarkeit für konkretes Problem oft schwer zu entscheiden.

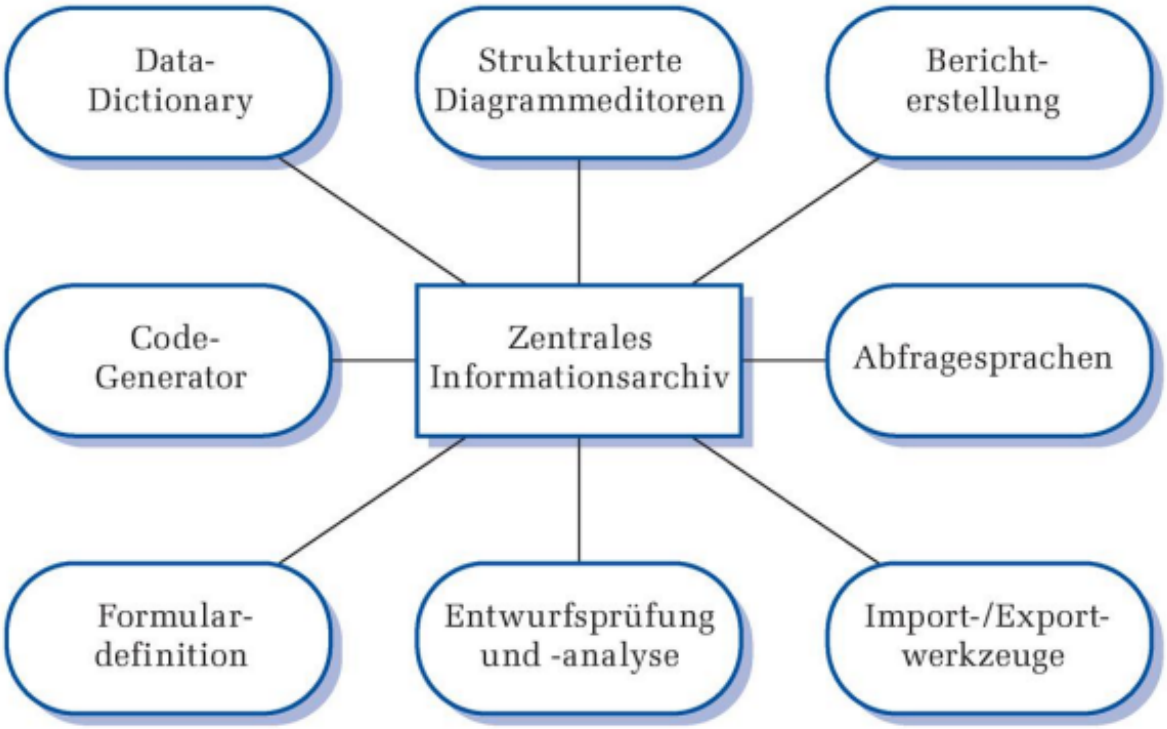

Abbildung 4.7: Strukturierte Methoden

4.5 Anforderungsmanagementsystem

Die Anforderungen an ein System ändern sich mit der Zeit, diese können ursprünglich unvollständig sein. Oder es verbessert sich das Verständnis des Problems / es ensteht eine bessere Sicht der Dinge. Aber auch das Umfeld kann sich verändern, z.B. wirtschaftlich, technisch, juristisch ...

Der Prozess umfasst:

- das
  - Sammeln
  - Verstehens
  - Klassifizierens
  - Validierens
  - Priorisierens

- von
  - Anforderungen
  - Änderungen der Anforderungen
- und ihren Auswirkungen auf
  - andere Anforderungen
  - Entwurfsentscheidungen
  - das System

Dauerhafte Anforderungen Sie sind relativ stabil und sind mit dem Kern der Anwendung verwoben. Oft aus Standardisierten Modellen entnohmen.

Veränderliche Anforderungen Anforderungen mit hoher Änderungswahrscheinlichkeit. Können wirtschaftliche Randbedingungen, technische Randbedingungen und gesetzliche Randbedingungen.

4.6 Pflichtenheft

Zusammenstellung der vollständigen und detailierten Benutzeranforderungen und Systemanforderungen (großes Problem ist die Vollständigkeit). Darf nach Fertigstellung nicht mehr geändert werder und muss so implementiert werden wie es festgehalten wurde (auch wenn sich die darin enthaltenen Entscheidungen als falsch herrausstellen). Dies muss aus juristischen Gründen so sein, da eine nachträgliche Änderung als eine Benachteiligung der Mitbewerber interpretiert werden kann.

Synonyme sind die Begriffe Produktdefinition, Produktspezifikiation, Systemdefinition und Systemspezifikation.

Adressaten des Pflichtenhefts sind unter anderem Systemkunde, Manager, Systementwickler, Systemtester, Systemwarter und Juristen.

Informationen im Pflichtenheft sind qualitative und quantitative Anforderungen (mandatory requirement) sowie zusätzliche Wünsche (optional requirements) dessen Erfüllung als sehr positiv betrachtet werden und informative Bestandteile. Diese Bestandteile dienen dem Verständniss und sind streng genommen unverbindliche Informationen.

In vielen Ländern gelten Abbildungen, Tabelle und Anhänge als informative Bestandteile, wenn sie nicht anderst gekennzeichnet sind (engl. "normativ").

Aufbau des Pflichtenhefts

- Einleitung
  - Ziele des Pflichtenhefts
  - Anwendungsbereich des Produkts
  - Definitionen, Akronyme und Abkürzungen
  - Referenzen
  - Überlick über das Pflichtenheft
- Allgemeine Beschreibung
  - Produktperspektive
  - Produktfunktion
  - Produktcharakteristika aus Benutzersicht
  - Beschränkungen
  - Voraussetzungen und Abhängigkeiten
- Spezifische Anforderungen (funktionale u. nicht funktionale Anforderungen)
- Anhänge
- Index

4.7 Anforderungensarten

Funktionale Anforderungen unterstützen Definition von Funktionen zur Fehlerprüfung, zur Wiederherstellung im Fehlerfall und Aspekte zum Schutz gegen Systemausfälle.

Als nichtfunktionale Anforderungen werden u.a. die Systemzuverlässigkeit und Systemverfügbarkeit festgelegt.

Negativanforderungen Beschreibung von Verhalten oder Eigenschaften, die das System auf keinen Fall zeigen darf.

4.8 Risikomanagement

Risikomanagement besteht aus Gefahrenbestimmung, Risikoanalyse und Gefahrenklassifizierung, Gefahrenvereinzelung und Festlegung zur Risikominimierung.

Murphys Law Alles was schief gehen kann, wird irgendwann einmal schief gehen.

Risikoerkennung Ziel ist es zu Erkennen von Rsikiken und Gefahren. Probleme die dabei entstehenkönnen sind die Wechselwirkung zwischen Systemkomponenten bzw. die Wechselwirkungen mit der Umwelt.

Risikoanalyse und Risikoklassifizierung Analyse von Unfallwahrscheinlichkeit, Schadenswahrscheinlichkeit und Schadeshöhe. Risiken sind klassifizierbar in:

- nicht tolerierbar
  - großer Schaden und / oder hohe Schadenswahrscheinlichkeit
  - Maßnahmen: unter allen Umständen Risiko ausschließen
- as low as reasonably possible (ALARP)
  - höchstens mittleres Schadenspotential und mittlere Schadenswahrscheinlichkeit
  - Maßnahmen: zumindestens Auschluss des Schadens unter Beachtung von wirtschaftlichen, vertraglichen oder technischen Randbedingungen
- vernachlässigbar
  - geringes Schadenspotential und / oder geringe Schadenswahrscheinlichkeit
  - Maßnahmen: Reduzierung der Schadenswahrscheinlichkeit ohne negativen Einfluss auf wirtschaftliche, technisches, oder vertragliche Randbedingungen oder andere Systemanforderungen

Risikozerlegung Ziel es die Ursachen aufzudecken. Dazu können Reviews, Checklisten, Petri-Netze, formale Login und Fehlerbäume verwendet werden.

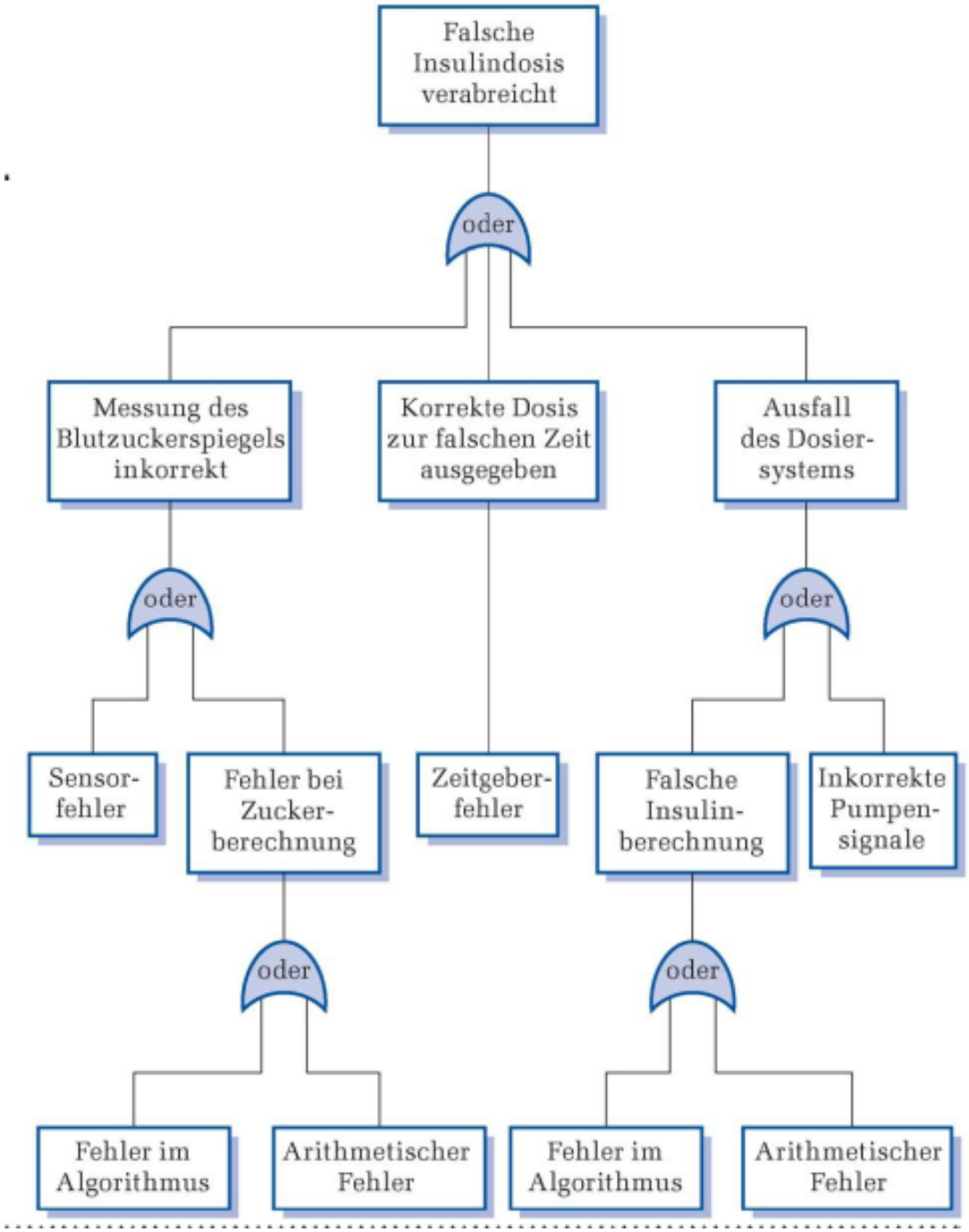

Abbildung 4.8: Fehlerbaum

Risiko-Minimierung Ziel ist das Vermeiden des Auftretens von Gefahren. In der Praxis bedeutet dies eine Kombination aller Strategien.

4.9 Betriebssicherheit

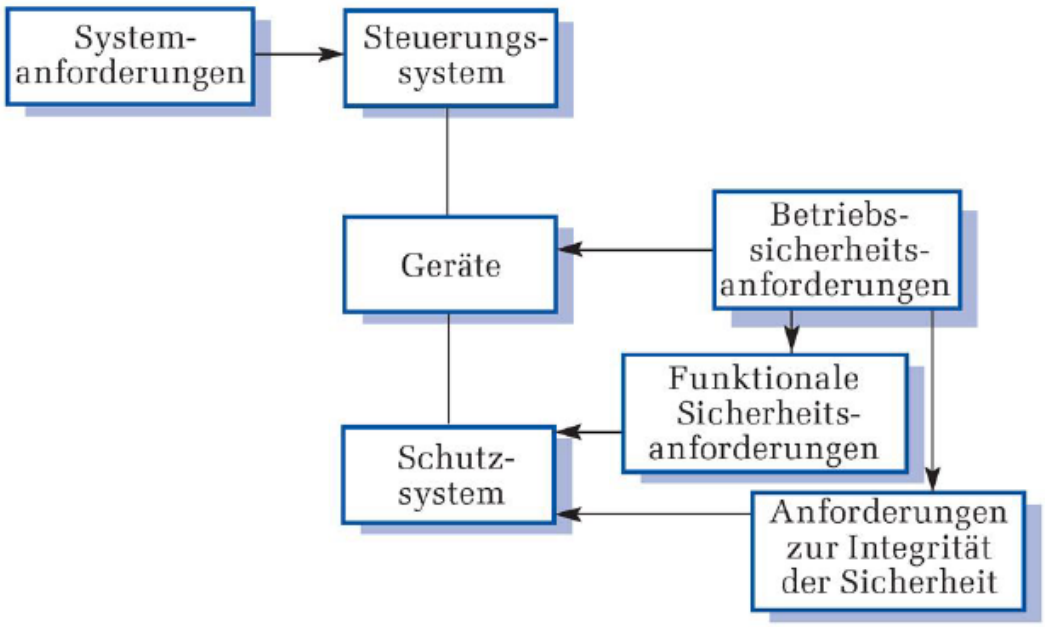

Abbildung 4.9: Betriebssicherheit nach IEC 61508

Klassifizierung

- Klasse 1 - Fehler die andere Anwedungen betreffen können
- Klasse 2 - Fehler die andere Benutzer der selben Anwedung betreffen können
- Klasse 3 - Wesentliche Funktion steht ohne Work-Around nicht zur Verfügung
- Klasse 4 - Wesentliche Funktion steht nur über Work-Around zur Verfügung bzw. Sekundärfunktion steht nicht zur Verfügung
- Klasse 5 - Verbesserungswunsch

4.10 Systemsicherheit

Hierbei geht es um direkte und indirekte Berodhungen.

Direkte Bedrohung wie z.B. unbefugtes Eindringen und DOS.

Indirekte Bedrohung wie Aufwand um Sicherheitsmaßnahmen zu installieren, konfigurieren und aktuell zu halten. Auch inkludiert sind Fehler und Nachlässigkeiten der Systemverwalter und monopolartige Stellungen einiger Hard- und Softwareanbieter da eine weite Verbreitung von Sicherheitslücken sehr wahrscheinlich ist.

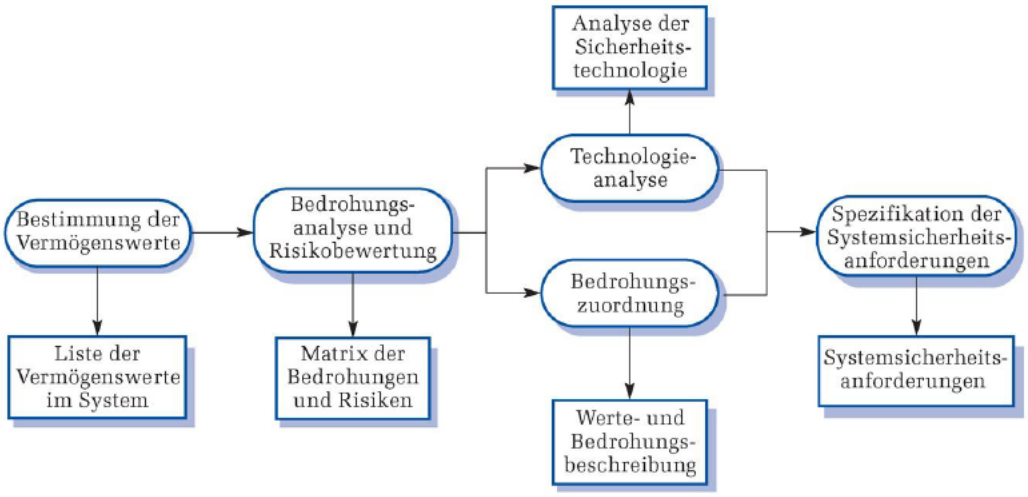

Abbildung 4.10: Sicherheit

4.11 Zuverlässigkeit

besteht aus Hardware-, Software- und Bedienerzuverlässigkeit.

Extreme Zuverlässigkeitsanforderungen lassen sich nicht testen!

Metrik	Erläuterung
POFOD Probability of failure on demand (Wahrscheinlichkeit eines Fehlers bei Anfrage)	Die Wahrscheinlichkeit, dass das System bei einer Dienstanforderung fehlschlägt. Eine POFOD von 0,001 bedeutet, dass eine von tausend Anforderungen zu einem Fehler führt.
ROCOF Rate of failure occurrence (Ausfallrate)	Die Frequenz, mit der ein unerwartetes Verhalten auftritt. Eine ROCOF von 2/100 bedeutet, dass pro 100 operativen Zeiteinheiten zwei Fehler zu erwarten sind. Diese Metrik wird manchmal auch Fehlerintensität (failure intensity) genannt.
MTTF Mean time to failure (Mittlere Zeit bis zu einem Ausfall)	Die durchschnittliche Zeit zwischen beobachteten Systemausfällen. Eine MTTF von 500 bedeutet, dass alle 500 Zeiteinheiten ein Ausfall erwartet werden kann.
AVAIL Availability (Verfügbarkeit)	Die Wahrscheinlichkeit dafür, dass das System zu einer gegebenen Zeit verfügbar ist. Eine Verfügbarkeit von 0,998 bedeutet, dass das System alle 1000 Zeiteinheiten wahrscheinlich in 998 Fällen verfügbar ist.

Abbildung 4.11: Zuverlässigkeit

Fehlerklasse	Beispiel	Metriken für Zuverlässigkeit
dauerhaft, nicht beschädigend	Das System funktioniert mit keiner der eingegebenen Karten. Die Software muss neu gestartet werden, um den Fehler zu beheben.	ROCOF 1 Vorfall/1.000 Tage
dauerhaft, nicht beschädigend	Die Magnetstreifendaten von einer unbeschädigten Karte können nicht gelesen werden.	ROCOF 1 pro 1.000 Transaktionen
dauerhaft, beschädigend	Ein Transaktionsmuster im Netzwerk verursacht eine Datenbankbeschädigung	Nicht messbar! Sollte während der Lebensdauer des Systems nicht vorkommen.

Abbildung 4.12: Zuverlässigkeit nach Sommervill

4.12 Formale Spezifikationsmethoden

An dieses Thema sind seit 1970 sehr hohe Erwarungen gerichtet aber der große Durchbruch blieb bisher aus. Gründe dafür sind die Entwicklung anderer Software-Engineering-Methoden, Marktveränderungen, beschränkte Einsetzbarkeit und mangelhafte Skalierbarkeit.

Erfolgreich verwendet wirds bei Raumfahrt-, und medizinischen Systemen sowie bei Überwachung des Luftverkehrs.

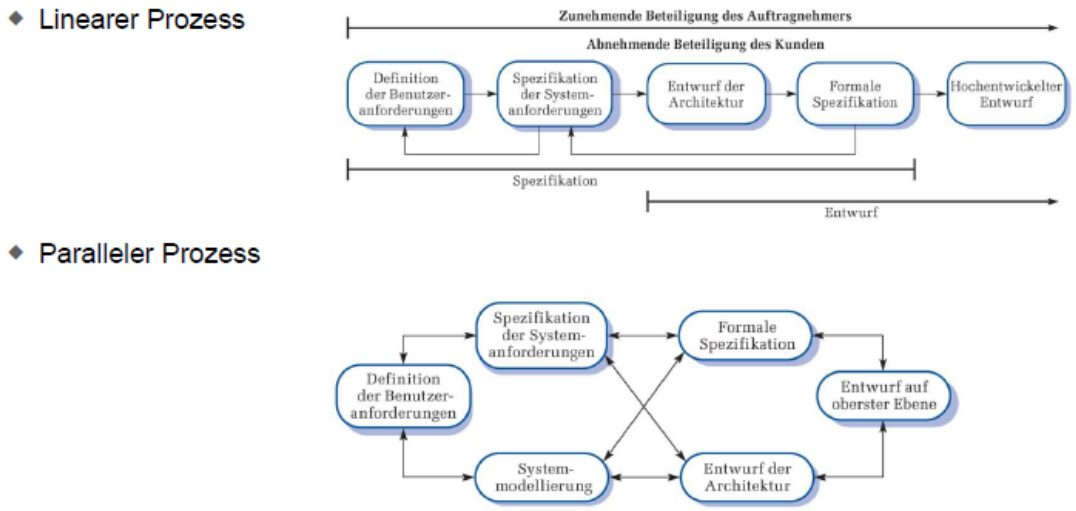

Abbildung 4.13: Formale Spezifikationsmethoden im Softwareprozess

Fazit Formale Spezifikationstechniken sind gute Ergänzungen, eindeutig und präzise aber schwer verständlich für den Laien. Erzwingen die frühzeitige Analyse der Systemanforderungen wenn die Fehlerbehebung noch billig ist.

4.13 Beispiel: Anforderungen an Quadratische Gleichungen lösen

- Ergebnis x element aus Reelen Zahlen : $ax^2 + bx + c = 0$
- Kein Absturz bei komplexer Lösung
- Vernünftige Reaktion auf Eingabefehler
  - unvollständig
  - $4ac > b^2$
- Maximaler Rundungsfehler: Relativ 0,1% + Beweis
- Immer Definiertes Ergebnis (Ausnahmebehandlung)
- Keine Endlosschleife
- Zeitbedarf max 1mSec
  - Verwendeter CPU / Prozessor / Rechner

  - Maximal 2xSQRT einer bestimmten implmentierung
- Signatur
  - Nmae
  - parameter
  - Ergebnis (Return)

5 Entwicklung

Strategien

- Fehlervermeidung
- Fehlerentdeckung
- Fehlertoleranz

Laut der Kostenentwicklung nach Sommerville ist eine Software mit vielen Fehlern günstiger als eine Software mit weniger Fehlern. Jedoch steigen die Kosten nicht linear sondern exponentiell.

Techniken

- Verlässliche Softwareprozesse
- Qualitätsmanagement
- Formale Spezifikation
- Statische Verifikation (Probleme durch Reviews beheben)
- Starke Typisierung
- Sichere Programmierung
- Ausnahmebehandlung
- Geschützte Information

5.1 Fehlervermeidung

Verläsliche Softwareprozesse

- Inspektion der Anforderungen
- Anforderungsmanagement
- Überprüfung der Modelle: statisch / dynamisch
- Inspektion des Entwurfs / Codes
- Statische Analyse des Codes (Review)
- Planung und Management von Tests
- Konfigurationsmanagement

5.2 Sichere Programmierung

- sichere verwendung von GOTO (mach da weiter ohne überprüfung)
- sichere verwendung von POINTER (such da mal nach deinen daten => goto in der Datenwelt)
- Rundungsfehler von Gleitkommazahlen
- Dynamische Speicherallokierung
- Parallelität
- Rekursion
- Interrupts
- Vererbung
- Aliasing
- Fehlende Überprüfung von Arraygrenzen
- Konfiguationsdateien

Ausnahmebehandlung Wird in manchen Sprachen mangelhaft Unterstützt (Beispiel: C). Kann auch dazu verwendet werden um die Lesbarkeit der Anwendung zu erhöhen und damit können Fehler vermieden werden.

5.3 Fehlertoleranz

Fehlererkennung Typen sind Vorbeugend (ich versuche vorher Fehler zu erkennen) und Rückblickend (ich behebe den Fehler wen einer auftritt => z.B. die Transaktion zurücksetzen).

Wichtige Hilfsmittel Validierung: Invarianten, Vorbedinungen, Nachbedinungen => dürfen nicht abschaltbar sein.

Wiederherstellung nach Fehlern Pesimistisch (nur zurücksetzen geht nicht ...) oder Optimistisch (erneut versuchen den gewünschten zustan zu erreichen).

Vorbeugende Fehlererkennung hilft diesen Aufwand zu vermeiden.

5.4 Fehlertolerante Architekturen

Widerherstellungsblöcke Verschiedene Algorithmen implementieren. Im Betrieb den ersten ausführen und testen, bei Fehler nimm nächsten Algorithmen. Probleme: Rundungsfehler

5.5 Wartungsaufwand

Gründe für hohen Wartungsaufwand sind Informatinosmanagement (Verlust des Entwicklungsteam und seiner Erfahrung), Vertragliche Zuständigkeit (Entwicklung von einer Firma, wartung von einer anderen), Fähigkeiten der Metarbeiter (Mangelnde Erfahrung mit dem Anwendungsgebiet und Technologien) und Alter bzw. Struktur des Programs (Dokumentation nicht mehr aktuell oder nicht verhanden, fehlendesKonfigurationsmanagement).

Zusätzlichen Entwicklungsaufwand um Software besser wartbar zu machen zahlt sich im allgemeinen nicht aus.

5.6 Weiterentwicklungsprozesse

Es gibt für jede Anforderung einen passenden Prozess (laufende weiterentwicklung, änderungungen, Bug-Behebungen, ...)

Software-Reengineering

Dabei handelt es sich um eine Neuentwicklung von Software, meistens eine Umstellung auf neuere Technologien (z.B. von COBOL auf Java).

Dafür ist auch das Beherrschen der alten Technologien notwendig, da man andernfalls das alte System nicht analysieren kann. Die Beweggründe hinter der Umstellung ist ein verringertes Risiko und geringere Kosten.

Normalerweiße teilt sich das Reengineering in Reverse Engineering und in eine Verbesserung der Programmstruktur. Im Anschluss daran müssen natürlich auch die Daten auf eventuell neue Formate transformiert werden.

Legacy Systeme

Legacy Systeme werden nach zwei Kriterien beurteilt: Qualität und Geschäftswert.

Wenn beide Kriterien schlecht sind wird die Software nicht mehr weiterentwickelt. Bei hoher Qualität kann man das einfach weiter laufen lassen, bei hohem Geschäftswert aber niedriger Qualität wird das komplett neu geschrieben, und wenn beide Kriterien hoch einzustufen sind rentiert sich ein Software Reengineering Ansatz.

6 Testing

Betrifft die Kernel Alphas Softwaresystem, Requirements und Work. Tests sollen uns helfen Probleme zu entdecken und beheben, die Kunden überzeugen und die Softwarequalität demonstrieren. Dabei ist allerdings zu beachten, dass nur die Anwesenheit von Fehlern getestet werden kann, allerdings nicht ihre Abwesenheit.

- Fehlverhalten (Failures) beschreiben ein inkorrektes Verhalten eines Systems zur Laufzeit
- Fehler kommen wirklich im Quellcode vor
- Irrtürmer (Errors) sind inkorrekte Wahrnehmungen von Menschen

Dabei muss es nicht zwingendermaßen sein dass nicht jeder Irrtum zu einem Fehler, und nicht jeder Fehler zu einem Fehlverhalten führt.

Für die Reduzierung von Fehlern gibt es verschiedene Maßnahmen, z.B. das 4-Augen-Prinzip zur Vermeidung, Tests zum Entdecken, und standardisierte Prozesse zum Beseitigung von Fehlern.

Weiters wird zwischen Blackbox- und Whitebox-Tests unterschieden, bei Blackbox Tests wird rein gegen die Spezifikation getestet, ohne den Code zu testen, wohingegen Whitebox-Tests den Code berücksichtigen.

Ein weiterer Typ ist ein Regressionstest, der neben der erwarteten Funktionalität sicherstellen soll, dass das System frei von alten Fehlern ist. Das Einführen alter Fehler kann sehr schnell passieren, wenn Code mit anderen Absichten refactored wird.

Ein Testfall wird in einen abstrakten und konkreten Teil eingeteilt. Der abstrakte Teil beschreibt den Test, der konkrete ist eine Ausführung des Tests. Eine Testsuite vereint mehrere Tests zu einer Ausführungseinheit.

6.1 Testprozesse

Ein Testprozess muss geplant und vorbereitet werden. Dazu gehören die Definitionen der Testziele und verwendeten Messungen, sowie die Eingabewerte und Testsuites.

Die Messungen müssen dann auch noch analysiert werden, vor allem geht es da um das Auffinden von Fehlern. Diese Fehler müssen von Softwareentwicklern behoben werden, insofern bei der Planung der Messungen keine Fehler gemacht werden. Bei TDD testen sich die Software und Tests gegenseitig, da beim Auftreten eines Fehlers sowohl das System under Test als auch der Test selbst fehlerhaft sein kann.

6.1.1 Teststrategien

Ausprobieren

Planloses Ausprobieren ist die häufigste Form des Testens, allerdings ist diese Form des Testens in vielen Fällen unzureichen. Es liefert keine reproduzierbare Ergebnisse, und erlaubt auch keine Aussage über die Qualität.

Das ist vor allem unzureichend wenn man eine Gewährleistung auf die Software geben will. Die rechtlichen Rahmenbedingungen sind momentan noch im Sinne der Softwareanbieter, allerdings könnte sich das in Zukunft ändern, gerade selbstfahrende Autos könnten da Ausschläge in diese Richtung geben.

Vollständiges Testen

Das vollständige Testen ist meist theorethisch möglich, aber nur in den seltensten Fällen wirtschaftlich. Unser Ziel muss es deshalb sein eine möglichst hohe Testabdeckung zu erreichen, und dabei auch noch wirtschaftlich zu bleiben.

Checklisten

Ein mögliches Vorgehen sind Checklisten, die die wesentlichsten Funktionen des Systems definieren, und nur diese getestet werden. Natürlich kann man Checklisten auch für andere Kritieren definieren.

Wenn man sichere Aussagen über die Qualität machen will, wird man sehr umfangreiche Checklisten bekommen. Andererseits ist der Vorteil das man eine verbesserte Reproduzierbarkeit erhält, allerdings wird sich niemand die Zeit nehmen vollständige Checklisten zu erstellen.

Partitionen

Eine Partition ist eine nicht überlappende Menge von Untermengen einer Menge, die bei der Beseitigung der Nachteile von Checklisten behilflich sein können. Diese Untermengen werden als Äquivalenzklasse bezeichnet, und es reicht wenn nur ein Element einer Äquivalenzklasse getestet wird. Bei sinnvoller Partitionierung kann der Testaufwand bei gleichzeitig erhöhter Qualität verringert werden.

Durch die gleiche Gewichtung der Äquivalenzklassen führt allerdings zu unwirtschaftlicher Verteilung der Testaufwände. Administrationwerkzeuge werden z.B. weit weniger aufgeruft, und haben dadurch schon eine niedriger Fehlereintrittwahrscheinlichkeit.

Benutzungsprofile

Da das Testen mit Partitionen immer noch sehr aufwändig sein kann, ist die Verwendung von Benutzungsprofilen möglich. Dabei wird die Benutzung der Partitionen mit einberechnet, und man kann damit mehr Testaufwand für mehr benutzte Partitionen aufwenden.

Bei existierenden Produkten kann für die Erstellung der Benutzungsprofile anhand des aktuellen Produkts bzw. von Vorgängerversionen gemacht werden. Bei neuen Produkten muss man hier wohl auf ein Konkurrenzprodukt zurück greifen. Bei einem komplett neuartigen Produkt bleibt einem nur noch das Schätzen übrig, wobei man hier auf Fachleute der jeweiligen Domäne heranziehen sollte.

Hilfreich ist hier die Definition von Kundenprofilen, die danach anhand der Benutzung gewichtet werden.

Überdeckung

Die Partitionen werden auf Basis der Eingabedaten gemacht. Die Eingaben in einer Partitionen müssen gleich behandelt werden. Z.B. realle/imaginäre Lösung der Eingabe für unser Lösungsprogram.

Die größte Fehlerwahrscheinlichkeit besteht an der Partitionsgrenzen. Wenn ein Array z.B. 100 Elemente umfasst, so sollte auch das Einfügen am Index 100 getestet werden, da hier bereits kein Fehler auftreten sollte.

Partitionen sollten Überlappungsfrei und vollständig sein, um eine möglichst hohe Testqualität zu gewährleisten. Für jede Unterdomäne müssen Eingaben ausgewählt werden, für die dann Tests durchgeführt werden. Im Idealfall werden so alle Möglichkeiten im Code abgearbeitet.

Probleme die auftreten können sind unter anderem widersprüchliche Verarbeitungsregeln (überlappende Unterdomänen) oder Probleme mit Rundungsfehlerdurch Grenzdefinitionen. Auch überflüssige Grenzen werden in einen unnötig hohen Testaufwand resultieren.

Schwache Nx1-Strategie Es werden N unabhängige Testpunkte auf den Granzen ausgewählt + einen zusäzlichen Testpunkt der nicht auf der Grenze liegt (in minimalem abstand von der Grenze). Dadurch können Abschlussprobleme, Grenzverschiebungen und fehlende Grenzen erkannt werden. Zsätzlich können überflüssige Grenzen erkannt, aber nicht notwendigerweise alle.

Endliche Automaten und Markov-Modelle

Die meisten Anwendungsprogramme lassen sich als endliche Automaten mit Ausgabe darstellen. Allerdings ist die Anzahl meist zu hoch zum testen. Es sind vereinfachungen notwendig.

Diese Teststrategie eignet sich für Web-Applikationen. Man beschränkt sich auf relevante Seiten und lösst dadurch Eintritts und Austrittsprobleme.

Für die Konstruktion werden die Zustände, Übergänge definiert, dadurch können die IO-Beziehungen abgeleitet werden und das Modell validiert werden. Ziele für die Zutände sind z.b. die Erreichbarkeit, Vollständigkeit und die Notwendigkeit zu testen. Ziele für die Übergänge sind z.b. die Durchführbarkeit, Vollständigkeit und die Notwendigkeit zu testen.

Markov-Modelle / -Ketten sind endliche Automaten mit Wahrscheinlichkeiten an jedem Zustandsübergang. Die Addition der Wahrscheinlichkeiten kann unter 100% fallen wenn nicht alle Möglichkeiten bekannt sind.

Die Wahrscheinlichkeit eines konkreten Übergangs ist das Produkt aller Wahrscheinlichkeiten auf dem Pfad vom Start bis zum Ausgangszustand, multipliziert mit der Wahrscheinlichkeit dieses konkreten Übergangs => diese Tatsache knn bei Schleifen Probleme machen.

UMM - Unified Markov Model ist ein kohärenter Satz von hierarchisch geschachtelte Markov-Modellen. Kann in der Praxis sehr gut angewendet werden. Es können alle Arten von statistischen Aussagen getätigt werden.

Kontrollfluss

Das Testziel ist die ausführung aller Pfade im Programm zu testen. Dazu wird ein CFG (Control Flow Graph) erzeugt. Die Knoten sind ausführbare Anweisungen, Startknoten sind erste, Endknoten sind die letzten Anweisungen und die Kanten sind mögliche Verarbeitungssequenzen.

Bei Schleifen können die Anzahl der Ausführung getestet werden (0, 1, 2 Durchläufe und evtl. die Grenzfälle - bei einer Vorhersehbare durchläufe). Es wird empfohlen die Pfade von hinten her aufzubauen. Dies unterstützt die Identifikation von unbenötigtem Code.

Datenfluss

Bei einem **DDG - Data Dependency Graph** sind die Knoten die Zustände derVariblen und die Kanten sind Operationen des Programmes. Er beschreibt den Zustand von der Variable X in jedem Zeitpunkt.

Die Richtung der Pfeile bestimmt die Art zu Zugriffes (Hin schreiben, Weg lesen). Die Zugriffsfolgen können nun bestimmte Probleme vermuten lassen. Beispiel Schreiben und sofort nochmal Schreiben -> könnte ein Überschreiben bedeuten oder ein mehrfaches Schreiben könnte eine ineffizienz bedeuten. Ein weiteres Beispiel ist das lesen von nicht initialisierten Variablen (Lesen ohne vorher zu schreiben.

Schleifen sollten vorsichtshalber zweimal durchlaufen werden um sicherzugehen.

6.2 Testverfahren

Unit-Test sind White-Box Tests durch den Entwickler. Informelles Testen um Fehler zu lokalisieren.

Komponenten-Test können sowohl als White- als auch als Black-Box Tests durchgeführt werden.

Integrationstest Black Box in Bezug auf die zu integrierenden Komponenten, White Box in Bezug auf die Integrationsschicht, durch Entwickler.

Systemtest Black Box durch professionelle Tester.

Regressionstest Black Box durch Entwickler

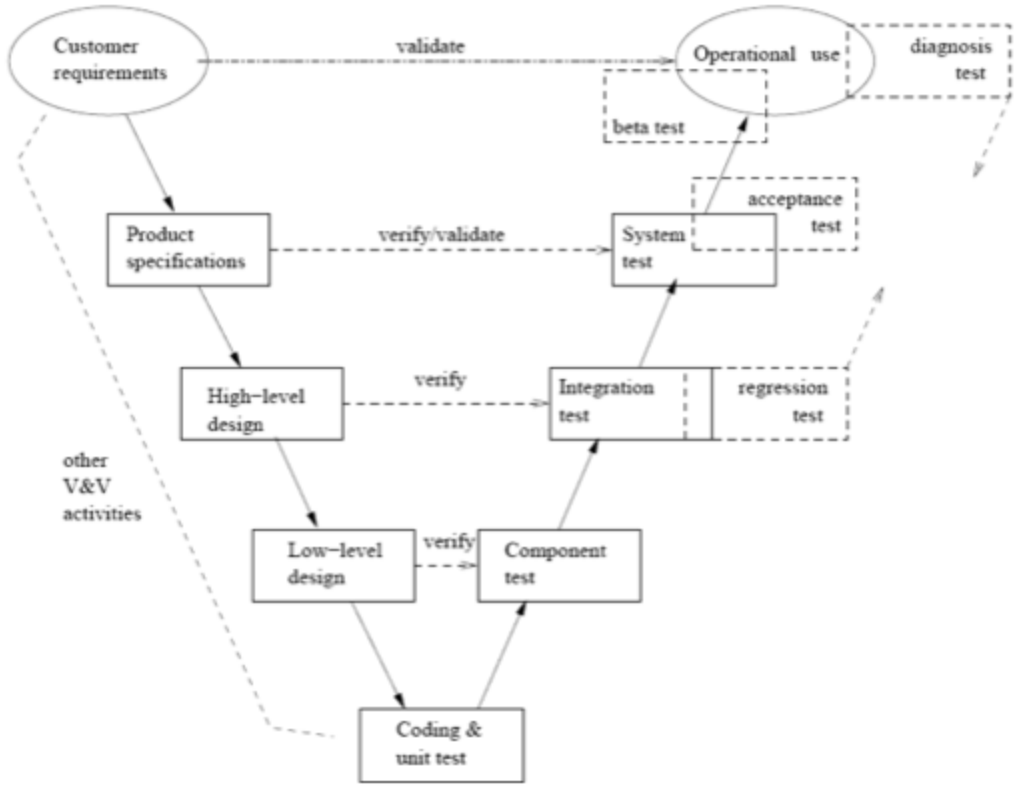

Abbildung 6.1: Testverfahren

Abnahmetest Black Box durch Kunden

Betatest Black Box durch Endanwender

6.2.1 Testwerkzeuge

Unit-Tests gibt es für viele Sprachen. Jedoch bleibt die Frage: Wie teste ich den Testcode? Der TestCode testet sich mit dem Programmcode gegenseitig.

Emulatoren und Simulatoren

Simulator Modell (Vereinfachung) eines Systems zum Zeck der Analyse dieses Systems

Emulator (Computertechnik) Nachbildung wesentlicher Verhaltensaspekte eines Systems durch ein anderes. Anmerkung: Das emulierte wie das emulierende System können aus Hardware und/oder Software bestehen

7 Verifikation und Validierung

Verifikation von Software Überprüfung ob Software ihrer Spezifikation entspricht bzw. den Spezifikationen entsprechend umgesetzt wurde.

Validierung von Software Überprüfung ob die Software den Anforderungen und Erwwartungen ihrer Benutzer entspricht - sind die Benutzeranforderungen also richtig und vollständig erfasst und umgesetzt worden.

Im Normalfall traegt die Verifikation zur Validierung bei, doch sind zusaetzliche Maßnahmen für die Validierung notwendig.

Fehlerquellen Ein Anwender beobachtet vermeintliches Fehlverhalten. Missverständnisse und tatsächliche Fehlverhalten müssen voneinander unterschieden werden, was bedeutet, dass der Fehler in der Spezifikation oder Umsetzung liegt.

7.1 Planung der Qualitätsprüfung

Prüfverfahren können statich oder dynamisch stattfinden und die Ziele sind Verifikation Entdeckung von Fehlern in der Software) und Validierung (Aufbau von Vertrauen in die Anwendbarkeit der Software).

7.2 Verifikationstechniken

Verifikationstechniken umfasst Softwareinstpektion, automatisierte statische Analyse, formale Methoden und Softwaretests.

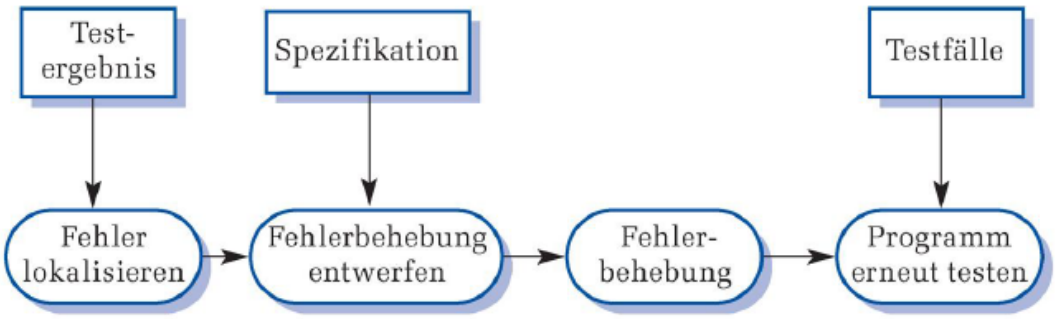

Abbildung 7.1: qualitaetscheck

7.2.1 Softwareinspektion

Fehlerlokalisierung auf Basis von Test ist eine Art von Inspektion und weitere Prüfziele wie z.B. Normen und Standards, Portiertbarkeit, Wartbarkeit, Eignung von Algorithmen und Programmierstiel könne verfolgt werden.

Voraussetzungen Voraussetzungen für die Inspektion sind eine genaue Spezifikation, Vertrautheit der Teammitglieder mit einzuhaltenden Standards und Normen sowie die Verfügbarkeit einer aktuellen zumindestens kompilierbaren Codeversionen.

Bei der Softwareinspektion gibt es mehrere Rollen für die Teilnehmer. Diese Umfassen Autor/Eigentümer, Inspektor, Vorleser, Protokollant, Vorsitzender/Moderator und Chefmoderator.

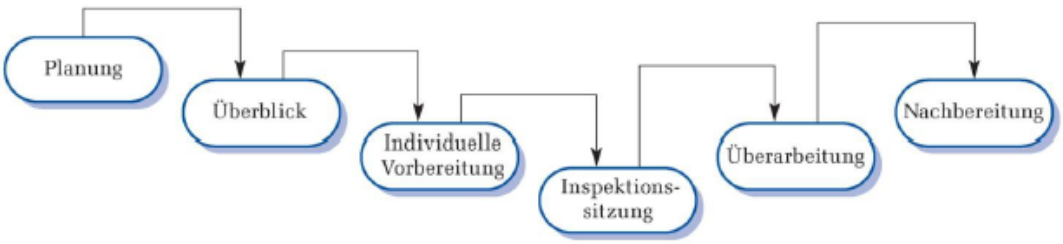

Abbildung 7.2: softwareinspektion

7.2.2 Verifikationstechniken

Automatisierte statische Analyse Softwareinspektion ist eine Form der statischen Analyse und wird oft mit Hilfe von Checklisten und Heuristiken verwendet.

Statische Analysewerkzeuge erkennen Anomalien im Code (nicht jede Anomalie ist ein Fehler) und diese sollten Gegenstand eines Reviews sein. Solche Anomalien können sein:

- nicht initialisierte Variablen
- nicht verwendete Variablen
- verletzte Indexgrenzen
- nicht deklarierte Variablen
- unerreichbarer Code
- falsch zugeordnete Parametertypen und/oder -anzahl
- nicht verwendete Funktionsergebnisse
- ...

Die Phasen der statischen Analyse umspannen die Analyse der Steuerung, unerreichbaren Code un die Pfadanalyse. Außerdem gibt es eine Analyse der Datenverwendeung (Lesen von nicht initialiserten Variabeln, redundante Tests), der Schnitstellen (z.B. Parameteranzahl in Deklaration und Aufruf) und des Informationsflusses (z.B. unbenutzte Eingaben).

Formale Methoden Die Vision ist, dass mit Hilfe formaler Methoden die ultimative statische Verfikation möglich ist. Voraussetzungen dafür sind eine vollständige formale Spezifikation. Dies würde die Möglichkeit bieten automatisierte Überprüfungen der Spezifikationen auf Inkonsistenzen zu machen und die Verifikation des Codes gegen Spezifikation ermöglichen.

Ein großes Problem dabei ist, dass die Spezifikation nicht die Benutzeranforderungen sind. Ein Weiteres ist die Komplexität und der Umfang der Beweise und unzutreffende Nutzungsmuster als Voraussetzung des Beweises (was den Beweis ungültig macht).

7.2.3 Validierungstechniken

Qualitaetssicherung der Anforderungen durch Anforderungsinterviews mit einer Vielzahl unterschiedlicher Beteiltigter bzw. Betroffener sowie Reviews der Anforderungen durch andere Personen die nicht die Anfoderungen erhoben haben (Experten des Anwedungsgebiets, Experten für Systemeigenschaften, ...).

Entwicklungsphase In der Entwicklungsphase sollten Design-Reviews für Interkations-Szenarios mit Beteiliten / Betroffenen gemacht werden (Powerpoint.

Validierungsphase In der Validierungsphase sollten Tests durch echte Anwender von möglichst allen Anwendergruppen gemacht werden und eine einfache Feedbackmöglichkeit bereitgestellt werden.

Einsatzphase Marktbeobachtungen, Benutzergruppen, Produktwiki

Validierung der Zuverlässigkeit Gibt Auskunft darüber wie groß ist der Anteil der korrekten Ergebnisse ist. Dabei gibt es verschiedene Probleme bei der statistischen Zuverlässigkeitsmessung:

- Unsicherheit über Betriebsprofil
- Kosten der Testdatenerstellung
- Statistische Unsicherheiten

Validierung der Betriebssicherheit Ist die Eigenschaft eines Systems, bei richtiger wie falscher Benutzerung und auch beim Auftreten externer Fehler kein Verhalten zu zeigen, das Benutzer oder Dritte schädigt oder gefährdet. Zuverlässigkeit sich zwar messen (ROCOF) aber nicht Betriebssicherheit.

Die formale Verifikation des gesamten Systems erforder eine vollständige Liste aller, auch externer Fehler –> ist also unmöglich und wäre auch sehr teuer und fehleranfällig. Die Vorgehensweise ist eine formale Definition einer Reihe von Bedrohungen in Form vom Prädikaten.

Bedeutung von Betriebssicherheit Unfälle sind zu selten um sie mit statistischen Verfahren in den Griff zu bekommen und sind sehr oft Verletzungen von "darf-nichtÄnforderungen.

- Einrichtung eines Systems zur Überwachung von Protokollierung von Unfällen (von der Gefahrenanalyse über die Testphase bis zur Systemvalidierung)
- Zuweisung von persönlicher Verantwortung für Betriebssicherheit an geeignete Personen
- Betriebssicherheit-Reviews während des gesamten Prozesses
- Formale Zertifizierung kritischer Komponenten

Validierung der Systemsicherheit Das Internet ist eine tolle Infrastruktur um von einem beliebigen Rechner auf ein beliebiges System auf irgendeinem Rechner irgendwo auf der Erde zuzugreifen / angreifen.

Es ist im allgemeinen vollkommen unmöglich nachzuweisen, dass ein System unter keinen Umständen etwas tut, was es nicht tun darf.

Überprüfungsansätze

- Werkzeuge für erfahrungsbasierte Validierung
- Checklisten
- angeheuerte, erfahrene Hacker
- Formale Verifikation

Keine Kette ist stärker als ihr schwächstes Glied!

8 Glossar

Verifikation Die Verifikation stellt fest ob die Software mit der vorhandenen Spezifikation übereinstimmt.

Validierung Die Validierung bestimmt ob die Software für den Kunden auch wirklich nützlich ist.

Technische Schuld Technische Schuld (technical debt) ist ein wichtiges Thema in der Qualitätssicherung. In der Softwareentwicklungen ist damit gemeint, dass ungeschickte/schlechte Lösungen irgendwann besser gelöst werden müssen.

Softwareprozesse Eine Abfolge von Tätigkeiten, durch die ein Software-Produkt entsteht.

Vorgehensmodell Vereinfachte Beschreibung eines Softwareprozesses

Methode Strukturierter Ansatz für die Software-Entwicklung

Software-Engineering Software-Engineering ist eine technische Disziplin welche eine Lösung für den Anwender bieten will unter Einsatz von Theorien, Methoden und Werkzeugen und Berücksichtigung von Organisation, Management und Entwicklung.

ESSENCE Kernel Overview In diesem Model gibt es verschiedene Zustände für die Anforderungen und jeder dieser Zustände hat selbst wieder Kriterien welche einem dabei helfen in welchem Zustein ein Projekt ist.

Qualität Grad in dem die inhärenten Eigenschaften des Produkts Anforderungen erfüllen.

Anforderungen Eine Anforderung ist eine Aussage über die notwendige Beschaffenheit oder Fähigkeit, die ein System oder Systemteile erfüllen oder besitzen muss, um einen Vertrag zu erfüllen oder einer Norm, einer Spezifikation oder anderen, formell vorgegebenen Dokumenten zu entsprechen. [1]

Implizite Anforderungen Anforderungen welche existieren aber den Stakeholdern nicht bewusst sind.

Nicht explizite Anforderungen Stakeholer wissen, dass sie diese Anforderungen gibt aber sie teilen diese nicht mit (sind quasis eh klar).

Objektive Anforderungen Vollkommen klar, dass man etwas braucht.

Subjektive Anforderungen Vermeintliche Anforderungen, welche nicht wirklich wichtig sind.

Technisches computer-basiertes System System welches ausschliesslich aus Soft- und Hardware-Komponenten besteht.

Soziotechnisches System System bestehend aus einem oder mehreren technischen Systemen, den Menschen die es bedienen, den notwendingen Arbeitsprozessen, organisatorischen Richtlinien.

Kritische Systeme Systeme, bei dessen Ausfall oder Fehlfunktion groÿen Schaden anrichten kann (wirtschaftliche Verluste, physische Schäden, Gefahr für Gesundheit und Leben von Menschen).

Sicherheitskritisches System Schäden an der Umwelt und/oder Gefahr für Gesundheit und Leben von Menschen (Bohrinsel im Golf von Mexiko).

Aufgabenkritisches System Aufgaben die ein System erledigen solle werden nicht durchgeführt (z.B. Bank).

[1] vgl.: `http://de.wikipedia.org/wiki/Anforderung` (12.06.2014)

Geschäftskritisches System Extrem hohe Kosten bzw. signifikante Gewinnausfälle können die Folge eines Systemausfalls sein.

Ethnographische Methode Die Grundidee ist das Beobachten ohne einzugreifen.

Review Mit dem Review werden Arbeitsergebnisse der Softwareentwicklung manuell geprüft. Jedes Arbeitsergebnis kann einer Durchsicht durch eine andere Person unterzogen werden. [2]

Nutzwertanalyseteam Bewertet den Nutzen für die Anwender.

Kostenanalyseteam Schätzt die Kosten jeder Anforderung

Kontextmodelle Definiert die Systemgrenzen des Gesamtsystems und des technischen Systems. Das Gesamtsystem umfasst das technische System und die menschliche Komponente und definiert den Kontext.

Verhaltensmodelle Definiert die Abläufe in einem System. Sie können Datenfluss (Datenfocus)- oder Ereignis (Ereignisfocus) - Orientiert geschehen.

Datenmodelle Definiert die logische und persistente (lokal, übergreifend mttels Datenbank und Datenaustausch) Datenstruktur.

Objektorientierte Modellierung Vereinigt die Funktionalität von Daten- und Verhaltensmodelle und können Daten, Datenflüsse, Datenstrukturen und Erreignise erfassen.

Strukturierte Methoden Detailliert definierte Vorgehensweise bei der SW-Entwicklung.Normalerweise basierend auf einem Satz von Diagrammtypen. Definiert zusätzliche Regeln und Richtlinien.

Anforderungsmanagementsystem Die Anforderungen an ein System ändern sich mit der Zeit, diese können ursprünglich unvollständig sein. Oder es verbessert sich das Verständnis des Problems / es ensteht eine bessere Sicht der Dinge. Aber auch das Umfeld kann sich verändern, z.B. wirtschaftlich, technisch, juristisch ...

[2] vgl.: `http://de.wikipedia.org/wiki/Review_(Softwaretest)` (12.06.2014)

Dauerhafte Anforderungen Sie sind relativ stabil und sind mit dem Kern der Anwendung verwoben. Oft aus Standardisierten Modellen entnohmen.

Veränderliche Anforderungen Anforderungen mit hoher Änderungswahrscheinlichkeit. Können wirtschaftliche Randbedingungen, technische Randbedingungen und gesetzliche Randbedingungen.

Pflichtenheft Zusammenstellung der vollständigen und detailierten Benutzeranforderungen und Systemanforderungen (groÿes Problem ist die Vollständigkeit). Darf nach Fertigstellung nicht mehr geändert werder und muss so implementiert werden wie es festgehalten wurde.

Funktionale Anforderungen Funktionale Anforderungen unterstützen Definition von Funktionen zur Fehlerprüfung, zur Wiederherstellung im Fehlerfall und Aspekte zum Schutz gegen Systemausfälle.

Nichtfunktionale Anforderungen Als nichtfunktionale Anforderungen werden u.a. die Systemzuverlässigkeit und Systemverfügbarkeit festgelegt.

Negativanforderungen Negativanforderungen Beschreibung von Verhalten oder Eigenschaften, die das System auf keinen Fall zeigen darf.

Risikomanagement Risikomanagement besteht aus Gefahrenbestimmung, Risikoanalyse und Gefahrenklassifizierung, Gefahrenvereinzelung und Festlegung zur Risikominimierung.

Risikoerkennung Ziel ist es zu Erkennen von Rsikiken und Gefahren. Probleme die dabei entstehenkönnen sind die Wechselwirkung zwischen Systemkomponenten bzw. die Wechselwirkungen mit der Umwelt.

Risikoanalyse und Risikoklassifizierung Analyse von Unfallwahrscheinlichkeit, Schadenswahrscheinlichkeit und Schadeshöhe.

Risikozerlegung Ziel es die Ursachen aufzudecken. Dazu können Reviews, Checklisten, Petri-Netze, formale Login und Fehlerbäume verwendet werden.

Risiko-Minimierung Ziel ist das Vermeiden des Auftretens von Gefahren. In der Praxis bedeutet dies eine Kombination aller Strategien.

Systemsicherheit Hierbei geht es um direkte und indirekte Berodhungen.

Direkte Bedrohung Z.B. unbefugtes Eindringen und DOS.

Indirekte Bedrohung Z.B. Aufwand um Sicherheitsmassnahmen zu installieren, konfigurieren und aktuell zu halten. Auch inkludiert sind Fehler und Nachlässigkeiten der Systemverwalter und monopolartige Stellungen einiger Hard- und Softwareanbieter da eine weite Verbreitung von Sicherheitslücken sehr wahrscheinlich ist.

Zuverlässigkeit Zuverlässigkeit besteht aus Hardware-, Software- und Bedienerzuverlässigkeit.

Formale Spezifikationsmethoden Formale Spezifikationstechniken sind gute Ergänzungen, eindeutig und präzise aber schwer verständlich für den Laien. Erzwingen die frühzeitige Analyse der Systemanforderungen wenn die Fehlerbehebung noch billig ist.

Ausnahmebehandlung Wird in manchen Sprachen mangelhaft Unterstützt (Beispiel: C). Kann auch dazu verwendet werden um die Lesbarkeit der Anwendung zu erhöhen und damit können Fehler vermieden werden.

Software-Reengineering Dabei handelt es sich um eine Neuentwicklung von Software, meistens eine Umstellung auf neuere Technologien (z.B. von COBOL auf Java).

Testing Betrifft die Kernel Alphas Softwaresystem, Requirements und Work. Tests sollen uns helfen Probleme zu entdecken und beheben, die Kunden überzeugen und die Softwarequalität demonstrieren. Dabei ist allerdings zu beachten, dass nur die Anwesenheit von Fehlern getestet werden kann, allerdings nicht ihre Abwesenheit.

Testprozesse Ein Testprozess muss geplant und vorbereitet werden. Dazu gehören die Definitionen der Testziele und verwendeten Messungen, sowie die Eingabewerte und Testsuites.

Vollständiges Testen Das vollständige Testen ist meist theorethisch möglich, aber nur in den seltensten Fällen wirtschaftlich. Unser Ziel muss es deshalb sein eine möglichst hohe Testabdeckung zu erreichen, und dabei auch noch wirtschaftlich zu bleiben.