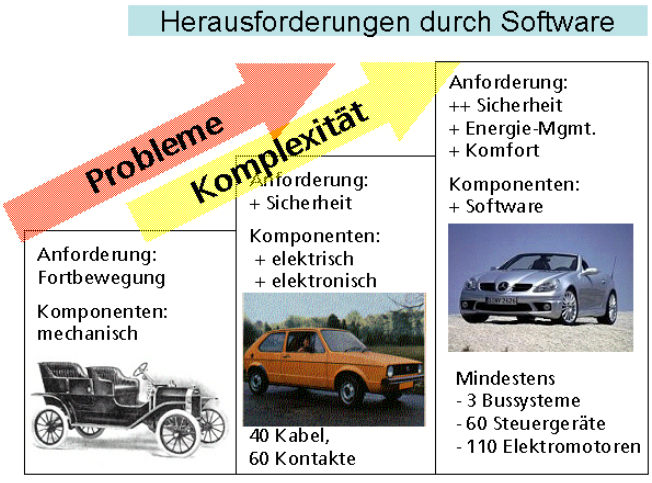
Fraunhofer Institut Experimentelles Software Engineering

Seminar:
Risikomanagement für sicherheitskritische Software-Systeme

Wintersemester 2004/05

Betreuer:
Prof. Dr. Dieter Rombach
Marcus Ciolkowski
Susanne Hartkopf
Reinhard Schwarz

Teilnehmer:
Bernd Burkhard
Mathias Int-Veen
Pascal Adrian Hagedorn
Florian Munz
Sebastian Weber



IESE-Report Nr. 031.05/D
Version 1.0
Januar 2005

Eine Publikation des Fraunhofer IESE

Das Fraunhofer IESE ist ein Institut der Fraunhofer-Gesellschaft.
Das Institut transferiert innovative Software-Entwicklungstechniken, -Methoden und -Werkzeuge in die industrielle Praxis. Es hilft Unternehmen, bedarfsgerechte Software-Kompetenzen aufzubauen und eine wettbewerbsfähige Marktposition zu erlangen.

Das Fraunhofer IESE steht unter der Leitung von
Prof. Dr. Dieter Rombach (geschäftsführend)
Prof. Dr. Peter Liggesmeyer
Sauerwiesen 6
67661 Kaiserslautern

Inhalt

Kapitel 1. Problemorientierte Vorgehensweisen

Bernd Bucrkhardt

Technische Universität Kaiserslautern
B_burckh@informatik.uni-kl.de

Kurzfassung. Risikomanagement gewinnt immer mehr an Bedeutung und damit auch Verfahren, die das Risikomanagement unterstützen. An vorderster Front werden hier problemorientierte Vorgehensweisen eingesetzt, um schon möglichst früh potentielle Risiken zu identifizieren, zu priorisieren und den Umgang mit diesen Risiken zu planen. Problemorientierte Ansätze sind Top-Down-Methoden, die mögliche Risiken der zu lösenden Aufgabe – dem Problem – in der Anwendungsdomäne analysieren, und die Auswirkungen dieser Risiken auf die Systementwicklung – der Lösung des Problems – untersuchen. Es existieren zwar schon zahlreiche Methoden aus den Bereichen der Chemie, dem Militär und der Luft- und Raumfahrt, jedoch sind nicht alle für den Einsatz in der Software-Entwicklung geeignet. Die hier vorgestellten Methoden sollen einen Einblick in existierende Verfahren und deren Einsatz in der Softwareentwicklung geben. Auch die Frage nach möglicher Toolunterstützung wird diskutiert.

1. Warum Risikomanagement

Tom Gilb fiel zum Thema Risiko folgendes ein:

> *„If you don't actively attack the risks, the risk will actively attack you."* [1:343]

Demnach sollte man angemessene Maßnahmen ergreifen, um dem Risiko möglichst früh zu begegnen. Risikomanagement bietet diese Möglichkeit, pro aktiv mit Risiken umzugehen.

Worum geht es beim Risikomanagement genau? Was versteht man überhaupt unter einem Risiko? Und wie kann man es einordnen? Diese Fragen sollen im Folgenden geklärt werden, um einen groben Einblick in das Risikomanagement zu geben.

1.1. Illustration

> *„The vessel Baltic Star, registered in Panama, ran aground at full speed on the shore of an island in Stockholm waters on account of thick fog. One of the boilers had broken down, the steering system reacted only slowly, the compass was maladjusted, the captain had gone down into the ship to telephone, the lookout man on the prow took a coffee break, and the pilot had given an erroneous order in English to the sailor who was tending the rudder. The latter was hard of hearing and understood only Greek."* [2]

Diese kleine Anekdote soll die Komplexität von selbst einfachen Aufgaben verdeutlichen. Oft werden Unfälle allzu sehr vereinfacht, und es wird nicht genau untersucht, was im einzelnen Einfluss auf den Unfall hatte. Ein Modell, das diesen verschiedenen Einflussfaktoren Rechnung trägt ist in Abbildung 1-1 zu sehen. Zudem wirft die Anekdote zahlreiche Fragen auf, die die verschiedenen Aspekte dieses Modells betreffen. Im Zentrum dieses Modells steht die Aufgabe (Msn), etwa das Schiff sicher an Land zu bringen. Der Nebel wäre dem Modell zufolge dem Umfeld (Media) zuzuordnen.

Warum das Schiff nun auch bei Nebel volle Fahrt halten muss, oder die Mannschaft keine gemeinsame Sprache spricht, ist dem Management(Mgt) zu zuschreiben. Weshalb ein Wachoffizier nicht auf seinen Posten ist, betrifft die menschliche Seite (Man) des Modells. Wieso der Kompass defekt ist, ist eine Frage der Technik oder auch der Software (Mach.), je nachdem welche Art von Kompass installiert ist.

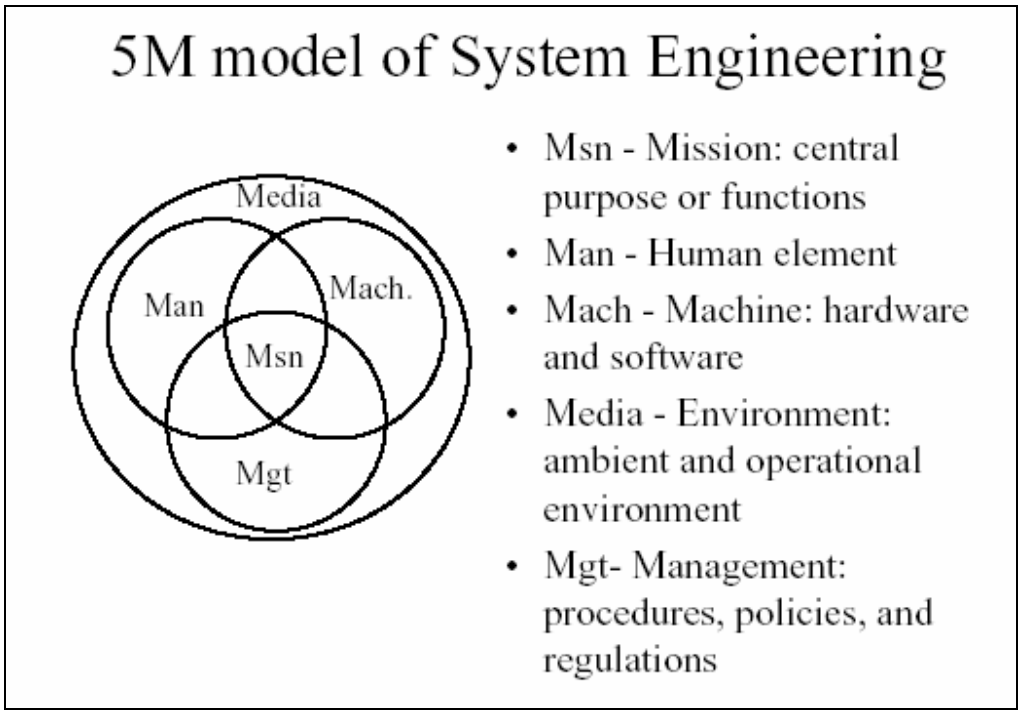


Abbildung 1-1. 5M-Modell aus [3]

1.2. Definition

Risiko kommt aus dem italienischen und bedeutet ‚etwas wagen'. Damit lässt sich alleine zunächst wenig anfangen. Bringt man nun noch eine Ursache (wie etwa schlechte Sicht) und eine Konsequenz (wie etwa in letzter Instanz das Auflaufen eines Schiffes) ins Spiel, kann man das Ereignis (etwa das Auftreten von Nebel) als Risiko ansehen.

Eine einheitliche Definition für ein Risiko gibt es zwar nicht, jedoch formuliert es Wallmüller wie folgt:

> *„Risiko ist die Möglichkeit, Verluste durch ein Ereignis, das mit gewisser Wahrscheinlichkeit eintritt, zu erleiden."* [1:343]

Möchte man nun etwa den Verlust eines Schiffes vermeiden, muss man lediglich alle Risiken identifizieren, deren Resultat das Sinken des Schiffes ist. Das alleine hilft noch nicht, den Verlust zu vermeiden, aber man kann analysieren, warum das Schiff sinken könnte. Ausgehend davon kann man entscheiden, wie man mit den einzelnen Risiken umgeht. Man kann zum Beispiel Schutzmassnahmen ergreifen (wie etwa einen Ersatzkompass) oder kann das Risiko ganz vermeiden (wie etwa eine Mannschaft, die keine gemeinsame Sprache spricht). Manche Risiken lassen sich jedoch nur schwer greifen und schon gar nicht vermeiden, wie etwa Nebel oder noch schlimmer eine Flutwelle. Für letzteres kann man sich bestenfalls versichern.

Um genau diese Dinge geht es nun beim Risikomanagement. Wallmüllers beschreibt es mit folgenden Worten:

> *„Risikomanagement ist ein systematischer Prozess zur Identifikation, Analyse und Kontrolle von Risiken in Projekten oder Organisationen.“* [1:344]

1.3. Risikoeinordnung

Um nicht den Überblick zu verlieren, bzw. entscheiden zu können welches Risiko wichtiger ist, muss man sich Gedanken um Prioritäten machen. Um diese Prioritäten besser festlegen zu können, ist es hilfreich die Risiken erst einmal zu klassifizieren. Schließlich sieht nicht jeder Mensch das gleiche als wichtig an (für den einen mag es wichtiger sein den Wachoffizier zu strafen für den anderen ist die gemeinsame Sprache der Mannschaft wichtiger).

Wie kann man nun ein Risiko einordnen? Was macht es aus? Generell kann man ein Risiko recht gut mit zwei Faktoren greifen: der Eintrittswahrscheinlichkeit des Risikos und der Schwere seiner Konsequenz. Tabelle 1-1 zeigt eine Klassifizierung des U.S. Militärs für die Risikokonsequenz und Tabelle 1-2 zeigt eine Einordnung für die Eintrittswahrscheinlichkeit [4].

Description	Category	Definition
Catastrophic	I	Death, and/or system loss, and/or severe environmental damage.
Critical	II	Severe injury, severe occupational illness, major system and/or environmental damage.
Marginal	III	Minor injury, minor occupational illness, and/or minor system damage, and/or environmental damage.
Negligible	IV	Less then minor injury, occupational illness, or less then minor system or environmental damage.

Tabelle 1-1. MIL STD 882C: Severity of Consequences

Description	Level	Definition
Frequent	A	Likely to occur frequently
Probable	B	Will occur several times in the life of system.
Occasional	C	Likely to occur some time in the life of the system.
Remote	D	Unlikely but possible to occur in the life of the system.
Inprobable	E	So unlikely, it can be assumed that occurrence may not be experienced.

Tabelle 1-2. MIL STD 882C: Event Likelihood (Probability)

Ausgehend von diesen Definitionen lässt sich nun ein Portfolio erstellen, um Risiken einordnen zu können, wie es Tabelle 1-3 zeigt. Damit lassen sich nun identifizierte Risiken priorisieren und es damit auch entsprechend ihrer Bedeutsamkeit behandeln.

Frequency of Occurrence	Severity			
	(1) Catastrophic	(2) Critical	(3) Marginal	(4) Negligible
(A) Frequent	1A	2A	3A	4A
(B) Probable	1B	2B	3B	4B
(C) Occasional	1C	2C	3C	4C
(D) Remote	1D	2D	3D	4D
(E) Improbable	1E	2E	3E	4E

Risk Categories: High, Serious, Medium, Low

Tabelle 1-3. MIL STD-892C Portfolie for risk classification

2. Problemorientierte Risikomanagement Methoden

Generell kann Risikomanagement Methoden für Softwareentwicklung in zwei Gruppen einteilen: strukturorientierte und problemorientierte Ansätze. Die strukturorientierten Ansätze beinhalten hauptsächlich Reviewtechniken, die sich auf die Analyse von vorhandener Dokumentation oder Entwürfen stützt (die jedoch möglicherweise schon Fehler beinhalten, bzw. Risiken nicht identifiziert haben). Problemorientierte Ansätze werden früher eingesetzt, und können deswegen nur selten auf vorhandene Entwürfe oder Dokumentationen zurückgreifen. Bevor man sich nun mit einzelnen problemorientierten Methoden beschäftigt, muss man sich daher zunächst über die Eigenarten von problemorientierten Ansätzen Gedanken machen. Wann und Wo im Entwicklungsprozess ist ihr Einsatz sinnvoll? Welchen Herausforderungen müssen sie sich stellen? Welche Anforderungen ergeben sich für den Umgang mit den Ansätzen?

2.1. Wann und wo?

Um über das Wann und Wo sprechen zu können, muss man sich zunächst einmal den Software-Entwicklungsprozess vor Augen führen. Ein vereinfachtes V-Modell (Abbildung 1-1) komprimiert die einzelnen Schritte auf das Problem des Kunden, Analyse dieses Problems (mit entsprechender Dokumentation), den Entwurfsprozess auf Basis der Analyseergebnisse, anschließend die Implementierung des Entwurfs und die abschließende Integration der einzelnen Komponenten zum lauffähigen System. Ähnliches, was für das Auffinden von Fehlern bei solchen Prozessen gilt, lässt sich auch auf das Risikomanagement übertragen: „Je früher, desto besser“.

Somit kann man zum Wann sagen: Problemorientiertes Risikomanagement sollte so früh wie möglich stattfinden. Schließlich kann man sich nur auf solche Risiken angemessen vorbereiten, die man auch vorher erkannt hat. Entdeckt man bei der Integration ein untragbares Risiko (beispielsweise eine gravierende Sicherheitslücke), so muss unter Umständen der komplette Entwurf überarbeitet werden. Hätte

man nun schon bei der Analyse festgestellt, dass das Kunden-Problem ein hohes Maß an Sicherheit erfordert, dann hätte man sich mehr auf mögliche Sicherheitslücken konzentrieren können.

Somit lässt sich nun auch das Wo beantworten: Problemorientiertes Risikomanagement konzentriert sich auf das Kundenproblem, die Analyse und Dokumentation des selbigen; also das Problem und die Analysephase in Abbildung 1-1.

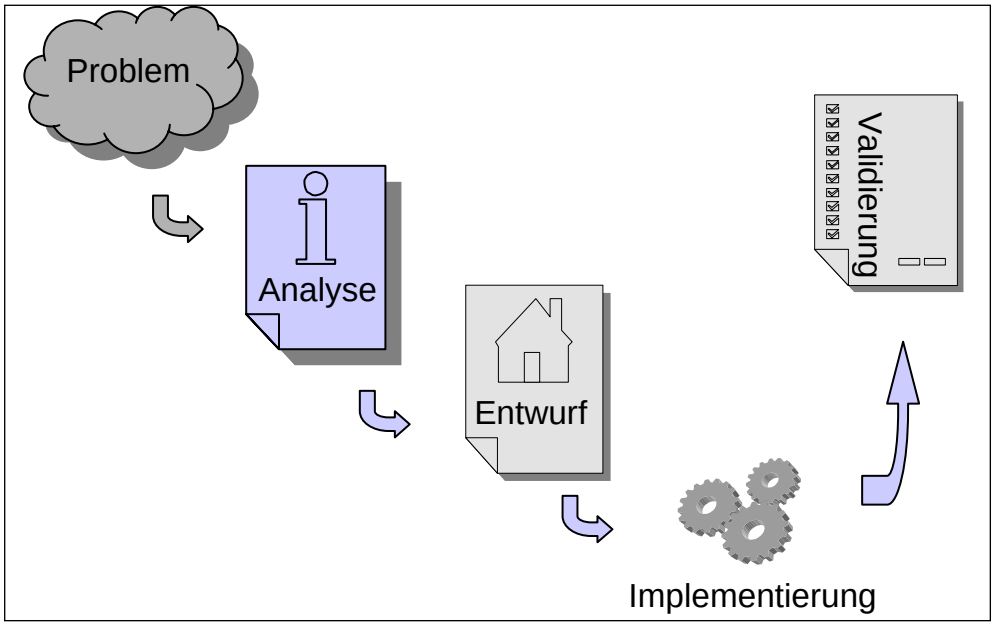


Abbildung 1-1. Vereinfachte Darstellung des V-Modells. In der Analysephase werden die bekannte Problembeschreibung, Benutzeranforderung und Entwickleranforderung erstellt. In der Validierungsphase, werden die einzelnen Systemkomponenten integriert und gegen die Anforderungen validiert.

2.2. Herausforderungen

Mit welchen Herausforderungen man es hierbei zu tun hat, lässt Abbildung 1-2 erkennen. Das Problem, und damit auch eine implizite Vorstellung der Lösung, ist zunächst nur ein fiktives, abstraktes Konstrukt im Kopfe des Kunden bzw. seines Domänenexperten. Tritt nun ein solcher an einen ‚Problemlöser' – hier einen Softwareingenieur - heran um das Problem zu lösen (also ein System dafür zu entwickeln), treffen zwei verschiedene Wissenshorizonte aufeinander. Der Problem-Experte kennt sich sehr gut mit dem Problem und seinen Risiken aus, der Softwareingenieur kennt sich mit seiner Welt ebenfalls sehr gut aus, kann jedoch nur von außen auf den Horizont des anderen blicken. Damit ist die Hauptherausforderung ein ‚sich mit dem anderen konstruktiv Zusammenraufen'. Am Ende entsteht somit ein gemeinsamer Horizont, in den jeder sein Expertenwissen eingebracht hat um die domänenspezifischen Risiken zu entdecken.

Diese Herausforderung erscheint umso größer wenn man die Geschichte des Risikomanagements betrachtet. Die meisten Methoden und strukturierten Vorgehensweisen kommen aus der Chemie-, Kraftwerksbranche und dem Militär. Dort kommen der Problemexperte und der Problemlöser oft aus der gleichen Domäne, daher müssen viele Methoden angepasst werden, bzw. muss der Nutzer der Methode für diesen Umstand sensibilisiert sein um dieser Herausforderung begegnen zu können.

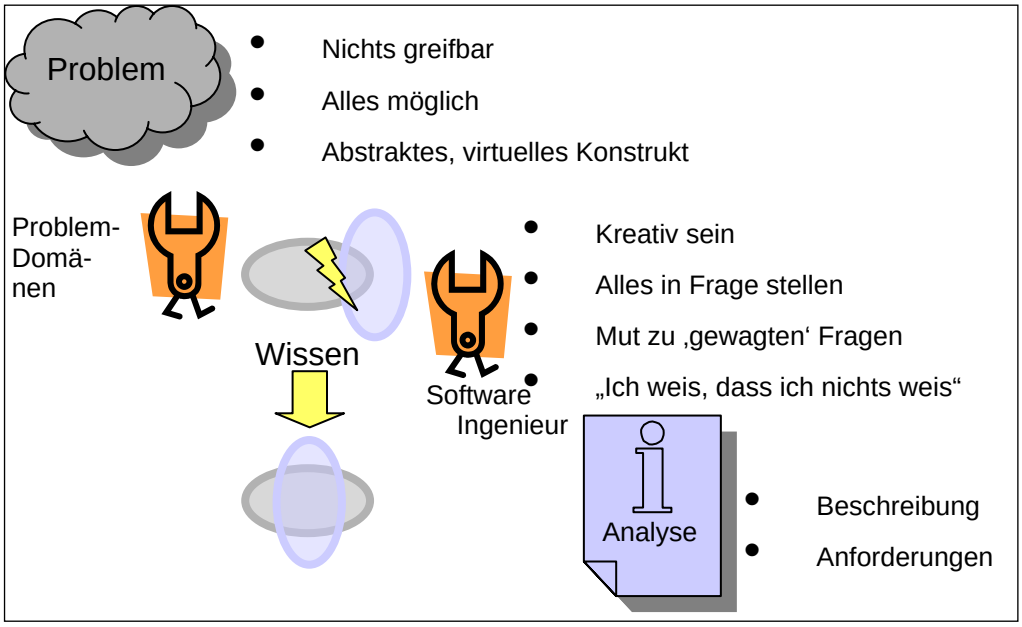


Abbildung 1-2. Veranschaulichung der Besonderheiten in der Analysephase, aus Sicht des Softwareingenieurs (rechtes Männchen). Der Softwareingenieur muss sich mit dem Problemexperten zusammen über das Problem klar werden. Für den Softwareingenieur ist das Problem der Ausgangspunkt, der analysiert wird und dann in Beschreibungen und Anforderungen umgesetzt wird. Hierbei sind die einzelnen Wissenshorizonte eingezeichnet und die üblicherweise nur sehr schmale Schnittstelle zwischen beiden. Schmal bedeutet hier, dass der Problemexperte noch implizites Wissen über Gefahren und Risiken seiner Domäne hat, die dem Softwareingenieur nicht bewusst sind. Das Ziel ist es nun sich soweit anzunähern, dass dem Softwareingenieur alle für die Problemlösung relevanten Risiken und Gefahren bewusst werden.

2.3. Anforderungen

Der Softwareingenieur muss eine Vielzahl von Softskills mitbringen, um mit Hilfe geeigneter Methoden auch den erwünschten Nutzen zu erzielen. Generell sollte er alles in Frage stellen und sich sehr kritisch mit dem Thema auseinandersetzen. Zum einen muss er das Problem genau hinterfragen, zum anderen darf er sich nicht zu schnell dem ‚Ich-Weiß-Bescheid' Gefühl hingeben.

Er sollte demnach nur in seinem Wissenshorizont Schlussfolgerungen ziehen und es dem Problemexperten überlassen, diese in der Problemdomäne zu ziehen. Selbst einfach erscheinende Teillösungen, wie etwa das Nutzen von Regenreifen um Aquaplaning zu begegnen, kann in manchen Bereichen nicht möglich sein (siehe 2.1 What-If: Beispiel). Dabei ist Selbstsicherheit und Mut gefragt, um auch Fragen zu stellen, die einem selbst trivial, lächerlich oder peinlich vorkommen mögen. Oft sind es solche Fragen, die den Problem-Experten dazu bringen über Dinge zu reden, die er sonst für nicht erwähnenswert halten würde, oder sogar völlig übersehen hätte. Denn auch der Problemexperte muss für die möglichen Risiken sensibilisiert werden, um ein Bewusstsein für diese Risiken zu entwickeln. Ein hohes Maß an kreativer Vorstellungskraft und flexibles Denken helfen sich in das Problem, und damit in die Welt des Problemexperten, ‚hinein zu denken'.

2.4. Methoden

Da es eine Vielzahl problemorientierter Methoden gibt, zwar nur wenige, die auf software-spezifische Probleme eingehen, sind es zu viele, um alle vorzustellen. Daher werden hier drei Vertreter vorgestellt, um einen Einblick in dieses Feld zu geben. Die ersten beiden Methoden kann man durchaus als Grundlagen für fast alle anderen (hier nicht behandelten) ansehen, die letzte soll einen Ausblick über eine sinnvolle Integration mehrer Methoden in den Software-Entwicklungsprozess geben.

Bei jeder Methode wird auf ihre *Anwendung*, das *Verfahren* selbst und die *Anforderungen* an den Softwareingenieur eingegangen. Abschließend wird ein kleines *Beispiel* zu der Methode gegeben und auf ihre *Besonderheiten* eingegangen.

What-if Analysis

Die What-If Analysis ist eine Brainstorming-Methode die aus der Chemieindustrie kommt. Ihr Ergebnis ist eine unbewertete Liste von Risiken, deren Konsequenzen und möglichen Präventionen und/oder Alternativen.

Anwendung. Da es sich hier nicht primär um eine Reviewtechnik handelt, eignet sich diese Methode sehr gut, um in der initialen Problemfindung eingesetzt zu werden, also als Risikoanalysemethode in der Problemanalysephase (siehe Tabelle 1-4).

Diese Methode unterstützt somit in einer unstrukturierten Form den Annäherungsprozess von Problemexperte und Softwareingenieur, da beide an der Durchführung dieser Methode beteiligt sind. Da es sich um eine reine Brainstorming-Technik handelt und es keine Richtlinien oder ähnliche Strukturen gibt, sind die Ergebnisse stark von der Erfahrung des Teams abhängig.

Generell kann man die What-If Analysis auch in fast jeder Phase des Software-Entwicklungsprozesses anwenden. Aufgrund ihrer unbewerteten Resultate und stark vom Analyseteam abhängigen Ergebnisqualität ist sie jedoch nicht unbedingt für jede Phase geeignet.

Verfahren. Ein Team von Personen, die mit dem Problem vertraut sind, stellt hier Fragen oder diskutiert mögliche unerwünschte Ereignisse, die bei der Problemlösung auftreten könnten oder mit dem Problem an sich einhergehen. Üblicherweise gehören mindestens ein Problemexperte sowie ein Experte in der Analyse-Methode (als Moderator) zum Team.

Die Methode verlangt nun vom Team, ‚What-If' Fragen durchzudenken. Was könnte passieren, wenn…? Bei diesem Frageprozess identifiziert das Team auch mögliche Konsequenzen und schlägt potentielle Präventionen oder Alternativen vor. Je nach Einsatzstadium kann man sich auch auf schon vorhandene oder erstellte Szenarien beziehen.

Das Ergebnis ist in der Regel eine Liste dieser What-If's und ihrer Konsequenzen und Präventionen, ohne dass diese qualitativ beurteilt oder klassifiziert werden.

Anforderungen. Die Mitglieder des Analyseteams müssen sich mit Ingenieur- und Prozessabläufen auskennen und mindestens ein Experte aus der Problemdomäne und ein Experte in der What-If Analysis müssen dazugehören. Für kleinere Probleme eignet sich eine Teamgröße von zwei bis drei Personen. Für komplexere Probleme und damit auch komplexere Systeme werden hier natürlich mehr Personen gebraucht. Bei großen Teams ist es zudem empfehlenswert, die Analyse wenn möglich aufzuteilen und logische Problemzweige auf kleinere Teams zu verteilen.

Beispiel. Als Beispiel soll hier der Warschau Aircrash [5] dienen und aufgezeigt werden, dass sich der Fehler in diesem System durch diese Methode hätte entdecken lassen.

Bei der Analysephase hätte sich das Teams zum Thema Einflüsse bei der Landung zum Beispiel folgende Fragen einfallen lassen können:

Was könnte passieren, wenn…:

1) …Seitenwind auftritt?
2) …starker Regen die Landebahn nässt?

Dies wird in der Regel mit den weiteren Konsequenzen und Präventionen in einer Tabelle (siehe Tabelle 1-4) festgehalten.

Als Prävention zu 2) hätte sich ein Softwareingenieur die Benutzung von Regenreifen, wie er es bei seinem Auto gewohnt ist, einfallen lassen können und sich dafür auch ein eigenes Szenario ausdenken können. Aufgrund der Benutzerfreundlichkeit kann der Problemexperte dieses Ergebnis sehr leicht analysieren und somit sein Wissen einbringen, dass dies eben keine durchführbare Prävention bei Flugzeugen ist (die Querrillen bei Regenreifen würden bei einer Flugzeuglandung einfach abgerissen werden).

What-If	Konsequenzen	Präventionen/ Alternativen	Szenario	Anmerkungen
… Seitenwind auftritt ?	Ungleiche Gewichtsverteilung auf dem Fahrwerk	Keine		
… starker Regen die Landebahn nässt ?	Aquaplaning	Spezialreifen	2a	Regenreifen beim Flugzeug nicht möglich

Tabelle 1-4. Beispiel einer Ergebnisdokumentation der What-If Analyse auf Basis des Wahrschau Aircrash's [5]

Besonderheiten. Die What-If Analysis kann eine sehr nützliche Methode sein, wenn das Team sehr erfahren und das Problem nicht zu komplex ist. Aufgrund der Unstrukturiertheit neigt diese Brainstorming-Methode ansonsten dazu, zu viele (unnötige) Resultate zu produzieren und damit auch einzelne Aspekte zu übersehen.

Kleinere Analyseteams sind in der Regel effizienter, da es dort kürzere Diskussionen gibt.

Vorteile sind hier die Einfachheit der Methode und damit auch die günstige Anwendung bei nicht zu großen/komplexen Problemen. Die ‚Benutzerfreundlichkeit' hilft die Methode in frühen Analysephasen mit dem Problem-Experten durchzuführen, ohne dass dieser explizite Erfahrung mit der Methode haben muss.

Nachteile sind hier ganz klar der starke Teameinfluss auf die Ergebnisqualität und die Gefahr, dass lange Diskussionen unnötige Resultate liefern und wichtige Aspekte verloren gehen. Dies geschieht vor allem bei größeren und komplexeren Problemen.

Checklist Analysis

Die What-If/Checklist Analysis ist eine einfache, seit Jahren bewährte Analysemethode im Rahmen der Process Hazard Analysis der OSHA[1]. Sie baut auf der What-If Analysetechnik auf und strukturiert diese mit Hilfe von Checklisten. Diese Kombination ergibt ein mächtiges, vielseitiges und flexibles Analysewerkzeug, das je nach Checkliste effektiv eingesetzt werden kann. Die Ergebnisse werden in der Regel wie die der What-If Analysis festgehalten, jedoch erlauben Checklisten eine objektive Bewertung.

Anwendung. Durch die Checklisten ergibt sich ein sehr weites Einsatzspektrum. Überall wo eine What-If Analysis möglich ist, kann auch eine Checklist Analysis eingesetzt werden. Außerdem eignet sie sich auch als Review Tool, um zum Beispiel eine Problembeschreibung zu den Benutzeranforderungen zu verfeinern und dabei potentielle Risiken zu entdecken. Somit bietet sie einen problemorientierten Ansatz, der beliebig stark strukturiert werden (je nach Strenge der Checklisten – Führungsleine versus Würgestrick) und früh zum Einsatz kommen kann.

Darüber hinaus bietet sich auch der Einsatz in anderen Phasen/Bereichen an:

- *Design Stages*: Entwurfsrichtlinien und Entwurf-Reviews in einzelnen Phasen, um Vollständigkeit zu gewährleisten
- *Construction*: Reviews in allen passenden Konstruktionsphasen, um Vollständigkeit zu gewährleisten
- *Maintenance*: Präventiv, als Folge einer Störung oder eines Ausfalls

Verfahren. Als Kombination aus der What-If Analysis und der reinen Checklist Analysis ist die What-If/Checklist Analysis in fünf Schritte unterteilt:

1) Vorbereitung zum Review
2) Liste von What-If's und Ursachen erstellen
3) mit Checkliste Lücken überbrücken/Checklisten konsultieren
4) Auswerten der Ergebnisse
5) Dokumentieren der Ergebnisse

Diese fünf Schritte dienen als Basis und können je nach Einsatzgebiet noch leicht variiert werden.

Schritt 2 und 3 werden oft vertauscht, um mit Checklisten Leitfragen zu entwickeln. Auch können diese Schritte iterativ verfeinert werden (top-down Ansatz). Damit kann der Teamleiter in einem initialen Schritt einzelne Problemaspekte identifizieren, um dann mit Hilfe angepasster Checklisten die einzelnen Problemfelder aufzudecken. Diese können dann von separaten Teams analysiert werden können.

1) Vorbereitung: Der Analyseteam-Leiter wählt sein Team aus und berücksichtigt dabei die Domäne des zu analysierenden Problems. Ebenso sollte ein Experte aus dieser Domäne in das Team integriert werden. Große Probleme sollten generell in Teilprobleme aufgegliedert werden, bei nicht zu komplexen Problemen kann dies schon in diesem Schritt geschehen. Der Teamleiter hat nun auch die Aufgabe, die

[1] Occupational Safety and Health Administration – U.S. Department of Labor with the responsibility to ensure safety and healthful work environments.

Checklisten auszuwählen. Diese sollten an das Problem und Vorgehen angepasst werden. Wie etwa Metachecklisten, für iteratives Vorgehen oder spezielle Checklisten, bei überschaubaren Problemen. Auch hier kann dem Teamleiter eine Checkliste helfen, um das Problem vorab zu klassifizieren.

2) What-If's erstellen: Nach der Erstellung einiger Leitfragen beginnt das Team mit der schon bekannten What-If Analysis.

3) Checklisten konsultieren: Anschließend werden die Checklisten herangezogen, um mögliche Lücken zu überbrücken und/oder neue mögliche Probleme zu erkennen. Sollten neue Probleme auftreten, wird für diese wieder mit Schritt 2 begonnen.

Wie schon erwähnt stellt sich hier die Frage, ob das Team die Checklisten als Führungsleine oder eher als Würgestrick ansehen wird. Zuviel Vorschrift kann die Kreativität einschränken, verhindert aber auch, dass gewisse Standarda unbemerkt bleiben. Daher werden die Checklisten benutzt, um mit ihrer Hilfe What-If-Fragen zu generieren (Schritt 3 vor 2) oder man überprüft die gefunden What-If's mit Hilfe der Checklisten (Schritt 2 vor 3). Hier ist die Kompetenz des Teamleiters gefragt, er muss entscheiden, welches Vorgehen für sein Team und die Aufgabe am besten geeignet ist.

4) Auswertung: Die entdeckten Risiken werden nun vom Team bezüglich ihrer Ereigniswahrscheinlichkeit und ihres Schadenspotentials analysiert. Das Team listet zu jedem Risiko existierende Sicherheitsrichtlinien zur Risikominimierung, -vermeidung oder -akzeptanz aufgelistet (siehe auch [1:352] Planung von Kontrollmaßnahmen)auf, und entscheidet welche Reaktion auf dieses Risiko empfohlen wird. Häufig wird diese Analyse von speziellen team-externen Mitarbeitern durchgeführt und danach vom Team überprüft bzw. übernommen.

5) Dokumentation: Diese Ergebnisse werden dann ganz ähnlich der What-If Analysis in einer tabellarischen Form festgehalten. Zudem können hier Checklisten eingesetzt werden, um eine Klassifizierung vorzunehmen. Angaben über die benutzten Checklisten können die Vollständigkeit der Ergebnisse belegen, wenn es entsprechende dokumentierte Erfahrungswerte gibt.

Anforderungen. Die Hauptanforderungen beziehen sich hier auf die Kompetenz des Teamleiters. Die Teammitglieder sollten die üblichen Kenntnisse in Ingenieurtechniken und Prozessen haben. Darüber hinaus sollte sich das Team mit den Projektcharakteristiken und Zielen auskennen um sowohl technische wie auch administrative Aspekte bei der Analyse berücksichtigen zu können. Spezielle Kenntnisse in der Analysemethode sind nicht notwendig, vielmehr ist hier Kreativität und der angemessene Umgang mit den Checklisten gefragt.

Beispiel. Hier kann man sich wieder des Beispiels aus 2.2 bedienen. Eine Checkliste hätte als Ausgangspunkt für die entdeckten What-If's dienen können. Diese hätte zum Beispiel Bezug auf das 5M-Systemmodell nehmen können und folgende Punkte zur Umwelt enthalten können:

- Umwelteinflüsse:
- Wetterphänomene (Regen, Wind, etc.)
- Katastrophen (Erdbeben, etc.)
- Interaktion mit 3. (Kommunikation mit Menschen aus dem Umfeld – Fluglotsen)
- Interaktion mit Technik von 3. (Eingesetzte Roboter, etc.)
- Interaktion mit Tieren (Vogelschwärme, etc.)

Nun wäre es noch möglich, die beiden identifizierten Risiken, Seitenwind und regennasse Rollbahn, zu klassifizieren. In beiden Fällen sind zunächst keine Menschenleben in Gefahr, es kann jedoch zu gefährlichen Verletzungen führen. Daher könnte

man es nach dem MIL STD-882C in Klasse II oder sogar I einordnen. Was die Eintrittswahrscheinlichkeit angeht, so hängt diese stark von den Wetterbedingungen am Standort ab. Bezieht man sich auf Warschau, so kann man sagen, dass es öfters eine nasse Fahrbahn gibt als starken Seitenwind; also könnte man den Regen mit B und den Seitenwind mit Eintrittswahrscheinlichkeit C klassifizieren. Damit ergibt sich für den Regen die Risikoklassen IIB und für den Seitenwind IIC. Konsultiert man nun ein Portfolio wird man das Regenproblem als das zunächst wichtigere oder schwerwiegendere betrachten.

Besonderheiten. Generell gelten hier die gleichen Eigenheiten wie schon in der What-If Analysis. Die Methode kann sehr einfach aber auch fordernd sein, je nach Größe und Komplexität des Problems. Ein Team, dessen Mitglieder aus unterschiedlichen Fachbereichen kommen ist eher in der Lage, mit größeren interdisziplinären Problemen umzugehen.

Vorteile sind die Strukturiertheit der Methode und damit ihre bessere Aufteilbarkeit. Dank der Checklisten ist sie vielseitig einsetzbar und es sind keine expliziten Methodenkenntnisse nötig. Zudem erlauben die Checklisten einen dokumentierten Lerneffekt über mehrere Projekte.

Nachteile entstehen durch zeit- und arbeitsintensive, komplexe Probleme und den starken Einfluss der Checklisten auf die Ergebnisqualität. Um dem entgegen zu wirken bedarf es eines gewissen Pflegeaufwandes für die Checklisten.

Checklisten sind der wesentliche Kern dieser Methode, deswegen ergeben sich einige wichtige Besonderheiten. Mit ihrer Hilfe lassen sich Aussagen über die Abdeckung, also Vollständigkeit einer Analyse treffen, allerdings nur, wenn die angewandten Checklisten entsprechend gewartet wurden. Dies setzt ein entsprechendes Knowledge-Management voraus. Die Checklisten müssen regelmäßig gewartet werden und die Ergebnisse ihrer Verwendung ebenfalls in einer angemessenen Form gepflegt werden. Bemerkt man etwa, dass Items einer Checkliste nicht mehr zu gewünschten Ergebnissen führen, müssen diese überarbeitet werden.

Auch stellt sich die Frage, wie man zu solchen Checklisten kommt. Zum einen kann man sie extern besorgen und dann verwenden und anpassen. Ein anderer Weg wäre das Selbsterstellen anhand gängiger Richtlinien. So sollte eine Checkliste übersichtlich und einfach gehalten sein. Aber auch hier gibt es eine Vielzahl von unterschiedlichen Richtlinien, aus denen man einen Unternehmensstandard extrahieren kann. Zudem können auch Checklisten zum Erstellen von Checklisten herangezogen werden (Metachecklisten). Ebenso nutzt man Metachecklisten, um problemspezifische Checklisten zu finden oder zu erstellen.

Ein Beispiel hierfür ist SEI Taxonomy-Based Risk Identification (Abbildung 1-3)[6]. Daraus lässt sich eine Checkliste mit Fragen etwa für den Anforderungsbereich ableiten (Abbildung 1-4). Weitere nützliche Checklisten sind unter [7] zu finden.

A. Product Engineering
1. Requirements
   a. Stability
   b. Completeness
   c. Clarity
   d. Validity
   e. Feasibility
   f. Precedent
   g. Scale
2. Design
   a. Functionality
   b. Difficulty
   c. Interfaces
   d. Performance
   e. Testability

B. Development Environment
1. Development Process
   a. Formality
   b. Suitability
   c. Process Control
   d. Familiarity
   e. Product Control
2. Development System
   a. Capacity
   b. Suitability
   c. Usability
   d. Familiarity
   e. Reliability
   f. System Support
   g. Deliverability

C. Program Constraints
1. Resources
   a. Schedule
   b. Staff
   c. Budget
   d. Facilities
2. Contract
   a. Type of Contract
   b. Restrictions
   c. Dependencies
3. Program Interfaces
   a. Customer
   b. Associate Contractors
   c. Subcontractors

Abbildung 1-3. Auszug aus [6] Taxonomic Group Definitions

1. Requirements

a. Stability
[Are requirements changing even as the product is being produced?]

[1] Are the requirements stable?
(No) (1.a) What is the effect on the system?
- Quality
- Functionality
- Schedule
- Integration
- Design
- Testing

[2] Are the external interfaces changing?

b. Completeness
[Are requirements missing or incompletely specified?]

[3] Are there any TBDs in the specifications?

[4] Are there requirements you know should be in the specification but aren't?
(Yes) (4.a) Will you be able to get these requirements into the system?

[5] Does the customer have unwritten requirements/expectations?
(Yes) (5.a) Is there a way to capture these requirements?

[6] Are the external interfaces completely defined?

Abbildung 1-4. Auszug aus [6] Taxonomic-Based Questionnaire

Software Hazard Analysis (SHA)

Die Software Hazard Analysis, kurz SHA, ist ein Methodenpaket, das in den Softwareentwicklungsprozess eingebunden ist. Es verbindet somit an die Bedürfnisse der Software-Entwicklung angepasste Methoden zu einem ‚Ganzen'. Die SHA ist eine strukturierte, analytische Methode um Risiken zu finden und, wenn möglich, zu beseitigen oder entsprechende Vorkehrungen zu treffen.

Anwendung. Risiken können nicht alle in der Anforderungsanalyse gefunden werden, sondern auch während des Entwicklungsprozesses auftreten. Softwarequalität hängt nicht nur von der Qualität der Anforderungen ab, sondern auch von der Umsetzung dieser. Dies bedeutet, dass ein ganzheitlicher Risikomanagement-Prozess über alle Phasen der Entwicklung laufen muss, um Sicherheit und Qualität garantieren zu können. Dabei dürfen Ergebnisse aus vorigen Schritten nicht verloren gehen, damit auch am Ende des Entwicklungsprozesses erkannte Risiken noch berücksichtigt werden.

Verfahren. Die SHA Methode bietet ein weites Spektrum diverser Methoden, von denen jede einen speziellen Teil des Software-Entwicklungsprozesses und dessen Eigenheiten berücksichtigt. Auch sind die Methoden unterschiedlich aufwendig und spezialisieren sich auf besondere Risiken. Somit können die Methoden immer an das Projekt und Problem angepasst werden.

Besondere Berücksichtigung findet hier die Unterscheidung zwischen Software- und Hardwarefehlern und damit spezieller Risikoanalysen für diese.

Software macht keine Fehler; sie macht vielleicht nicht das, was sie sollte, aber nur deshalb, weil sie so programmiert wurde. Dies erschwertes, Software mit konventionellen quantitativen Methoden oder Vorhersage-Techniken zu untersuchen. Hardware hingegen ist weitaus berechenbarer, wie eine wechselnde Signalqualität. Logische Hardware-Funktionen sind vorhersagbar, was für Softwaretiming nicht gilt. In manchen Fällen lässt sich der Auftritt eines Ereignisses nicht nur nicht vorhersagen, sondern noch nicht mal wiederholen.

Zur SHA gehören, unter anderem, folgende Methoden:

Preliminary Hazard Analysis (PHA): Die PHA identifiziert und analysiert sicherheitskritische Bereiche von Beginn der Software-Entwickllung an. Die erkannten Risiken werden kontinuierlich verfolgt und dokumentiert, um angemessene Gegenmaßnahmen ergreifen zu können. Hierbei wird keine spezielle Methode vorgegeben, für dieses Verfahren könnte man demnach etwa die What-If Analysis einsetzen.

Fault Hazard Analysis applied to software (FHA): Ein grundlegendes induktives Analyseverfahren. Die Auswertung beginnt mit dem spezifischsten Modul des Systems und integriert dann die einzelnen Ergebnisse in ein Gesamtsystem. Zweck der FHA ist es, ein System systematisch zu überprüfen, um Risiken und ihre Effekte zu erkennen.

Soft-Tree (Fault Tree Analysis applied to software) (ST): Es werden vorher erkannte Risiken im Programmfluß verfolgt. Ausgehend vom Risiko wird nun ein Baum erzeugt, in dem nur solche Ereignisse berücksichtigt werden, die in letzter Konsequenz zum Risiko führen. Somit wird ein kritischer Pfad erkannt, der dann später beseitigt oder entsprechend kontrolliert werden kann.

Software Sneak Circuit Analysis (SCA): Der Zweck des SCA ist es, unbeabsichtigte Programmpfade oder Anweisungsreihenfolgen zu identifizieren, die dann unerwünschte oder unpassende (zeitlich nicht gewünschte) Ereignisse produzieren.

Software Checklists (SC): Review Technik, die anhand von Checklisten einzelne Risiken identifiziert.

Software design guidelines (DG): Richtlinien, die aus Erfahrungswerten oder durch empirische Studien gewonnen wurden. Hierzu gehören etwa Richtlinien zum Entwerfen von GUI's. Bewährte Verfahren, für die Fehlermodelle existieren, können hier ebenfalls als Richtlinien dienen.

Software coding standards (CS): Richtlinien, die aus Erfahrungswerten gewonnen wurden oder durch empirische Studien belegt wurden. Sie dienen dazu, den Code zum Beispiel besser lesbar zu gestalten, um damit die Wartbarkeit des Codes zu gewährleisten.

Petri Net Analysis (PNA): Modellierungsmethode für Systeme, die auch mathematisch formuliert werden kann. Somit sind automatisierte Analysen möglich. Das bekannteste Resultat eines Petri-Netzes ist wohl der Erreichbarkeitsgraph. Dieser Systemablaufgraph zeigt alle erreichbaren Zustände und die Transaktionen, die zu diesen Zuständen führen, bzw. von ihnen ausgelöst werden.

Da es sich hierbei um ein, in den ganzen Entwicklungsprozess, integriertes Verfahren handelt, kommen hier auch strukturorientierte Methoden wie die FHA, ST, SCA und PNA zum Einsatz.

Anforderungen. Die Anforderungen an das Analyseteam hängen in starkem Maße von den Anforderungen der eingesetzten Methoden ab. Da es nicht gerade wirtschaftlich ist, für jede Methode ein Spezialisten-Team einzusetzen, müssen sich die Teammitglieder mit mehreren Methoden auskennen. Ein Team könnte etwa aus zehn Mitgliedern bestehen, von denen jedes in einer Methode spezialisiert ist. Daraus kann dann ein Team von ca. fünf Personen für jede Methode gebildet werden. Die Anforderungen reichen von den einfachen Ingenieurs- und Prozesskenntnissen bis hin zu formalen Verifikationstechniken.

Beispiel. Sowohl die NASA als auch das US-Militär haben ihren eigenen Software Hazard Analysis Standard. So hat das Department of Defense in mehren Projekten und Revisionen den MIL STD-882B [8] festgelegt:

- SW Requirements Hazard Analysis (SRHA): Traces system-level hazards to SW requirements.
- Top-level Design Hazard Analysis (TLDHA): Architecture analysis, analysis of communication and resource sharing at the external interface of the configuration item.
- Detailed Design Hazard Analysis (DDHA): Lower level architecture analysis of communication, and resource sharing within a configuration item.
- Code-level Software Hazard Analysis (CLHA): Logic and data analysis.
- Software Safety Testing (SST): Establishes coverage of hazards identified in SRHA, using test data derived from TLDHA, DDHA, and CLHA.
- Software/User Interface Analysis: Man-machine and human factors analysis in conjunction with OSHA.
- Software Change Hazard Analysis: Management of the change process and the request for change effect on safety.

Besonderheiten. Die SHA erlaubt eine sehr individuelle Analyse. Detailgrad und Umfang der Methoden sowie Analyseschwerpunkte lassen sich durch die Wahl der Methoden setzen.

Vorteile ergeben sich im Besondern durch Integration in den gesamten Softwareentwicklungsprozess. Die Skalierbarkeit und der starke Softwarebezug unterstützen dies.

Nachteile sind jedoch die je nach Methodenwahl hohen Anforderungen an das Personal, und auch die große Menge an möglichen Methoden kann Schwierigkeiten bereiten.

3. Toolunterstützung

Wie könnte nun eine Toolunterstützung für problemorientierte Risikoanalysemethoden generell aussehen? Denkbar ist hier eine einfache automatische Dokumentation von entdeckten Risiken. Auch ein Checklisten-Verwalter, der anhand gewisser Charakteristika Checklisten vorschlägt, wäre denkbar. Lösungen für einzelne Analysemethoden stellen den Nutzer später möglicherweise vor ein Kompatibilitätsproblem. Wer möchte schon seine Anforderungsanalyse mit Tool-A, die entdeckten Risiken mit Tool-B und später die Überwachung der Risiken mit Tool-C durchführen? Also müssen solche Tools zumindest für ein kleines Umfeld eine breite Unterstützung oder Begleitung bieten. Auch wäre es wünschenswert wenn Teilprozesse automatisierbar wären. Für kreative Prozesse, was die meisten initialen problemorientierten Ansätze sind, ist dies zwar nicht möglich, bei späteren Analysen aber durchaus.

Ein Tool, das solch einen ganzheitlicheren Ansatz verfolgt, wird in 3.1 vorgestellt. Ein Tool ist im Allgemeinen nur so hilfreich wie die Bereitschaft es einzusetzen, zum anderen stellt sich aber auch die Frage des wirklichen Nutzens. Kapitel 3.2 fasst die Ergebnisse einer Studie über dieses Thema zusammen.

3.1. SpecTRM

SpecTRM = Specification Tools and Requirements Methodology ist ein Werkzeug zur Spezifikation, Entwicklung und späteren Wartung von sicherheitskritischen (Software)Systemen (siehe Abbildung 1-5). Es bietet eine Entwicklungsumgebung die sowohl Inspektionen, formale Verifikations-Tools, als auch Simulationen unterstützt. Zudem werden nicht nur Anforderungen, sondern auch Sicherheitsrichtlinien durch die ganze Dokumentation und den Entwurf verfolgt. Somit können wichtige Sicherheitsmerkmale von Anfang an in das System eingebaut werden und verursachen am Ende keinen nachträglichen Implementierungsaufwand [9].

Die Entwicklungsumgebung unterstützt dabei Risikoanalyse-Techniken wie etwa die Hazard Analysis. Für die Systemspezifikation benutzt SpecTRM eine eigene Spezifikations-Methode, die so genannte ‘intent specification’. Sie beinhaltet alle nötigen Aspekte, um ein sicherheitskritsiches System zu strukturieren. Dies beinhaltet die Verfolgbarkeit von Top-Level Anforderungen und Sicherheitsrichtlinien bis hinunter zu den Komponentenentwürfen und der Implementierung. Dabei ist der Umgang mit dieser ‘intent specification’ durch eine spezielle Sprache so vereinfacht, dass man sie ohne spezielle mathematische Kenntnisse oder aufwendiges Training nutzen kann. [10]

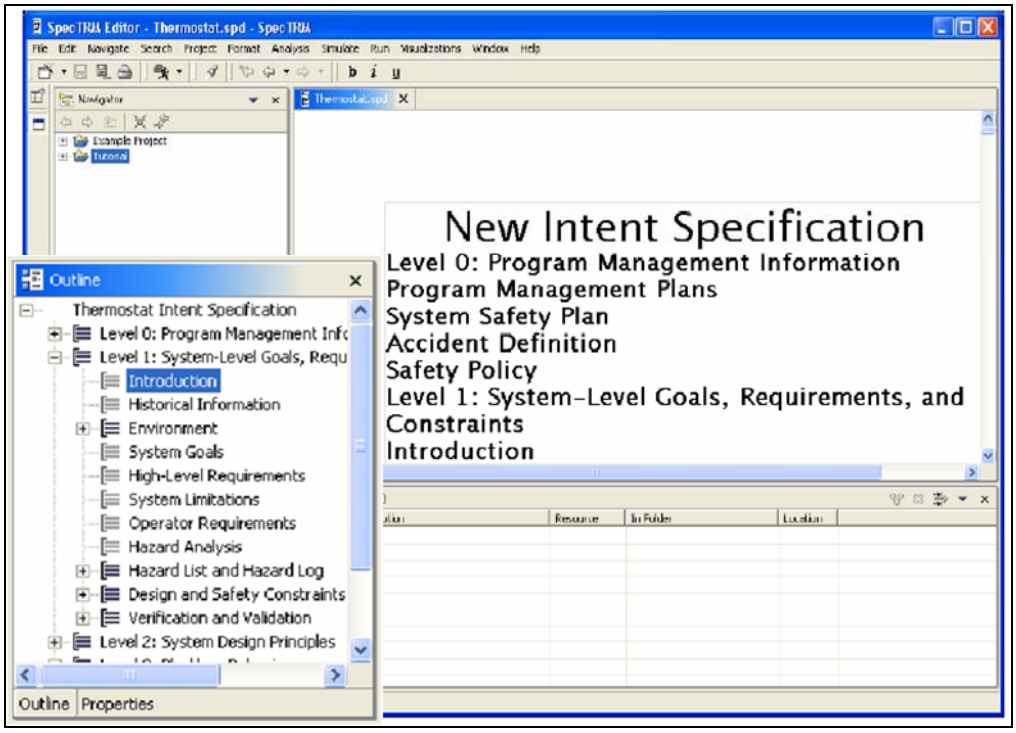


Abbildung 1-5. Oberfläche von SpecTRM und das Outline-Fenster, in dem durch die ‚intent specification' navigiert werden kann.

Abbildung 1-6 zeigt die Struktur der 'intent specification'. Jedes Level repräsentiert eine eigene 'Intent'-Tiefe, die jedoch nicht mit speziellen Software-Entwicklungsphasen verbunden. Die Informationen in jedem Level beschreiben die Umgebung, das System und die auszuführenden Aufgaben. Die Systembeschreibung kann dabei nahezu beliebig in Subkomponenten zerlegt werden (decomposition). Außer den zwei Richtungen (Intent und Decomposition) ist die 'intent specification' in der Lage, mit jedem möglichen Abstraktionsniveau ('Refinement'-Tiefe) zu arbeiten. Damit ist auch eine iterative Verfeinerung während der Entwicklung möglich, ohne dabei einmal spezifizierte Richtlinien (Anforderungen, Sicherheitsrichtlinien) zu vernachlässigen.

In SpecTRM ist es möglich mit einem beliebigen Teil der 'intent specification', d.h. einem beliebigen Level, mit beliebiger Verfeinerungs-Tiefe zu beginnen. SpecTRM wurde so entworfen, dass das Modellieren sogar für Ingenieure, die mit formalen Methoden nicht vertraut sind, einfach ist. Das Blackbox-Modell auf Level 3 ist immer ausführbar und für formale Analyse und Simulation verwendbar, einschließlich 'hard-ware-in-the-loop testing'.

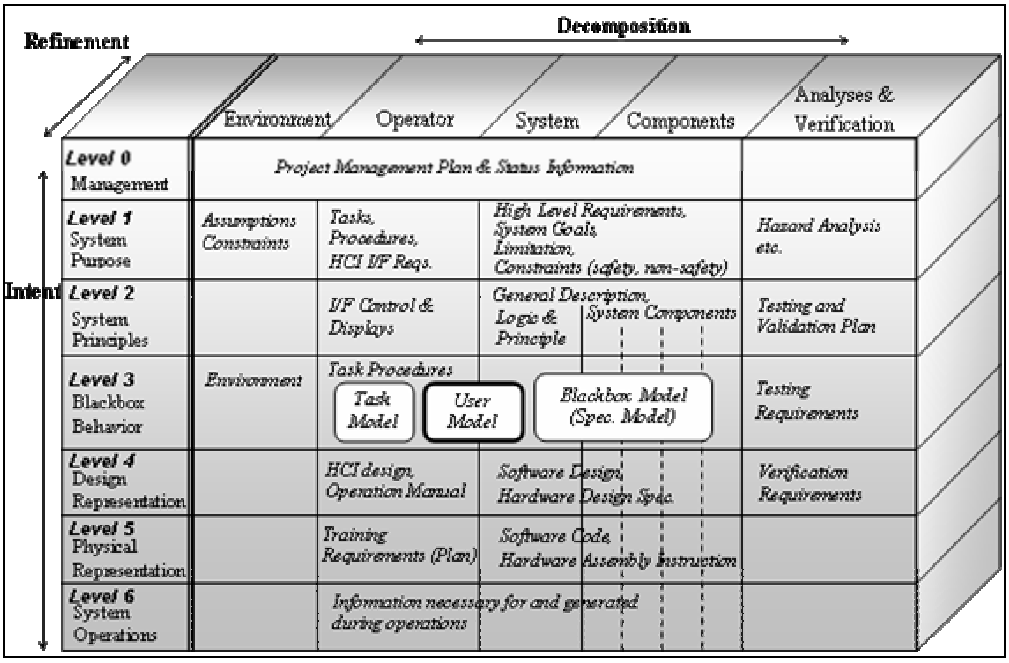


Abbildung 1-6. SpecTRM Intent Specification Structure

3.2. MIT-Studie über SpecTRM

Wie schon erwähnt, kann man über ein Tool viel Gutes sagen, aber solange es nicht im Einsatz getestet wurde, ist dies wenig aussagekräftig. Um zu belegen, dass SpecTRM auch das hält, was es verspricht, wurde durch eine Studie des MIT[2] untersucht, ob SpecTRM den NASA-STD unterstützt, bzw. konform dazu arbeitet.

Für ein aussagekräftiges Ergebnis wurde ein Modell mit realistischer Komplexität entworfen und alle wichtigen Attribute dabei identifiziert. Hierzu gehören Vorhersagbarkeit, abstraktes Modellieren, Ausführbarkeit und ein einfacher Aufbau. Im Versuch konnte man all diese Schlüsselattribute bei SpecTRM vorfinden. Zudem wurde SpecTRM mit aktuellen Sicherheitsstandards verglichen und konnte auch hier beweisen, dass es diese sehr gut unterstützt. Die Benutzung von SpecTRM erlaubte eine korrekte und einfache Analyse des Systemverhaltens. Die Studie kam zu dem Schluss, dass SpecTRM für folgende Bereiche gut geeignet ist:

- *Konzeptioneller Systementwurf*, der am Anfang der Entwicklungs-Phase stattfindet
- *Reverse-Engineering*, einschließlich ‘accident investigations’ oder ‘problem corrections’
- *Shadow process*, wie IV&V (Independent Verification and Validation)
- *Als Teil des Entwicklungs-Prozesses* selbst

Der Entwicklungsprozess, den SpecTRM vorschlägt, ist im Bezug auf Verbesserung des Software-Entwicklungsprozesses eines sicherheitskritischen Systems viel ver-

sprechend. Er erlaubt Ingenieuren oder Managern, das formale Verhalten des Systems zu verstehen und es mit leistungsfähigen Werkzeugen vom Anfang des Ent-

[2] MIT – Massachusetts Institute of Technology.

wicklungsprozesses an zu analysieren. Neue Arten von Fehlern können ebenfalls mit der Modell-Analyse gefunden werden.

Die Studie hat auch Möglichkeiten entdeckt, um redundante Architektur zu vermeiden. Heute entscheiden sich Designer häufig für eine redundante Computer- und Software-Architektur (z.B. Backup Controller, N-Version Software, etc.) als Entwurfslösung für sicherheitskritische Systeme. Das redundante Computersystem erhöht nicht nur Kosten und Aufwand, sondern erhöht auch die Komplexität des Systems und beinhaltet neue Fehlerquellen. Ein Grund, warum die redundante Architekturlösung gewählt wird, ist die Schwierigkeit exaktes Software-Verhalten für alle möglichen Fälle vorherzusagen oder zu analysieren. Indem man nun SpecTRM nutzte, konnte man sich auf die Analyse des Systemverhaltens konzentrieren und unerwünschtes Verhalten ohne Redundanz beseitigen.

Somit kam die Studie zu dem Schluss, dass SpecTRM sehr gut für Raumfahrtanwendungen und den NASA-STD geeignet ist [11]. Tabelle 1-5 zeigt die Ergebnisse der Gegenüberstellung von SpecTRM und dem NASA-STD.

NASA Software Safety Standard (NASA-STD-8719.13A)	Intent Specification / SpecTRM tool
System safety analysis Preliminary Hazard Analysis, software safety analysis to define safety-critical software (potential cause, or supporting the control of a hazard) and software safety requirements through the project life cycle.	In level 1 of the intent specification, the hazard analysis data can be described including software fault tree analysis. The software safety requirements can be described in level 1 as safety constraints.
Software safety The objective is to ensure that safety is considered throughout the software life cycle.	All life cycle activities can be treated as the appropriate level of intent, refinement, or decomposition.
Software safety tasks by life cycle phase	
Software safety planning Planning shall be documented in Software Management plan or Safety Management Plan Software Requirements Specification Developments Development of software safety requirements Analysis of software requirement for potential hazards. Test planning is begun.	The planning can be included or referenced in the "Program Management Plan" or "System Safety Plan" parts of the intent specification. The software safety requirements can be described in level 1 as safety constraints. Test planning can be described in level 5. Formal Analysis of requirements can be performed in level 3 with a blackbox behavior.
Software architectural design This design process shall include identification of safety design features and methods	The level 2, "System design principle", mainly shows such design features and methods.
Software detailed design Low-level design for the software units, safety-related information shall be into all user manuals. Development of test procedure	The level 3, " Physical and Logic Design Representations", shows detailed design. The safety information for "User manual" can be described in level 5 as well as test procedures.
Software implementation Code, which shall implement the safety feature and methods developed during the design process	The code is described in level 5. Unit testing information can be stored at V&V section in level 5
Software integration and acceptance testing Testing to verify of software safety requirements including hazard verification, Acceptance testing verify in conjunction with system hardware and operators.	The testing information can be described in level 5.
Software operations and maintenance When changes are made, performing hazard analysis, updating software safety requirements and specification, design, and operator document, regression testing	The operation information is described in level 6. When changes are made in the intent specification, all levels of information can be modified. The tools set assists us to identify the parts affected by the changes, using the traceability functions.
Phase independent tasks	
Safety requirements traceability To trace the flow down of the software safety requirements to design, implementation, and test	Using tags or linkage, we can define all requirements traceability down to any level and any words in the specification.
Discrepancy reporting and tracking Closed-loop tracking of safety-related discrepancies, problems, and failures.	There is currently no function for tracking itself, but we can indicate which parts have open issues and the status. Discrepancy reporting and tracking is planned for future versions of the SpecTRM tools.
Software change control All changes, modifications, and patches made to requirements, design, code, systems, test plans, procedures, or criteria shall be evaluated to determine the effect of the change on system/subsystem safety.	All changes made to any part of the intent specification can be analyzed for their effect using the traceability information.
Safety program reviews Supporting System safety review	The intent specification provides safety information to support safety review, because all descriptions in the intent specification are written in natural language English, or readable SpecTRM-RL.
Software safety analysis	
Software safety requirements analysis To identify software requirements that are safety-critical, and ensure the correctness and completeness	In addition to such manual analyses, the tool provides or will provide several tool analyses using formal models in level 3. Testing scenarios & results simulation & analysis Completeness and consistency analysis Software deviation analysis Mode confusion analysis
Software safety architectural design analysis To ensure the correctness and completeness, and test coverage	
Software safety detailed design analysis To ensure the correctness and completeness and ensure test coverage, safety-related information fro User's Guide	
Code safety analysis To ensure the correctness and completeness, identify potentially unsafe states, and ensure test coverage	
Software test safety analysis Test result shall be analyzed**.**	
Software change analysis To evaluate whether the change could invoke a hazardous state, affect a hazard control, increase the likelihood of hazardous state etc.	

Tabelle 1-5. Applicability table of SpecTRM to NASA software safety standard [13]

4. Zusammenfassung

Je früher man sich über Risiken bewusst wird, desto früher kann man ihnen angemessen begegnen und somit ihre Folgen abschwächen, verhindern und kontrollieren. Dies senkt nicht nur die Kosten (frühe Änderungen sind billiger als späte), sondern steigert letztlich auch die Qualität (was man kennt, kann einen nicht mehr überraschen). Der früheste Zeitpunkt um nach Risiken zu suchen ist in der initialen Problemanalyse, daher bei der Erstellung der Problembeschreibung (siehe V-Modell, Abbildung 1-1) und den Anforderungen.

Hierfür bieten sich vor allem zahlreiche problemorientierte Ansätze wie etwa die What-If Analysis an. Diese reichen von einfachem Brainstorming bis zu integrierten Analysepaketen mit formalen Verifikationen. Die meisten dieser Methoden kommen aus den Bereichen der Chemie, der Kraftwerke, dem Militär und der Luft- und Raumfahrt. Einige lassen sich sehr einfach auf softwarespezifische Umgebungen anpassen, andere geben nur wenig Hilfe für den Einsatz in der Softwareentwicklung oder sind hierfür gänzlich unbrauchbar.

Die hier vorgestellten Methoden sollten jedem eine Einstiegsmöglichkeit bieten. Mit einem Brainstorming, wie es die What-If Analysis bietet, sollte jeder zurechtkommen, um damit Erfahrungen zu sammeln. Mit zunehmender Erfahrung fällt der Schritt zum Erstellen oder Nutzen von Checklisten und damit auch Checklisten-Analysemethoden (wie What-If/Checklist Analysis) nicht schwer. Ob man nun zu weiteren Methoden oder gar einem Paket wie der Software Hazard Analysis greift hängt von den gewünschten Ergebnissen, den eigenen Möglichkeiten und dem Problem ab, um das es geht. Für manche mag es einfacher sein sich mit Tools zu beschäftigen um Risiken zu identifizieren, zu analysieren und letztlich zu kontrollieren.

Welche Analysemethode man nun einsetzen will, bleibt jedem selbst überlassen. Letztlich bleibt es doch eine Frage des Kosten-Nutzen Verhältnisses, der Ressourcen und des Geldes. Wobei man eines nicht vergessen darf, früher oder später wird Tom Gilb recht behalten:

> *„If you don't actively attack the risks, the risk will actively attack you."* [1:343]

Referenzen

[1] Ernest Wallmüller, Software-Qualitätsmanagement in der Praxis. München: Carl Hansenverlag, 2.Auflage 2001.

[2] Nancy G. Leveson, (2003), Accident Causes [Online]. Available: www.safeware-eng.com/index.php/white-papers/.

[3] Federal Aviation Administration, System Safety Handbook: Practices and Guidelines for Conducting System Safety Engineering and Management, 30 Dezember 2000.

[4] MIL-STD-882C, "SYSTEM SAFETY PROGRAM REQUIREMENTS", Department of Defense, Washington DC, Januar 1993.

[5] C. Amlinger, D. Czeremin, H. Minges, Proseminar "Analyse von Software-Unfällen: Software-Unfälle an den Beispielen Airbus 320 und X-31". AGSE, TU Kaiserslautern, Kaiserslautern 1998.

[6] M. Carr, S. Kondra, I. Monarch, F. Ulrich,C. Walke, "Taxonomy-Based Risk Identification", Software Engineering Institute, Carnegie Mellon, tech. Rep. CMU/SEI-93-TR-006, Juni 1993.

[7] Institut für Softwaretechnologie. (2004, November. 25). [Online]. Checklisten für den Software-Entwicklungsprozess. Available: www.iste.uni-stuttgart.de/se/links/checklists/.

[8] MIL-STD-882B, "SYSTEM SAFETY PROGRAM REQUIREMENTS", Department of Defense, Washington DC, AMSC Number F3329 FSC SAFT, März, 1984.

[9] SpecTRM, (2003), Accident Causes [Online]. Available: www.safeware-eng.com/index.php/products.

[10] N. Leveson, M. Heimdahl, and J. Damon Reese, “Designing Specification Languages for Process Control Systems: Lessons Learned and Steps to the Future”, presented at SIGSOFT FOSE '99 (Foundations of Software Engineering), September 1999.
[11] N. Leveson, M. Katahira, „Use of SpecTRM in space Applications“, presented at the 19th International System Safety Conference, Huntsville, Alabama, September 2001.
[12] Stephans, A. Richard, Talso, Werner, "System Safety Analysis Handbook", New Mexico Chapter: System Safety Society, 2nd Edition, 1997.
[13] NASA-STD-89719.13A, “Software Safety Standard, NASA, September 1997

Kapitel 2. Strukturorientierte Vorgehensweisen im Risikomanagement sicherheitskritischer Softwaresysteme

Mathias Int-Veen

Universität Kaiserslautern
m_intveen@web.de

Kurzfassung. Technische Systeme werden zunehmend in sicherheitskritischen Bereichen eingesetzt. Die Systeme müssen kontinuierlich höheren Sicherheitsansprüchen genügen. Dieser Beitrag beschäftigt sich mit den strukturorientierten Ansätzen, mit denen die Risikopotentiale, die bei der Nutzung der Systeme bestehen, gesenkt werden können. Er gibt einen kurzen Überblick über die grundlegenden Merkmale und Vertreter strukturorientierter Ansätze. Anschließend daran wird die Failure Modes and Effects Analysis als beispielhafter Vertreter strukturorientierter Ansätze genauer untersucht und erläutert, und die Übertragung dieser Methode auf Softwaresysteme analysiert. Abschließend werden Vor- und Nachteile der (Software) Failure Modes and Effects Analysis sowie strukturorientierter Ansätze im Allgemeinen herausgearbeitet und bewertet.

1. Einführung

Technische Systeme werden heutzutage für immer komplexere Aufgaben eingesetzt. Die Komplexität der Systeme wächst dabei mit. Diese Systeme werden auch in Gebieten eingesetzt, in denen, bei einem Versagen des Systems, ein großer wirtschaftlicher und materieller Schaden zu erwarten ist, enorme Umweltschäden zu erwarten sind oder Menschenleben gefährdet sind. Daher wird es immer wichtiger, diese Systeme so anzulegen, dass sie stabil und fehlerfrei respektive fehlertolerant laufen. Das Risikomanagement sicherheitskritischer Systeme wird folglich kontinuierlich wichtiger.

Man kann die verschiedenen Ansätze, die im Rahmen des Risikomanagements vorgenommen werden können, folgendermaßen klassifizieren:

- Strukturorientierte Maßnahmen zerlegen das System, und folgern für die Einzelteile welche Risiken auftreten können
- Domänenorientierte Ansätze beschreiten den umgekehrten Weg. Ausgehend von einem angenommen Fehler oder Risiko wird das System untersucht, und Gegenmaßnahmen im System verankert
- Humanorientierte Ansätze untersuchen die Interaktion zwischen Mensch und Maschine, und versuchen gegen die Risiken anzugehen, die in diesem Spannungsfeld auftreten.

Dieser Beitrag beschäftigt sich mit den strukturorientierten Ansätzen und insbesondere der Failure Modes and Effects Analysis als Vertreter dieser Ansatzgruppe.

1.1. Merkmale

Strukturorientierte Ansätze stellen das System und seine Struktur in den Mittelpunkt der Betrachtung. Der Ablauf einer Untersuchung des Systems verläuft dabei prinzipiell in folgenden Schritten:

1. Ausgangspunkt ist das bis zum Betrachtungszeitpunkt fertig gestellte System. Je nach Betrachtungszeitpunkt sind verschiedene Ansätze besser geeignet.
2. Davon ausgehend wird das System strukturiert. Es werden die Module des Systems festgestellt. Ein Modul ist dabei eine Systemkomponente, die eine bestimmte Aufgabe zu erfüllen hat.
3. Die Module werden dahingehend analysiert, welche Fehler im betrachteten Modul auftreten können.
4. Die Auswirkungen der Fehler werden untersucht. Es werden die Effekte auf das Gesamtsystem, auf übergeordnete Module und auf Module, die mit dem Modul in Wechselwirkung stehen, untersucht.
5. Für die erkanten Probleme werden Gegenmaßnahmen erdacht und untersucht, und in das System integriert.
6. Das veränderte System ist die Grundlage für den weiteren Entwicklungsprozess.

Es findet also eine Vorwärtsbetrachtung anhand von Ereignisketten innerhalb des Systems statt. Risikopotentiale durch die Domäne oder den Faktor Mensch werden nicht beachtet. Vorraussetzung für eine erfolgreiche derartige Betrachtung ist eine Dokumentation des Systems von hoher Qualität. Andernfalls können Fehlerquellen aufgrund mangelnder Spezifikation des Systems nicht gefunden werden. Dies kann zum Beispiel daran liegen, dass die Funktionen von Modulen nicht richtig spezifiziert sind oder die Auswirkungen nicht vollständig verfolgt werden können, wenn Zusammenhänge zwischen Modulen nicht eindeutig sind.

1.2. Vertreter strukturorientierter Ansätze

Es gibt viele Vertreter strukturorientierter Ansätze. Oft ist vor der Durchführung der Methoden eine weitere Formalisierung und Überarbeitung der erstellten System-Dokumentationen nötig. Die Methoden basieren zum Beispiel auf boolescher Logik oder Petri Netzen. Daher müssen die die Untersuchung durchführenden Personen teilweise einschlägige Erfahrung mitbringen. Beispielhaft sollen hier einige verbreitete Ansätze kurz erläutert werden.

Die Failure Modes and Effects Analysis (FMEA) „was developed by reliability engineers to permit them to predict equipment reliabilities“[7]. Die FMEA untersucht dabei die Module eines Systems auf mögliche Fehlerzustände, und ermittelt die Auswirkungen auf das Gesamtsystem. Die Fehlerzustände werden nach Auftrittswahrscheinlichkeit, Bedeutung und Entdeckungswahrscheinlichkeit bewertet. Die FMEA wird in Kapitel zwei als Beispiel für die Anwendung auf Hardwaresysteme und zur Verdeutlichung strukturorientierter Ansätze noch genauer untersucht.

Die Energy Analysis untersucht Energieflüsse innerhalb eines Systems. Die Energieflüsse können zum Beispiel chemischer, elektrischer oder mechanischer Art sein. Ausgehend von den Quellen der Energie werden alle Barrieren dahingehend untersucht, ob sie unerwünschte Auswirkungen auf Komponenten des Rest-Systems adäquat verhindern. Sie verfolgt somit die Auswirkungen der Flüsse auf das Gesamtsystem. Sie kann auf alle energieenthaltenden Systeme, wie zum Beispiel Kraftwerke, angewandt werden. Ihre Anwendung ist auf eine schnelle und wenig tiefgehende

Untersuchung beschränkt, und ihre Ergebnisse müssen durch weitere Analysen gestützt werden.

Die Network Logic Analysis fußt auf einer Systemrepräsentation durch boolesche Werte. Sie beschreibt eine Systemoperation durch ein Netzwerk logischer Werte. Durch die Veränderung dieser Werte können Ereignisse im System dargestellt werden. Ziel ist das Auffinden von Systemmodulen, die für Fehler anfällig sind.

Die Analyse durch Petri Netze basiert auf Erreichbarkeitsgraphen. Durch Petri Netze können Systeme auf mehreren Abstraktionslevels modelliert werden. Beginnend in einem Startzustand in einer Modellierung werden alle Folgezustände weitergeführt, bis alle Zustände des Systems identifiziert sind. Resultat ist der Erreichbarkeitsgraph. An ihm lassen sich sichere und unsichere Zustände des Systems erkennen. Nachteilig an Petri Netzen ist der hohe Grad an Vorwissen aus Disziplinen wie Mathematik und Informatik der benötigt wird, um die Analyse durchzuführen zu können.

1.3. Aufbau des Beitrags

In Kapitel 2 wird die FMEA als beispielhafter Vertreter strukturorientierter Maßnahmen erläutert. Die FMEA wird bei der Untersuchung von Hardwaresystemen eingesetzt. Zuerst werden grundlegende Merkmale, wie der Einsatzzeitpunkt im Projektablauf, dargestellt. Anschließend wird die prinzipielle Durchführung erklärt, und die FMEA bewertet. Im Anschluss daran wird die Failure Modes, Effects and Criticality Analysis (FMECA) als Erweiterung der FMEA erläutert. Die FMECA dient der (im Vergleich zur herkömmlichen FMEA) genaueren Bewertung der entdeckten Risiken. In Kapitel 3 wird dann die Software-FMEA erläutert. Ziel der Software-FMEA ist die Übertragung des FMEA – Ansatzes von Hard- auf Softwaresysteme. Die Durchführung der Software-FMEA wird erläutert und die Unterschiede zur FMEA werden herausgearbeitet. In Kapitel 3.3 wird dann eine Bewertung der Software-FMEA vorgenommen. Kapitel 4 enthält eine abschließende Zusammenfassung des Beitrages.

2. Die Anwendung bei Hardwaresystemen - Failure Modes and Effects Analysis

Anfang der sechziger Jahre des vergangenen Jahrhunderts wurde die FMEA zur Qualitätssicherung im NASA Apollo-Projekt eingesetzt. Ende der siebziger Jahre wurde sie dann in den Automobilbereich eingeführt, und 1980 als Ausfalleffektanalyse in die DIN 25448 aufgenommen [1][2]. Die FMEA ist anwendbar für alle Arten von Systemen, Subsystemen etc[3].

Bei der FMEA werden alle Module eines Systems dahingehend analysiert, welche Fehlerzustände sie generieren können. Man unterscheidet drei Arten der FMEA [1], die

1. System-FMEA,
2. die Konstruktions-FMEA,
3. und die Prozess-FMEA.

Die System-FMEA betrachtet das Gesamtsystem und das Zusammenwirken der Systemmodule. Die Konstruktions-FMEA betrachtet die Merkmalsebene zum Beispiel der Baugruppen und Bauteile. „Ziel ist [es] die konstruktive Auslegung der Merkmale sicherzustellen“[1]. Bei der Prozess-FMEA werden die Prozessabläufe in Schritte

gegliedert.[3] Diese Prozessschritte werden dahingehend untersucht, ob die Merkmale der Bauteile eingehalten werden. Im Folgenden wird diese Unterscheidung nicht aufrechterhalten, da die prinzipielle Vorgehensweise für alle drei Arten gleich ist.
Die FMEA ist eine „bottom-up" Methode. Diese Kategorie von Methoden geht vom „Ausfall einer kleinen Komponente aus und untersucht, welche übergeordneten Module dadurch getroffen werden."[6]

Die FMEA kann sowohl während der Entwicklung als auch, zur kontinuierlichen Verbesserung, auf ein fertiges Produkt eingesetzt werden. Sie ist also für zwei Vorgehensweisen geeignet, den präventiven und den korrigierenden. Der präventive Ansatz hat das Ziel „mögliche Schwachstellen [im System] zu finden, deren Bedeutung zu erkennen, zu bewerten und geeignete Maßnahmen zu ihrer Vermeidung bzw. Entdeckung rechtzeitig einzuleiten"[1]. Sie sollte „bereits in einem sehr frühen Stadium des Produktentstehungsprozesses (z.B. in der Lastenheftphase) eingesetzt werden und untersucht den zu diesem Zeitpunkt gültigen Entwicklungs- oder Planungsstand auf mögliche Fehler" [2]. Der korrigierende zielt darauf ab, „bestehende Produkte oder Prozesse zu verbessern" [1].

Auf die Unternehmensziele bezogen, soll die FMEA also die Erreichung der folgenden Punkte unterstützen [2]:

- Steigerung der Sicherheit und Zuverlässigkeit von Produkten, und Senkung von Garantie- /Kulanzkosten durch frühzeitige Entdeckung von Fehlern
- Verkürzung der Entwicklungsprozesse, störungsarme Serienanläufe und Erhöhung der Termintreue durch Vermeidung von zeitlich später Entdeckung von früh in der Entwicklung begangenen Fehlern
- Wirtschaftlichere Dienstleistung durch Kosteneinsparung bei Korrekturmaßnahmen
- Bessere Dienstleistungen und Verbesserung der innerbetrieblichen Kommunikation

2.1. Durchführung der FMEA

Sowohl beim präventiven als auch beim korrigierenden Einsatz ist das Vorgehen gleich. Jedoch unterscheiden sich die Literaturquellen teilweise bei der Einteilung der Schritte, der Anzahl der vorzunehmenden Schritte, den zu erstellenden Einzeldokumenten etc. (Vergleiche [1][2][3] [7]). Der prinzipielle Ablauf ist folgender (Vgl. Abbildung 1-1):

1. Ausgangspunkt ist die bis zu diesem Zeitpunkt erstellte Dokumentation, also der gültige Entwicklungs- oder Planungsstand.
2. Identifikation der Module im System. Module sind dadurch gekennzeichnet, dass sie eine bestimmte Funktion erfüllen, sie sind also Funktionsträger. Jedes Modul kann nur einmal existieren. Die Anordnung erfolgt hierarchisch, wobei die Anzahl der Ebenen nicht festgelegt ist [2].
3. Ergebnis des zweiten Schrittes ist eine dokumentierte Systemstruktur.
4. Für jedes Modul in der Systemstruktur werden die möglichen Fehlerzustände bei der Funktionserfüllung identifiziert, also die Zustände, die nicht dem spezifizierten Verhalten entsprechen. Als nächstes werden die Fehlerursachen gesucht. Diese können grundsätzlich aus drei Richtungen kommen:

    - Fehler des Moduls selber

[3] Ein Prozess beschreibt die „Gesamtheit von in Wechselbeziehungen stehenden Abläufen, Vorgängen und Tätigkeiten durch welche Werkstoffe, Energien und Informationen transportiert oder umgeformt werden."[1]

- Fehler untergeordneter Module
- Fehler die über Schnittstellen zugeordnet werden können. Schnittstellen entstehen in Hardwaresystemen an Stellen an denen sich Module physikalisch treffen (beispielsweise Schraube / Mutter oder Lauffläche / Wellendichtring / Einpresssitz) [2]

5. Ergebnis des vierten Schrittes sind die dokumentierten Fehlermodi.

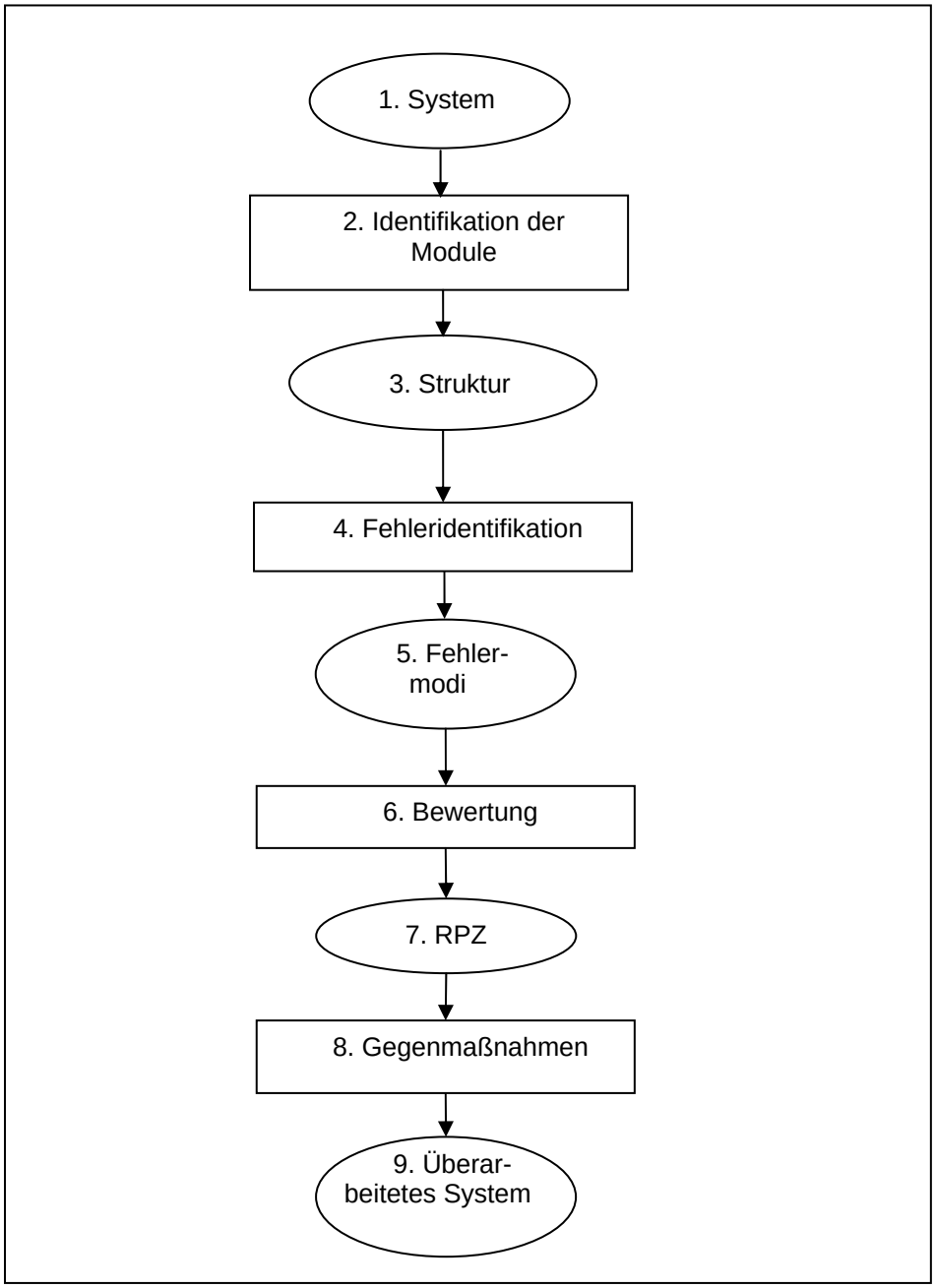


Abbildung 2-1. Ablauf der FMEA

6. Die Fehlermodi werden im Hinblick auf Bedeutung (B), Auftrittswahrscheinlichkeit (A) und Entdeckungswahrscheinlichkeit (E) bewertet. Die Bewertungen werden als Zahl zwischen eins und zehn vergeben.

- Die Bedeutung ergibt sich aus der Bewertung der Bedeutung für den Kunden. Eine zehn würde also zum Beispiel bei hohen Sicherheitsrisiken vergeben, oder wenn gesetzliche Vorschriften verletzt werden. Eine eins entsprechend bei geringer Bedeutung.
- A und E werden durch die Vermeidungs- und Entdeckungsmaßnahmen beeinflusst. Je mehr dieser Maßnahmen kombiniert angewandt werden, desto geringer sind die abhängigen Auftritts- und Entdeckungswahrscheinlichkeiten[2]
- Eine hohe Auftrittswahrscheinlichkeit erhöht die Bewertung für A.
- Eine zehn würde für die Entdeckungswahrscheinlichkeit vergeben, wenn keine Entdeckungsmaßnahmen vorgenommen würden. Eine eins würde beispielsweise vergeben, wenn sicher wäre, dass der Fehler definitiv durch Erprobungsmaßnahmen gefunden würde.

7. Ergebnis des sechsten Schrittes ist die Risikoprioritätskennzahl (RPZ), gebildet aus:

$$B * E * A = RPZ .$$

8. Definition von Gegenmaßnahmen und Integration in das System. Der Verband der Automobilindustrie e.V. schlägt folgende, nach Prioritäten geordnete Vorgehensweise vor:

   - „Konzeptänderung, um die Fehlerursache auszuschließen bzw. eine Fehlerfolge mit einer geringen Bedeutung zu erhalten
   - Erhöhung der Konzeptzuverlässigkeit, um das Auffinden der Fehlerursache zu minimieren
   - Wirksamere Entdeckung der Fehlerursachen (zusätzliches Prüfen möglichst vermeiden)“ [2]

9. Ergebnis des achten Schrittes ist eine überarbeitete Systemdokumentation.

Das Ergebnis der FMEA, die überarbeitete und um Fehler bereinigte Systemdokumentation, ist die Grundlage für die weiteren Entwicklungsschritte.

2.2. Bewertung der FMEA

Bei der Auswahl eines Ansatzes ist der trade-off zwischen dem zeitlichen und finanziellen Aufwand der Durchführung, und den möglichen Kosteneinsparungen die das Entdecken und Beheben der Fehler bringt, zu betrachten. Der Wert der entdeckten und behebbaren Fehler ist abhängig von der Qualität der Ergebnisse. Die Bewertung der FMEA orientiert sich daher an diesen zwei übergeordneten Punkten:

- Zeitlicher und finanzieller Aufwand der Durchführung
- Qualität der Ergebnisse

Je größer der zeitliche und monetäre Aufwand bei der Durchführung eines Ansatzes ist, desto eher werden sich die Verantwortlichen möglicherweise für einen „billigeren“ aber eventuell weniger gut geeigneten Ansatz entscheiden, oder die Mittel für die Durchführung begrenzen. Außerdem gilt, dass je mehr Spezialwissen oder Einarbeitungszeit für die Durchführung eines Ansatzes benötigt wird, desto teurer und fehleranfälliger wird er. Entweder entstehen Kosten durch zusätzliche Ausbildungszeiten der Teilnehmer, oder es wird die Gefahr in Kauf genommen, dass schlecht ausgebildete Kräfte aufgrund fehlender Fachkenntnisse die Qualität der Systemevaluation senken.

Die Qualität eines Ansatzes zeigt sich insbesondere darin, wie viele Fehler und Fehlerquellen er erkennt. Ein Ansatz der auf eine bestimmte Systemperspektive beschränkt ist, oder der nur wenige aller möglichen Fehler findet, ist demnach weniger gut geeignet für eine Betrachtung eines komplexen Systems. Auch die Aufdeckung von Fehlerzusammenhängen (multiple Fehler) ist hier von Bedeutung. Wenn beispielsweise zwei einzelne Fehler unabhängig voneinander nur einen geringen Effekt auf das Gesamtsystem zeigen, bei gemeinsamem Auftreten der Effekt aber unverhältnismäßig verstärkt wird, so muss der verfolgte Ansatz dies herausfinden.

Der Aufwand um eine FMEA durchzuführen kann enorm sein. Für jemanden der nicht direkt in das Projekt involviert ist, kann die Einarbeitung und die richtige Umsetzung des Systems bis zu mehreren Wochen dauern [3]. Auch [6] weißt daraufhin, dass „bottom-up“ Methoden einen sehr hohen Untersuchungsaufwand bedeuten. Die FMEA sollte von einem interdisziplinären Team durchgeführt werden um verschiedene Perspektiven zu gewährleisten[3].

Im Weiteren ist die Vollständigkeit der Ergebnisse zu betrachten, das heißt inwiefern alle Fehler gefunden werden. Die Vollständigkeit der Ergebnisse ist direkt abhängig vom Aufwand der betrieben wird um Fehlerzustände zu entdecken, Fehlerpropagierungswege zu verfolgen und Zusammenhänge zwischen Fehler zu finden. Die FMEA zeigt insbesondere bei Fehlerzusammenhängen Schwächen, weil jeder Fehlerzustand grundsätzlich unabhängig ist.[6]

Die Effekte von gleichzeitig auftretenden Fehlern (multiple Fehler) werden nicht explizit betrachtet. Daher können unvorhergesehene Fehlereffekte auftreten, wenn zum Beispiel Fehler mit geringer RPZ nicht vermieden werden, und dadurch weitere Fehler mit einer unverhältnismäßig stärkeren Wirkung erscheinen [3][7].

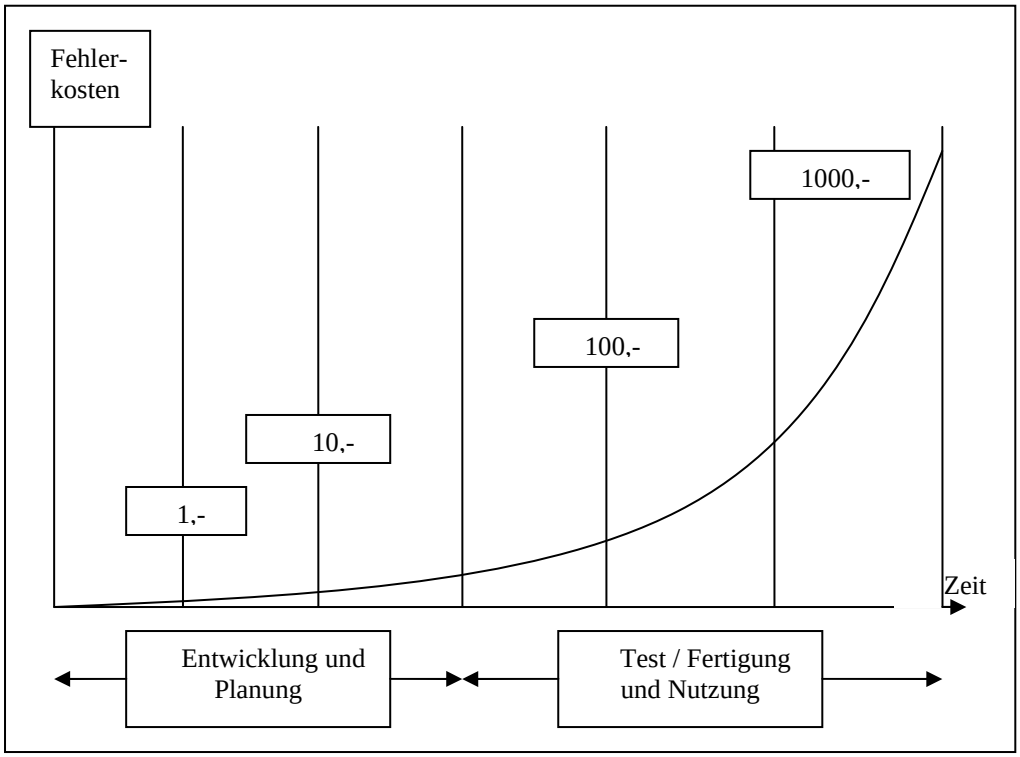


Abbildung 2-2: Entwicklung der Fehlerbehebungskosten in Abhängigkeit von der Zeit (in Anlehnung an [1])

Vorteilhaft ist, dass der Ansatz leicht verständlich ist, auch für komplexe Systeme geeignet ist (gerade durch die Herangehensweise der Subkomponenten-Bildung), und dass (fast) alle Fehler im System gefunden werden können.[3] Er ist weit verbrei-

tet und anerkannt.[8] Zudem führt die Deutsche Gesellschaft für Qualität e.V. aus, dass fast 85 % aller Fehler in der Entwicklungs- und Planungsphase entstehen und somit auch gefunden werden können. Daher können durch die frühzeitige Einsatzmöglichkeit die Entwürfe verbessert werden, und enorme Kosteneinsparungen realisiert werden[1][8] (Vgl. Abbildung 2-2.)

2.3. FMECA – Failure Modes, Effects and Criticality Analysis

Die Failure Modes, Effects and Criticality Analysis ist eine Erweiterung der FMEA. Ziel ist es, das Gefahrenpotential von Fehlermodi zu bewerten und die Wahrscheinlichkeit des Auftretens genauer zu verifizieren. Es wird eine bewertete Rangordnung der Fehlermodi zur FMEA hinzugefügt.

Die Erweiterung entspricht der Durchführung einer Criticality Analysis. Die Criticality Analysis kann auch für andere Methoden eingesetzt werden, die Fehler eines Systems bewerten. Man betrachtet dabei die Ausfallwahrscheinlichkeit und die Anzahl der Fehler über eine Zeitspanne. Es werden zwei Ansätze unterschieden, der

- qualitative Ansatz
- und der quantitative Ansatz.

Der qualitative Ansatz wird genutzt wenn keine genauen Fehlerraten für eine Systemkomponente vorhanden sind. Es werden qualitative Werte für die Komponenten angenommen, und in die Bewertung der Fehlermodi einbezogen.

Der quantitative Ansatz wird genutzt, wenn Daten bekannt sind. Diese Daten können zum Beispiel aus militärischen Handbüchern [4] stammen, in denen Vergleichswerte angegeben sind. Dabei werden die Werte in die Analyse der Fehlermodi miteinbezogen, und verbessern so die Einschätzbarkeit der Auswirkungen und der Eintrittswahrscheinlichkeiten. [7] verschiebt die Verwendung generischer Erfahrungswerte nicht von der FMEA in die FMECA, sondern benutzt sie schon in der FMEA, um die Bewertungen der Fehlermodi genauer zu verifizieren.

Die Vollständigkeit des Ansatzes ist insofern als komplett anzusehen, als das er vollkommen auf die Ausgangswerte einer anderen Analyse aufbaut. Für jedes fehlerhafte Element wird das Potential bewertet.[3]

3. Anwendung bei Softwaresystemen – Software-FMEA

Die Software Failure Modes and Effects Analysis (SFMEA) ist eine Übertragung der FMEA auf Softwaresysteme. Sie bewertet folglich das Gefahrenpotential, das durch das Versagen von Softwaremodulen entsteht. Die SFMEA ist ein Werkzeug zur Qualitätssicherung von Softwaresystemen, welche eine der „wichtigsten Aufgaben bei der Entwicklung von komplexer, umfangreicher Software und hybriden […] Systemen [5] und Anlagen „ [3] ist.

[4] MIL – HDBK -217 ist zum Beispiel geeignet für elektronische Teile;
andere Quellen sind z.B. :
IEEE STD 500, IEEE Guide to the Collection of Electrical, Electronic, Sensing Component, and Mechanical Equipment Reliability Data for Nuclear - Power Generating Station (1983)
oder International Atomic Energy Agency, IAEA – TECDOC – 478, Component Reliability Data for Use in Probalistic Safety Assesment (1988) [3]

[5] Hybride Systeme sind aus Hard- und Softwareanteilen zusammengesetzte Systeme.

3.1. Durchführung der Software-FMEA

Die SFMEA kann sowohl für reine Softwaresysteme, als auch für kombinierte Systeme, bestehend aus Hard- und Software, eingesetzt werden. Die Software-FMEA ist damit eine „konsequente Fortführung der System-FMEA zur Analyse der softwareintensiven Komponenten eines betrachteten Systems. Ihre Ergebnisse fließen in die System-FMEA zurück.“[5]

Durch ihre Konzeption, vom Ausfall eines Moduls auszugehen, und daraufhin die Auswirkungen für übergeordnete Module zu erkennen, gehört auch die Software-FMEA in die Kategorie der bottom-up Methoden [6].

Durchführungszeitpunkt für die SFMEA sind die frühen Phasen der Entwicklung, insbesondere die Phasen *vor* der Implementierung des Systems. Im Softwareentwicklungsprozess sollte die SFMEA in den Anforderungs- und Designphasen durchgeführt werden, Pseudocode wird aber als hilfreich erachtet. Die frühzeitige Durchführung kann zu Kosteneinsparungen führen, und „bedeutet Risikomanagement statt Krisenmanagement“[5]. Grundlage für dieses Vorgehen ist jedoch eine gut strukturierte Softwarearchitektur, anhand der die Zusammenhänge im System eindeutig zu erkennen sind. .

Die prinzipielle Vorgehensweise entspricht der der FMEA. Sie wird durchgeführt von einem interdisziplinären Team. Die Vorgehensweise lässt sich in fünf Punkten zusammenfassen [5][9]:

1. Die Softwarearchitektur wird in Module zerlegt. Es entsteht ein Strukturbaum, der „Besteht aus“ Beziehungen zwischen den Modulen beschreibt.
2. Die Aufgabe der Module wird beschrieben. Dadurch entsteht zwischen den unter- und übergeordneten Komponenten eine „Ruft auf“ Beziehung.
3. Die Fehlermöglichkeiten werden identifiziert und den Modulen zugeordnet. In übergeordneten Modulen lassen sich die Fehlerfolgen finden.
4. Bewertung der Bedeutung, der Auftrittshäufigkeit und der Entdeckungswahrscheinlichkeit der Fehlermodi.
5. Es folgt die Definition von Gegenmaßnahmen und die Einarbeitung in das System.

Mäckel empfiehlt die Durchführung durch Werkzeuge unterstützen zu lassen, da zwischen den Modulen „mannigfaltige Verknüpfungsmöglichkeiten“ bestehen [5].

3.2. Betrachtung von Risiken durch Software-FMEA

Bei Softwaresystemen werden zwei Fehlerfortpflanzungswege und vier Fehlerarten unterschieden. Fehler können sich vertikal in der Modulhierarchie fortpflanzen, und so Fehlverhalten in weiteren, übergeordneten Modulen provozieren. Andererseits können sich Fehler auch horizontal entlang des Moduldatenflusses weiterverbreiten.[6] Anders als bei Hardwaresystemen können sich Fehler jedoch *nur* entlang der Kontroll- und Datenflüsse verbreiten. Ein mechanisches Übergreifen wie es in Hardwaresystemen zum Beispiel bei einem Bruch eines Stahlträgers vorstellbar ist, ist in Softwaresystemen nicht möglich. Es werden vier Fehlerarten unterschieden:

1. Eine Nachricht wird zu spät geschickt.
2. Eine Nachricht wird zu früh geschickt.
3. Eine Nachricht wird gar nicht geschickt.
4. Die geschickte Nachricht ist die falsche oder fehlerhaft.

Durch diese Einschränkungen ergibt sich für Softwaresysteme zumindest eine Beschränkung des Fehlersuchraums. Die Risikobewertung bei Softwaresystemen führt zu höheren Risikoprioritätskennzahlen.[5]

Mäckel bemerkt, dass gleiche oder ähnliche Risiken von unterschiedlichen Teams verschiedenartig bewertet werden. „Ein Vergleich über mehrere FMEAs hinweg muss daher scheitern."[5]. Er lehnt die Ableitung von Faustregeln darum ab. Vielmehr sollten die Risiken diskussionsbasiert bewertet werden. Demzufolge können auch Ergebnisse zwischen den herkömmlichen System-FMEAs und der Software-FMEA nicht verglichen werden. [5]

Aus diesem Grund, „wurde eine neue Vorgehensweise zur Kennzahlbildung bei Software-FMEAs für die Auftritts (A-Wert) und die Entdeckungswahrscheinlichkeit definiert"[5]. Die Bewertung der Fehlerauswirkung wird analog zur System-FMEA durchgeführt.

Bei der System-FMEA kann die Auftrittswahrscheinlichkeit aus Erfahrungswerten und Herstellerangaben abgeleitet werden. Dies gilt, ob des andersartigen Wesens von Software, nicht für die Software-FMEA. Allgemein anerkannt ist, dass die Fehlerhäufung in Systemen mit der Komplexität des Systems steigt. Gleichzeitig gilt, dass mit der Komplexität der Software die Entdeckungswahrscheinlichkeit für Fehler sinkt.[5]

Mäckel empfiehlt einen zweigeteilten Ablauf, um die Entdeckungs- und die Auftrittswahrscheinlichkeit zu bestimmen. Zuerst wird in Abhängigkeit von der Modulkomplexität eine Ausgangskennzahl festgelegt. Die Ausgangskennzahl ist umso höher, je größer die Modulkomplexität ist. Im zweiten Schritt wird diese Zahl in Abhängigkeit „von der Qualität und der Beherrschung der durchgeführten"[5] Vermeidungs- und Entdeckungsmaßnahmen reduziert. Die Kombination von Entdeckungs- und Vermeidungsmaßnahmen führt zu einer stärkeren Senkung der Ausgangskennzahl. Mäckel kommt für die Ausgangskennzahl zu Ergebnissen zwischen zwei und zehn. Je höher die Ausgangskennzahl dann noch ist, desto dringlicher sind geeignete Gegenmaßnahmen. Als geeignete Entdeckungs- und Vermeidungsmaßnahmen nennt Mäckel unter anderem [5]

- formale Verifikation
- Reviews
- Checklisten, Black-Box-Tests

bzw.

- formale Entwurfsansätze
- strukturierte Analyse
- Entwurfs- und Codierrichtlinien

Die Auftritts- und Entdeckungswahrscheinlichkeiten können also nicht mehr durch Wahrscheinlichkeiten, sondern nur noch durch Beschreibungen von Wahrscheinlichkeiten in die Software-FMEA einfließen. Mäckel selbst merkt an, dass bei kleinen, aber softwareintensiven Systemen, diese Art der Bewertung zu „nahezu gleichen[n] A- und E-Werte[n]"[5] führt.

3.3. Bewertung der Software-FMEA

Die Bewertung der Software-FMEA orientiert sich, wie die Bewertung der FMEA, an den Punkten Aufwand der Durchführung und Qualität der Ergebnisse.

Genau wie die FMEA verursacht die Durchführung einer Software-FMEA einen großen Aufwand, der sich aus Einarbeitung in das System, Umsetzung des Systems

in eine Software-FMEA taugliche Darstellung und Analyse zusammensetzt. Für die Software-FMEA gilt gleichermaßen was für bottom-up Systeme gilt.[6]

Auch für die Vollständigkeit des Ansatzes gilt das gleiche wie für die System-FMEA. Die Vollständigkeit ist abhängig vom betriebenen Aufwand. Durch den gleichen Ansatz, zeigt die Software-FMEA die gleichen Schwächen im Bereich zusammenhängender Fehler wie die System-FMEA. Auch hier werden die Effekte von gleichzeitig auftretenden Fehlern (multiple Fehler) nicht explizit betrachtet. Daher können unvorhergesehene Fehlereffekte auftreten, wenn zum Beispiel Fehler mit geringer RPZ nicht vermieden werden, und dadurch weitere Fehler auftreten

Auch die Software-FMEA sollte von einem interdisziplinären Team durchgeführt werden um verschiedene Perspektiven zu gewährleisten [3].

Vorteilhaft ist, dass die Methode sehr einfach und systematisch ist, auch für komplexe Systeme geeignet ist (Subkomponenten-Bildung), und dass (fast) alle Fehler im System gefunden werden können. Sie ist sowohl für eine Validation des Designs als auch für eine Analyse der Requirements geeignet[10]. Zudem können gezielt Testfälle mit Risiko- und Ausfallorientierung generiert werden, was für sicherheitsrelevante Software von enormer Bedeutung ist. [5] Die Software-FMEA trägt somit auch zu einer Verringerung von Kosten im Entwicklungsprozess bei, da fehlerhafte und risikobehaftete Module frühzeitig korrigiert bzw. sicher gemacht werden können.

Problematisch erscheint die Ermittlung und Bewertung der Ausfall- und Entdeckungswahrscheinlichkeiten. Die Vergabe einer Ausgangskennzahl anhand der Modulkomplexität kann sich nur auf wenige „harte“ Daten (z.B. Lines of Code) stützen, und erscheint vor allem ein intuitiver Prozess anhand von subjektiven Erfahrungswerten zu sein. Gleiches gilt für die Verringerung der Ausgangskennzahlen anhand der Vermeidungs- und Entdeckungsmaßnahmen. Insbesondere die Übertragung der Bewertung von Kombinationen kann wohl nur schwerlich von einem System auf ein vollkommen andersgeartetes System gelingen, sondern nur auf Erfahrungswerten basieren. Diese Erfahrungswerte ähneln dann jedoch stark den von Mäckel selbst kritisierten Faustregeln.[5]

Somit ist die Bewertung von Risiken in der Software-FMEA problematischer als in der System-FMEA.

4. Zusammenfassung

Bei der Konstruktion hybrider Systeme sind „ganzheitliche Betrachtungsweisen und Techniken, die auf unterschiedlich realisierte Systemkomponenten – z.B. Software, elektronische Komponenten, technische Prozesse - gleichermaßen anzuwenden sind“ [3] erforderlich. Ein Teilgebiet der möglichen Ansätze sind strukturorientierte Ansätze. Strukturorientierte Ansätze stellen das System in den Mittelpunkt der Betrachtung. Die Analyse der Struktur und ihrer Funktionsweise führt dann zu den Risiken die von dem System ausgehen.

Ein Beispiel für einen strukturorientierten Ansatz ist die Failure Modes and Effects Analysis. Die FMEA ist für Hardwaresysteme gedacht. Zusätzlich existiert ein Software-Pendant, die Software-FMEA. Durch prinzipiell gleichen Aufbau und gleiches Vorgehen gelten die Vor- und Nachteile der Vorgehensweise für beide Arten. Bei der (Software-) FMEA liegt eine Vorwärtsbetrachtung anhand von Ereignisketten innerhalb des Systems vor. Ausgehend von den Modulen des Systems werden die Fehlermodi gefunden und die Auswirkungen der Fehler betrachtet. Mit Hilfe der (Software-) FMEA können fast alle Fehler innerhalb eines Systems gefunden werden. Durch die frühzeitige Anwendung der (Software-) FMEA können somit auch Entwicklungskosten eingespart werden. Anders als bei der System-FMEA, bestehen bei der Software-FMEA jedoch Probleme bei der Bewertung von Auftritts- und Entdeckungswahrscheinlichkeiten. Dies liegt daran, dass für Softwaresysteme keine standardisier-

ten Werte wie zum Beispiel für die Lebensdauer von Sensoren gegeben werden können. Die Risikobewertung in der Software-FMEA zeigt sich somit als Problemfeld.

Problematisch an beiden Ansätzen sind der tendenziell hohe Aufwand und die Beschränkung auf wenige Phasen des Projektablaufs.

Durch die Vernachlässigung von domänen- und humanorientierten Fehlern, bieten die Ansätze nur eine beschränkte Sichtweise ([10] betont insbesondere auch die Wichtigkeit der Verknüpfung mit einer rückwärts gerichteten Suche). Die erbrachten Ergebnisse decken also grundsätzlich nicht alle Risiken ab, die im Zusammenhang mit dem betrachteten System stehen. Insofern ist also eine Einbindung von domänen- und humanorientierten Ansätzen in den Analyseprozess wünschenswert.

Referenzen

[1] Deutsche Gesellschaft für Qualität(Hrsg.), „Fehlermöglichkeits- und Einflussanalyse: FMEA“, 2., veränd. Aufl., Berlin, 2001
[2] VDA, „Sicherung der Qualität vor Serieneinsatz“, Qualitätsmanagement in der Automobilindustrie, Oberursel, 1996
[3] System Safety Society: System Safety Analysis Handbook. 2nd ed. Albuquerque : System Safety Society, 1999
[4] Liggesmayer, Peter et. al “Qualitätssicherung Software-basierter technischer Systeme – Problembereiche und Lösungsansätze” in Informatik- Spektrum 21 , S.249 – 258, 1998
[5] Mäckel, Oliver „Mit Blick auf´s Risiko – Software-FMEA im Entwicklungsprozess softwareintensiver technischer Systeme“, 2001
[6] Montenegro, Sergio „Sichere und fehlertolerante Steuerungen: Entwicklung sicherheitsrelevanter Steuerungen“ , 1999
[7] Leveson, Nancy „Safeware- System Safety and Computers“, Addison Wesley, 1995
[8] Maier, Thomas „FMEA and FTA to support Safe Design of Embedded Software in Safety-Critical Systems“ in „Safety and Reliability of Software Based Systems – Twelfth Annual CSR Workshop”, 1995
[9] Reifer, D.J. “Software Failure Modes and Effects Analysis” in “IEEE transactions on reliability”, Heft 28(3), New York, S.247-249, 1979
[10] Lutz, Robyn R., Woodhouse, Robert M. “Experience Report : Contribution of SFMEA to Requirements Analysis”

Kapitel 3. Post-Mortem-Ansätze/ Kontinuierliche Verbesserung

Pascal Adrian Hagedorn

Technische Universität Kaiserslautern
hagedor@rhrk.uni-kl.de

Kurzfassung. Methoden und Modelle für Post-Mortem-Ansätze und Ansätze zur kontinuierlichen Verbesserung beim Risikomanagement für sicherheitskritische Software-Systeme sind Gegenstand dieser Arbeit. Als Vertreter der Post-Mortem-Ansätze wird die Root-Cause-Analyse (RCA) beschrieben, die sich vor allem nach Vorfällen mit weit reichenden Auswirkungen, aber auch nach Vorfällen, aus denen man lernen möchte, anbietet. Als ein ganzheitliches Konzept der kontinuierlichen Verbesserung wird das Systems Security Engineering Capability Maturity Model (SSE-CMM) ausführlich vorgestellt. Ziel dieses Modells ist es, den Entwicklungsprozess kontinuierlich weiter zu entwickeln, wiederholbar zu machen, effizient zu gestalten und zu gewährleisten, dass durch diesen Entwicklungsprozess die angestrebte Qualität auch wirklich erreicht wird.

1. Bedeutung der Post-Mortem-Ansätze und kontinuierlicher Verbesserung im Rahmen des Software-Risikomanagements

Diese Seminararbeit befasst sich im Rahmen des Seminars „Risikomanagement für sicherheitskritische Softwaresysteme“ mit kontinuierlichen Verbesserungs- und Post-Mortem-Ansätzen. Dabei werden Methoden und Modelle vorgestellt, die sowohl in späteren Lebenszyklusphasen, als auch während der Entwicklung Anwendung finden können.

Im ersten Abschnitt wird die grundsätzliche Bedeutung der Post-Mortem-Ansätze und des Ansatzes der kontinuierlichen Verbesserung beschrieben. Der zweite Abschnitt behandelt eine Möglichkeit der Kategorisierung und nennt verschiedene Methoden und Modelle, die sich für die Fragestellung dieses Seminarthemas anbieten. Jeweils eine ausgewählte Methode und ein ausgewähltes Modell wird in Abschnitt 3 näher vorgestellt. Dies sind hier im speziellen das SSE-CMM, ein aspekt- und prozessorientiertes Modell, und die Root Cause Analyse als Vertreter der vorfallzentrierten Post-Mortem-Ansätze. Nachdem in 3.3 die positiven und negativen Aspekte dieser beiden Ansätze gegenübergestellt werden, zieht der vierte und letzte Abschnitt ein Fazit und versucht einen Ausblick auf zukünftige Entwicklungen zu geben.

1.1. Risikomanagement

[1:59ff] definiert Risikomanagement als eine 3-Teilung in Risikoplanung, -kontrolle und -überwachung, und auch in [2:Kapitel 15] findet man im Rahmen des operationalen Risikomanagements eine Definition des Risikobegriffs und seinem Management. Wir verzichten an dieser Stelle aus Platzgründen auf eine präzise Definition von Risikomanagement und wollen darunter im Folgenden ganz grob die Vermeidung, die Erkennung und die Beseitigung von Risiken bzw. Schwachstellen verstehen.

Es gibt die verschiedensten Gründe, sich mit dem Software-Risikomanagement zu befassen. Ein Grund ist die steigende Zahl der Anwendungsgebiete, in denen Software zum Einsatz gelangt und in denen die Software eigenständige Aufgaben erledi-

gen kann. Ein weiterer Grund ist die Anwendung von Software in sicherheitskritischen Domänen, in denen man auf Schwachstellen sofort reagieren muss. Solche Domänen sind vor allem in der Atom-, der Luft- und Raumfahrt- sowie der Rüstungsindustrie anzutreffen, in denen es neben großen monetären Werten vor allem um das Leben und die Gesundheit von Menschen geht. Eine weitere Begründung für das Software-Risikomanagement liegt außerdem in den hohen Kosten der Fehlerbeseitigungen und Änderungen während der Softwareentwicklung. Dort sind die Kosten für die Mängelbeseitigung in der Regel in den späten Phasen um ein Vielfaches höher als in den frühen Phasen. Deshalb sollte die entwickelnde Organisation so zeitig wie möglich – also möglichst schon bei der Anforderungsspezifikation – Methoden anwenden, die dem Gedanken der kontinuierlichen Verbesserung Rechnung tragen.

1.2. Kontinuierliche Verbesserung und Kaizen

Der Gedanke der kontinuierlichen Verbesserung wurde erstmals in Japan nach dem 2. Weltkrieg umgesetzt, und wird dort seitdem in Form des Kaizen erfolgreich angewandt. Nach [3:138ff] resultiert aus den unterschiedlichen Lebensphilosophien eine nur eingeschränkte Übertragung des Konzeptes von japanischen auf westliche Unternehmen. Demnach legen die Japaner schon in ihrer Erziehung großen Wert auf den „Gemeinschaftssinn“, wohingegen die Europäer und Amerikaner die Leistung des Einzelnen priorisieren.

Beim Kaizen ist die Anwendung von „Hightech“ von niedrigerer Bedeutung, denn das Konzept gelangt durch eine ganzheitliche Sichtweise der Dinge zur vollen Entfaltung. Dabei können schon einfachste Mittel zum Erfolg führen, nur muss dass Ziel – also die kontinuierliche Verbesserung – konsequent verfolgt werden. Aus dieser Sichtweise heraus bedarf es beim Kaizen keiner gesonderten Definition von „Qualität“, sondern durch das kontinuierliche Bestreben nach Verbesserungen wird hier von einer implizierten „Qualität“ ausgegangen.

Ohne das rechtzeitige Aufdecken und Lösen von Problemen, was sich das Kaizen als Aufgabe gesetzt hat, kann es auch nicht zur Verbesserung kommen. Der Grundsatz „Fehler*vermeidung* ist ökonomisch, Fehler*beseitigung* ist unökonomisch [gilt, und] Fehlervermeidung kann aber, wie schon gesagt, nur *dann* praktiziert werden, wenn Probleme und Schwachstellen *gesucht* und *genannt* werden“[3:140].

Neben dem Menschen (und seiner Ideologie) stehen beim Kaizen auch Prozesse im Mittelpunkt, die den gesamten „Produktionsprozess“ beschreiben. Diese Prozesse sind beim Kaizen der Gegenstand des Verbesserungsinteresses. Als Ansätze solcher Prozessverbesserungen kann zum einen der *PDCA-Zyklus* genannt werden, wobei die Abkürzung für *P*lan, *D*o, *C*heck und *A*ct steht und zum anderen der *IDEAL-Ansatz*, bei dem die Buchstaben für *I*nitiating, *D*iagnosing, *E*stablishing, *A*cting und *L*earning stehen. Letzt genanter Ansatz, wird im – in Abschnitt 3.1 vorgestellten – SSE-CMM Modell zur Anwendung bevorzugt.

Die Abgrenzung zwischen Post-Mortem-Ansätzen und der „kontinuierlichen Verbesserung“ besteht darin, dass kontinuierliche Verbesserung eigentlich immer stattfinden sollte. D.h. in jeder Phase des Projekts treten Probleme auf, die Verbesserungspotential besitzen. Dessen sollte man sich immer bewusst sein.

Post-Mortem-Ansätze sind hingegen in den späten Lebenszyklusphasen anzusetzen, mit dem Grundsatz „aus Fehlern zu lernen“, der dem vorigen Gedanken der kontinuierlichen Verbesserung Rechnung trägt.

1.3. Post-Mortem und Vorfälle

Bei Post-Mortem-Ansätzen ist das „Kind“ im ungünstigsten Falle schon in den sprichwörtlichen „Brunnen“ gefallen. Um aus dem unerwünschten Ergebnis wenigstens

einen möglichst großen Nutzen zu ziehen, muss eine systematische Fehlersuche und –analyse ausgeführt werden. Hierbei wird in [4:21] von einem „vorfallzentrierten Ansatz“ gesprochen.

Natürlich müssen nicht zwangsläufig nur bei negativen Ereignissen Post-Mortem-Ansätze durchgeführt werden. Auch positive Ereignisse, die in Projekten gut liefen, können durch systematische Analyse bestimmt werden, und zum Beispiel in Form von Empfehlungen in anderen Projekten Anwendung finden.

In folgenden Abschnitt 2 werden Methoden und Modelle vorgestellt, die für eine erfolgreiche Umsetzung der kontinuierlichen Verbesserung und der Post-Mortem-Ansätze berücksichtigt werden können.

2. Methoden und Modelle

[5] führt mehrere Methoden auf, die sich mit Risikofeldern beschäftigen. Dabei geht es im Allgemeinen um Kernkraftwerke, Elektrizitätswerke und ähnliche sicherheitsbedürftige Bereiche. Wie das Thema verrät, liegt das Hauptaugenmerk dieser Arbeit auf Post-Mortem-Ansätzen und kontinuierlicher Verbesserung. Dies beinhaltet vor allem späte Lebenszyklusphasen. Der folgende Abschnitt soll einen Überblick zur möglichen Kategorisierung bieten.

2.1. Betrachtungsebenen

Die Risikobetrachtung sicherheitskritischer Systeme kann in verschiedenen Lebenszyklusphasen auf unterschiedlichen Betrachtungsebenen erfolgen. Dies liefert eine mögliche Klassifikation der Ansätze zur Schwachstellenerkennung gemäß Tabelle 3-1.

Die makroskopische *Betrachtungsebene* kann in die Aspekt- und Prozessebene geteilt werden. Die Analyse auf Aspektebene hat zum Ziel, „alle relevanten Gesichtspunkte die Einfluss auf die Gewährleistung von vorhersagbarer, bezahlbarer Produktqualität haben, möglichst vollständig zu berücksichtigen und ausreichend zu würdigen“ [4:13]. Wenn sich eine Organisation darauf festgelegt hat, welche Problembereiche sie angehen wird, muss auf *Prozessebene* entschieden werden, wie der abstrakte Workflow aussehen kann, d.h. „wer macht wann was?“ [4:14]. Ziel auf dieser Ebene ist es zu erkennen, ob alle notwendigen Schritte mit ausreichender Güte unter kontrollierten Bedingungen durchgeführt werden.

Eine Gliederung der *mikroskopischen Ebene*, bringt eine Methoden- und eine Produktebene hervor. Auch auf diesen Ebenen müssen die Schwachstellen in den Methoden und Produkten gefunden werden.

Betrachtungsebene		Potentielle Schwachstellen	Lebenszyklusphasen						
			Anforderungsanalyse	Entwurf	Realisierung	Test	Installation	Wirkbetrieb	Außerbetriebnahme
Makroskopisch	**Aspekte:**	**„Worauf muss man achten?“**							
	Problembereiche	Wurden alle relevanten Gesichtspunkte bedacht und durch Prozesse behandelte?							
	Reife (Tiefe der Aufgabenerfüllung)	Wird gleich bleibende Güte und maximale Effizienz der Prozesse erreicht?							
	Prozesse:	**„Wer macht wann was?“**							
	Workflow (Artefakte, Trigger, Transformationsschritte)	Ist die Durchführung korrekt, vollständig, transparent, steuerbar?							
	Agenten (Durchführungs- und Kontrollinstanzen)	Führen geeignete Rollen die Aufgaben durch?							
	Kontrolle	Werden Wirksamkeit und Effizienz ausreichend überwacht?							
mikroskopisch	**Methoden:**	**„Wie führt man es aus?“**							
	Eignung	Wie effektiv arbeitet die Methode?							
	Verfahrensweisen	Ist die Vorgehensweise klar, das Resultat aussagekräftig?							
	Ressourcen (Personal, Werkzeuge, Hilfsmittel)	Sind Hilfsmittel in ausreichender Güte und Menge verfügbar?							
	Produkte/Artefakte:	**„Was ist die resultierende Qualität?“**							
	Anforderungen	Werden alle relevanten Anforderungen des Einsatzfeldes befriedigt?							
	Qualitätsstufe	Erfüllt das Produkt/Artefakt die Anforderungen hinreichend gut?							

Tabelle 3-1: „*Schwachstellen* auf verschiedenen *Ebenen* und in verschiedenen *Phasen des Lebenszyklus*“[2:14]

2.2. Methoden

In diesem Abschnitt werden verschiedene Methoden aus [5] kurz vorgestellt, die sich für die Post-Mortem-Analyse eignen.

Accident Analysis [5:8ff]
Unfall- und vorfllzentrierte Methode, die zur Bestimmung der auslösenden Ursache dient.

Das Ziel ist es, die Auswirkungen von un-/wahrscheinlichen Unfällen bestimmter Szenarien zu evaluieren. Die Umsetzung dieser Methode fordert von den Teilnehmern Kenntnisse und Erfahrungen in den jeweiligen Anwendungsszenarien.

Repetitive Failure Analysis (RFA) [5:264ff]

Im Mittelpunkt dieser Methode liegen Wiederholungsfehler. Diesen wird systematisch nachgegangen, indem die Ursache für das Auftreten gesucht, die Evaluation des Vorfalls vorgenommen und die Lösung zur Beseitigung des Fehlers gefunden wird.

Durch das wiederholte Auftreten eines Fehlers an einem beliebigen Punkt ist die Funktionalität des Systems oder Subsystems nicht mehr gewährleistet. Dies zu verhindern ist die Aufgabe der RFA.

Anwendung findet sie vor allem in der Energie- und Nuklearindustrie. Aufgrund steigender Wiederverwendung von Softwaremodulen in der Softwareentwicklung, birgt die RFA auch ein beträchtliches Potential für die Softwareindustrie.

Root Cause Analysis (RCA) [5:270f]
Die Konzentration auf die grundlegenden Ursachen (Root Causes) der aufgedeckten Probleme und Vorfälle ist Aufgabe der RCA. Der Gedanke, dass der vordergründige Auslöser nicht der grundlegende Auslöser ist und dass die Beseitigung des Root Cause die Entstehung ganzer Fehlerbündel unterbindet, begründet die Anwendung dieser Methode.

Anwendung findet sie bereits in der Luft- und Raumfahrt, der atomaren Energieerzeugung sowie dem Gesundheitswesen. Abschnitt 3.2 gibt einen ausführlichen Überblick über diese Methode.

Safety Review [5:272ff]
Die Idee die hinter dieser Methode steht, ist die regelmäßige Untersuchung der implementierten Risikomanagement-Maßnahmen. Dies geschieht durch regelmäßige Inspektionen, die durch den Anwender selbst ausgeführt werden und durch die er gezwungen ist sich mit der Materie ausführlich auseinander zu setzen. Des Weiteren muss er sich nach seinen Inspektionen den Fragen eines fachkundigen Inspektions-Teams stellen.

Die Methode besteht vornehmlich aus den drei Schritten

- Vorbereitung der Überprüfung,
- Durchführung der Überprüfung und
- Dokumentation der Ergebnisse,

wobei dem ausführenden Analytiker die Anwendung neuer Technologien frei gestellt wird.

Der Erfolg der Methode hängt von der Ausführlichkeit der gesammelten Informationen ab. Vorteilhaft erweist sie sich in Bezug auf das Finden weiterer Auswirkungen, die ein Ereignis haben kann, nachdem eine Katastrophe aufgetreten ist.

Software Fault Tree Analysis [5:294ff]
Eigentlich ist diese Methode für alle Schritte der Design-Phase gedacht, um den Root Cause eines obersten unerwünschten Ereignisses aufzuspüren – also für eine sehr frühe Phase. Aber durch den systematischen Versuch, Fehler aufzuspüren, scheint sie auch in Post-Mortem-Fällen geeignet.

Durch einen Baum von logischen Gattern kann die Ausführung der Software nachvollzogen werden. Die Methode steht in enger Verknüpfung zur Fault Tree Analysis [5:132ff] die sich hauptsächlich mit Hardware- und Bedienfehlern beschäftigt. Einschränkend wirkt die enorme Komplexität eines größeren Software-Systems, dem aber durch Modularität entgegengewirkt werden könnte.

Neben der Anwendung bei der Entwicklung von Atomwaffen, findet die Methode auch reges Interesse in der Eisenbahn- und Flugsteuerungsentwicklung.

Statistical Process Control [5:305f]
Das Steuern und Kontrollieren von Prozessen ist Aufgabe der Statistical Process Control Methode. Aus dieser Aufgabe heraus soll es dem Anwender gelingen, Produkte von gleich bleibender Qualität zu erzeugen. Gemessene Informationen sind auch hier die Essenz der Methode, die bei ausreichender Verfügung zur Prozesskontrolle verwendet werden können. Misst das System Werte die von den früheren Werten abweichen, so können gewisse Gegenmaßnahmen eingeführt werden. Was jedoch gemessen werden kann ist oft die Frage, hat man diese aber sinnvoll beantwortet, stellt sie zumindest ein wirksames Mittel zur frühen Vorfallerkennung dar.

Die Gemeinsamkeit, die diese Methoden vor allem in Bezug auf die Post-Mortem-Ansätze aufweisen, ist vor allem die vorfallzentrierte Sichtweise, d.h. die Frage nach der eigentlichen ursprünglichen Ursache des Qualitätsmangels.

Um – nach Meinung des Autors – die beste Methode zur Post-Mortem-Analyse auszuwählen, bedarf es einer „allumfassenden" Infrastruktur, die es ermöglicht Messungen vorzunehmen, die Informationen liefern. Diese sollen dann ausgewertet werden und zur Ursachenfindung dienen.

Im Folgenden 3. Abschnitt soll solch ein „allumfassendes" Modell, neben einer ausgewählten Methode, ausführlicher vorgestellt werden.

2.3. Modelle

Bei einem Modell handelt es sich, im Gegensatz zu einer Methode, um eine „gröbere", aber ganzheitlichere, Sichtweise der Problemstellung. Aus dieser Ganzheitlichkeit heraus resultieren umfangreichere Dokumente, die dieses Modell beschreiben. Man könnte Modelle auch als eine grundlegende Ideologie verstehen, die mit Methoden anzureichern sind, um eine konkrete Vorgehensweise zu erhalten. Dieser Ideologie liegt auch der Gedanke der kontinuierlichen Verbesserung zugrunde.

Da das SSE-CMM die Aspekte der „Sicherheit" (SSE) beachtet und den vorher genannten Verbesserungsgedanken durch CMM realisiert, erscheint dem Autor dieses Modell für die Problemstellung dieses Themas geeignet zu sein.

Der folgende Abschnitt 3.1 stellt das SSE-CMM als ganzheitliches Modell und die Root Cause Analyse in Abschnitt 3.2 als einzelne Methode ausführlicher vor und versucht daraufhin die Ansätze in 3.3 zu bewerten.

3. Ausgewählte Methoden und Modelle

Die Wahl des SSE-CMM als Modell ist durch den ganzheitlichen Ansatz auf Aspekt- und Prozessebene begründet, wohingegen die Root Cause Analyse hier aufgrund ihrer relativ einfachen und systematischen Praktikabilität gewählt wurde.

Der Grund für die Entwicklung des im Folgenden vorgestellten Modells bestand in dem Mangel an einem übergreifenden Framework, das zur Evaluierung der Security Engineering Praktiken geeignet war. Kunden und Lieferanten von „sicherer" Software hatten gleichermaßen Interesse an solchen Verbesserungen, die der Entwicklung von Sicherheitsprodukten, -systemen und –dienstleistungen dienten.

3.1. Systems Security Engineering Capability Maturity Model (SSE-CMM) [6]

Bei dem hier vorgestellten Modell handelt es sich um ein aspekt- und prozessorientiertes Modell. Ursprünglich war dieses Modell dafür gedacht, Sicherheit in Informations-Technologie (IT) Domänen zu implementieren. Doch die Erfahrungen, die mit diesem Modell gesammelt werden konnten, zeigten, dass seine Nützlichkeit und Anwendbarkeit auch auf andere Security Domänen übertragbar sind. Wie oben

schon erwähnt liegt innerhalb der IT Domäne das Hauptaugenmerk auf den Prozessen, genauer gesagt auf der Reife (maturity) dieser Prozesse. Dabei wird aber kein bestimmter Prozess vorgegeben, sondern es werden die Prozesse berücksichtigt, für die sich die Organisation beliebige - auch aus anderen IT Leitfäden - Anregungen holen kann.

Obwohl es sich bei dem SSE-CMM um ein separates Modell handelt, ist es nicht ausschließlich als eigenständige Aktivität anzusehen. Vielmehr wird im Modell selbst zum Beispiel das Common Feature „Coordinate Security Practices“ (Koordiniere Sicherheitspraktiken) berücksichtigt, welches wiederum die Integration von Sicherheit in allen Disziplinen und Gruppen, die in das Projekt involviert sind, vorantreibt.

Geschichtliches. Im April 1993 begann die SSE-CMM-Initiative als ein von der National Security Agency (NSA) unterstütztes Arbeitsteam. Es stützte sich auf die bereits bestehenden Arbeiten über CMM, mit dem Ziel, ein auf Security Engineering spezialisiertes CMM zu entwickeln.

Auf einem Security Engineering Workshop im Januar 1995 beteuerten 60 Repräsentanten von verschieden Organisationen die Notwendigkeit eines solchen Modells, und schon im März trafen sich die ersten Arbeitsgruppen.

Die erste Version des Modells wurde dann im Oktober 1996 präsentiert. In der zweiten Version wurde vor allem auf die praktikable Anwendung und Umsetzung des Modells geachtet, für die eigens Arbeitsgruppen gebildet wurden.

Daraufhin bildete sich die International Systems Security Engineering Association (ISSEA), die sich von diesem Zeitpunkt an mit der Weiterführung des Projektes beschäftigt, und sich auch als Ziel gesetzt hat, das SSE-CMM als internationalen Standard (ISO/IEC 21827) durchzusetzen.

Security Engineering. Da keine allgemeingültige Definition des Security Engineering existiert, kann im SSE-CMM das Security Engineering wie folgt interpretiert werden:

- Sicherheitsrisiken, die mit einer Organisation in Verbindung stehen, müssen vollständig verstanden werden.
- Erforderliche Schutzmechanismen zu den identifizierten Risiken müssen eingeführt werden.
- Sicherheitsbedürfnisse müssen in Sicherheitsanweisungen, für die jeweils betroffene Domäne, transformiert werden.
- Vertrauen in die Korrektheit und Zuverlässigkeit der eingeführten Sicherheitsmechanismen muss geschafft werden.
- Akzeptierbarkeit des verbleibenden Restrisikos muss sichergestellt werden.
- Stärkung des Verständnisses eines vertrauenswürdigen Systems über alle beteiligten Ingenieur-Disziplinen und Spezialisten muss erreicht werden.

Dabei wird explizit auf die Interdisziplinarität über alle Lebenszyklusphasen und Interessengruppen hingewiesen.

Warum ist Security Engineering wichtig?
Sowohl dem Wandel zur Dienstleistungs- und Informationsgesellschaft und dem damit verbundenen Anstieg an produzierten Informationen, als auch der Zunahme des Einsatzes von Software in kritischen Anwendungen verdankt das Security Engineering seine Existenz.

Von Anfang an stand daher der Einsatz von Software in der Luft- und Raumfahrt, der Atomindustrie und der Rüstungsindustrie im Mittelpunkt der Anstrengungen des Security Engineering. Es weitete sich aber auch immer mehr auf Finanztransaktio-

nen, persönliche Daten, usw. bis hin zum weltweiten Internet aus. Diese Allgegenwart von Software und das damit einhergehende Bedürfnis nach Sicherheit, zeigt die Wichtigkeit dieser Disziplin, und findet daher in allen nur erdenklichen Sparten Anwendung.

Das SSE-CMM versucht

- alle Security Engineering Aktivitäten, die das Produkt/Sicherheitssystemleben überdauern,
- alle Anwender – der verschiedensten Disziplinen –, welche Sicherheitsdienste anbieten, und
- alle Formen und Größen von Security Engineering Organisationen

zu umfassen. Aus diesem Verständnis heraus definiert sich das SSE-CMM seine Ziele folgendermaßen.

Ziele des SSE-CMM. Das SSE-CMM Projekt versucht Security Engineering als eine definierbare, ausgereifte Disziplin mit messbaren Zielen voranzutreiben. Bei der Entwicklung und Anwendung von Sicherheitssystemen mit Hilfe des SSE-CMM stehen folgende Qualitäten, deren Erreichung die Hauptziele des SSE-CMM darstellen, im Vordergrund:

- *Kontinuität (Continuity):*
  Sicherheit, dass Wissen, welches in vorhergehenden Aktionen gewonnen wurde, in zukünftigen Aktionen nutzbringend angewandt werden kann.
- *Wiederholbarkeit (Repeatability):*
  Sicherstellung, dass erfolgreiche Aktionen in Projekten wiederholt werden können.
- *Effizienz (Efficiency):*
  Unterstützung, dass effiziente Arbeit der Entwickler und Evaluatoren möglich wird.
- *Gewährleistung (Assurance):*
  Gewissheit, dass die angestrebte Qualität auch wirklich erzielt wird.

Auf der einen Seite, sollen diese Qualitätsziele dem Kunden helfen, die implementierten Sicherheits- und Schutzmechanismen des Herstellers beurteilen zu können. Andererseits sollen sie dem Hersteller helfen, die gewünschten Qualitätsniveaus vorhersagbar zu erreichen.

Um all diesen Anforderungen gerecht zu werden, muss in den Organisationen ein „Leitfaden“ individuell erstellt werden, der ihnen dabei hilft, diese Ziele zu erreichen.

Die Prozess- und Produktverbesserungen stehen hier im Mittelpunkt des Interesses. Hierbei wird unter Prozess eine Sequenz von Schritten verstanden, um einen bestimmten Zweck zu erreichen.

Das Capability Maturity Model (CMM) stellt dabei ein Framework dar, eine unorganisierte und uneffektive Unternehmung in eine strukturierte und effiziente Organisation zu wandeln. Die angewandten Praktiken sollen durch dieses Modell unter statistische Prozesskontrolle geraten, um so die Leistungsfähigkeit der Prozesse zu erhöhen. Nach Angaben des Software Engineering Institut der Carnegie Mellon Universität haben Organisationen, welche CMM für Software anwendeten, positive Resultate in Bezug auf Kosten, Produktivität, Zeitplanung und Qualität erzielt. Die Anwendung dieser Prozesskontrolle auf das Security Engineering verspricht genau diese Ziele – nämlich vorhergesagte Kosten, Zeitpläne und Qualitäten – einzuhalten.

Realisierung der Verbesserungen. Das SSE-CMM stellt eine Ansammlung von Security Engineering Methoden bereit, die ein gewisses Maß an Hintergrundwissen zum Security Engineering voraussetzen.

Grundpraktiken und Prozessbereiche
22 Prozessbereiche (Process Areas) beinhalten *129 Grundpraktiken (Base Practices)*, wovon sich 11 Prozessbereiche auf Security Engineering und die anderen 11 auf Projekt- und Organisationsdomänen beziehen.

Grundpraktiken

- finden über die gesamte Laufzeit der Unternehmung Anwendung,
- überlappen sich nicht gegenseitig,
- repräsentieren eine „best practice“ der Security Community,
- müssen nicht dem Stand der Technik entsprechen,
- sind in verschiedenen Zusammenhängen anwendbar und
- spezifizieren keine konkreten Methoden und Werkzeuge.

Prozessbereiche

- vereinigen ähnliche Aktivitäten zur einfacheren Anwendung,
- finden über die gesamte Laufzeit der Unternehmung statt,
- können in verschiedenen domänenspezifischen Kontexten implementiert werden,
- können als einzelner Prozess verbessert werden,
- beinhalten alle Grundpraktiken, die benötigt werden, um das Ziel des Prozessbereiches zu erreichen.

Generische Praktiken
Generische Praktiken (Generic Practices) sind auf alle Prozesse anwendbar und beziehen sich auf Aspekte des Managements, der Messbarkeit und der Institutionalisierung eines Prozesses. Sie sind gruppiert in *Common Features*, die sich wiederum in die nachfolgend erläuterten fünf für das CMM typischen Reifegrade aufspalten lassen. Im Gegensatz zu den Grundpraktiken sind die Generischen Praktiken nach den Reifegraden angeordnet. Ein Auf- oder Abstieg innerhalb der Common Features ist auf eine große Prozessveränderung innerhalb der Organisation zurückzuführen.

Reifegrade. Wie im vorigen Absatz erwähnt, können die Common Features in eine hierarchische Ordnung innerhalb der Reifegrade gebracht werden. Folgende Sätze geben die Ideen, die hinter den Reifegraden stehen, wieder [6:44f]:

1. „You have to *do it* before you can manage it.“
2. „*Understand what’s happening* on the project before defining organization-wide processes.“
3. „*Use the best of what you’ve learned* from your projects to create organization-wide processes. “
4. „*You can’t measure it until you know what ’it’ is.*“ und „Managing with measurement is only meaningful when you’re measuring the right things.”
5. „A *culture of continuous improvement* requires a foundation of sound management practice, defined processes, and measurable goals.”

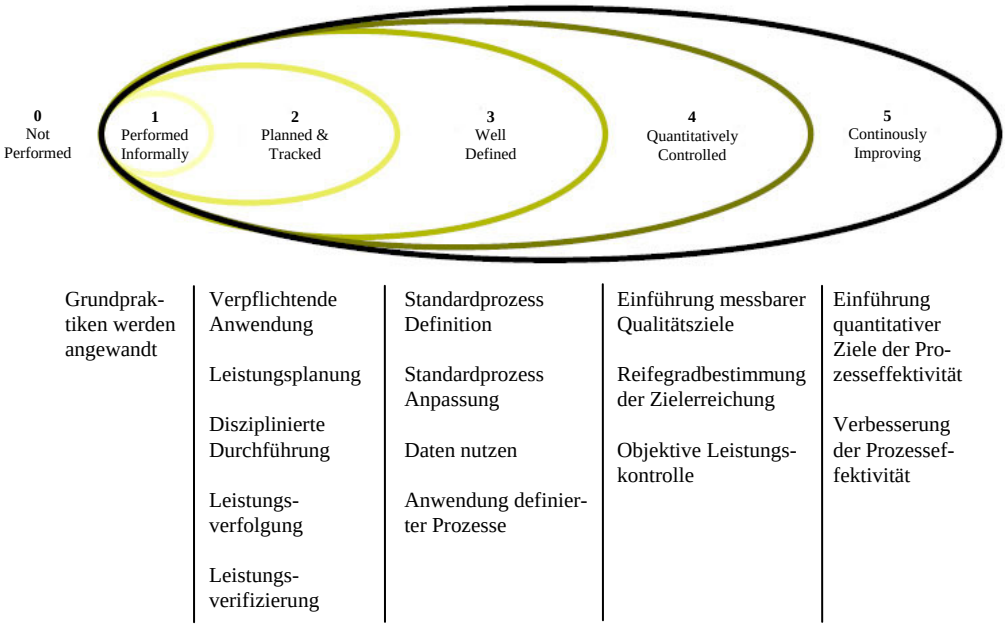


Abbildung 3-1. Reifegrade spiegeln die Fähigkeiten der Organisation wider [6:44]

Abbildung 3-1 versucht diesen Zusammenhang zu visualisieren und zeigt auch, dass höhere Reifegrade immer die Fähigkeiten der niedrigeren implizieren. In letzter Konsequenz bedeutet das, wenn in einen Reifegrad nicht alle Common Features umgesetzt sind, wird der Reifegrad verfehlt. Im schlimmsten Fall praktiziert eine Organisation nicht einmal die Grundpraktiken und erreicht somit nicht einmal den Reifegrad 1.

Der Organisation wird damit eine konsequent aufeinander folgende Umsetzung der Common Features in den einzelnen Bereichen nahe gelegt, was dem Erreichen der Unternehmensziele dienen soll.

Kombination der Praktiken. Durch die Trennung zwischen Grundpraktiken und Generischen Praktiken ist eine Kombination der Praktiken möglich. Diese Trennung vollzieht sich in der 2-dimensionalen Darstellung des Modells. Eine Dimension ist die „Domäne“, die andere der „Reifegrad“. In der Domäne wird dem Security Engineering durch die *Grundpraktiken* Rechnung getragen, der Reifegrad wird durch die *Generischen Praktiken* bestimmt. Abbildung 3-2 zeigt eine konkrete Kombination der Praktiken um einen bestimmten Reifegrad zu erreichen. So kann man daraufhin einen schnellen Überblick über die Fähigkeiten einer Organisation erlangen, indem die Fragestellung eines interessierten Kunden beispielsweise folgendermaßen lauten würde: „Zeichnet Ihre Organisation die Ergebnisse von Messungen auf, um gewisse Risiken zu beobachten?“ Mit einem kurzen „Ja“ erhält der Kunde zumindest einen schnellen Überblick über die grundsätzliche Aktivität. Durch ausführliche Dokumente kann man außerdem zusätzliche Informationen über den Reifegrad erhalten.

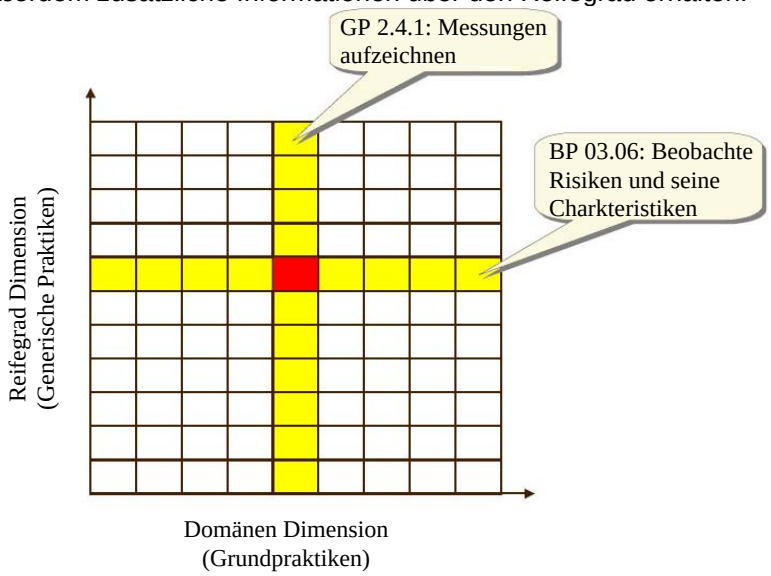


Abbildung 3-2. 2-Dimensionale Darstellung des Modells [6:39]

Mit der Beantwortung aller – aus der Kombination der Praktiken – resultierenden Fragen, kann sich jeder Interessent ein angemessenes Bild der Security Engineering Fähigkeiten der Organisation machen.

Tabelle 3-2 verdeutlicht den Zusammenhang der Prozessbereiche und der Common Features innerhalb des SSE-CMM. Wie weiter oben schon erwähnt, besteht jeder Prozessbereich aus verschiedenen Grundpraktiken die sich in den Prozessbereichen individuell gruppieren. Ebenso besteht jedes Common Feature aus bestimmten Generischen Praktiken.

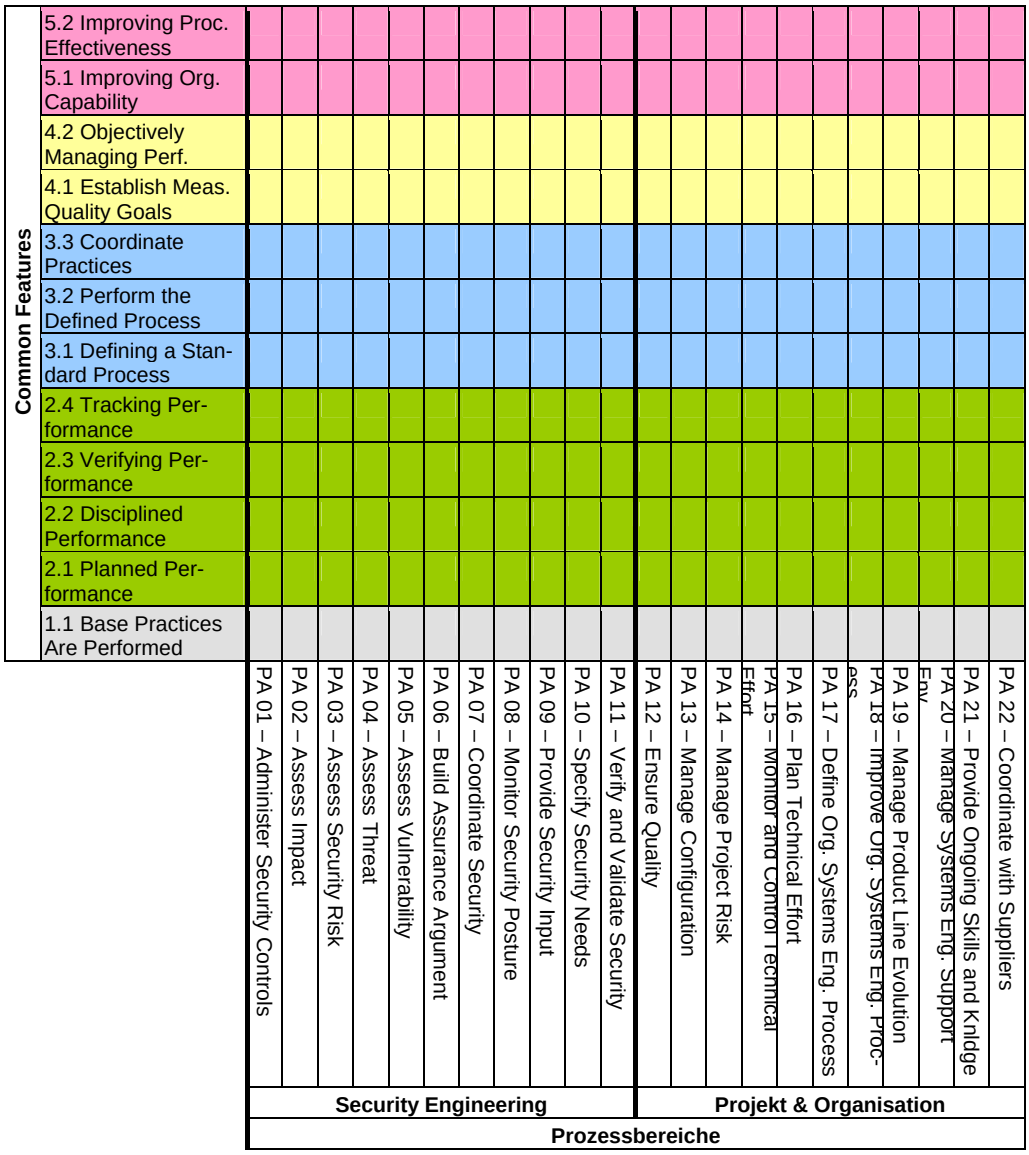


Tabelle 3-2. Zusammenfassung der Praktiken und Reifegrade in einer Matrix [6:49]

Security Engineering Prozesse (nach der SSE-CMM-Terminologie). Wie in Abbildung 3-3 dargestellt, wird der Security Engineering Prozess in 3 Hauptbereiche unterteilt. Trotz der klaren Trennung des Security Engineering Prozess in Risikoprozess, Engineering-Prozess und Gewährleistungsprozess sind diese Prozesse in starker Abhängigkeit untereinander. Diese Prozesse arbeiten zusammen, um die Ziele des Security Engineering zu erreichen.

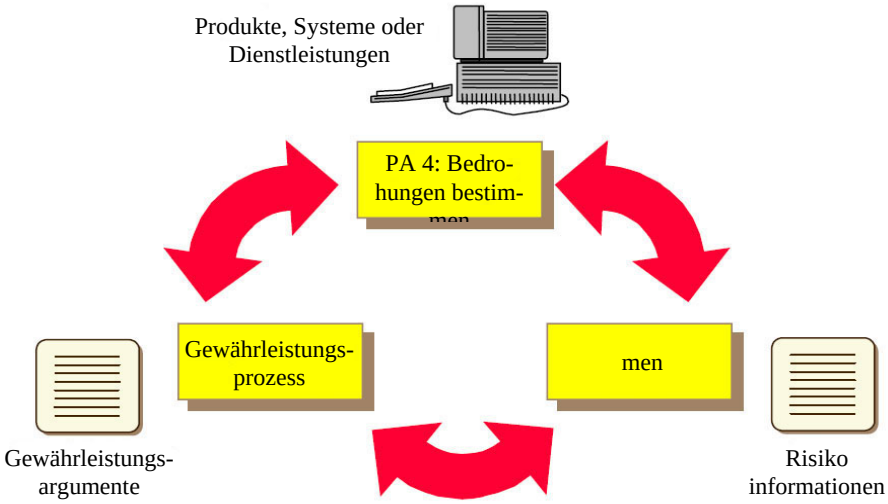


Abbildung 3-3. Die *drei Hauptbereiche des Security Engineering Prozess* [6:33]

Risikoprozess
Dieser Prozess ist für die Identifizierung und Priorisierung der – das Produkt oder System betreffenden – Risiken zuständig. Hierbei helfen die Eintrittswahrscheinlichkeiten der Bedrohungen und die Schwachstellen des Systems sowie die möglichen Auswirkungen des ungewollten Vorfalls. Die Eintrittswahrscheinlichkeit ist jedoch mit einem gewissen Maß an Unsicherheit verbunden, wodurch die Voraussagen nur bis zu einem bestimmten Maß eintreffen werden. Das gleiche gilt für das Ausmaß des Vorfalls. Die Umsetzung zuverlässiger Schutzvorrichtungen ist Aufgabe des Risikoprozesses.

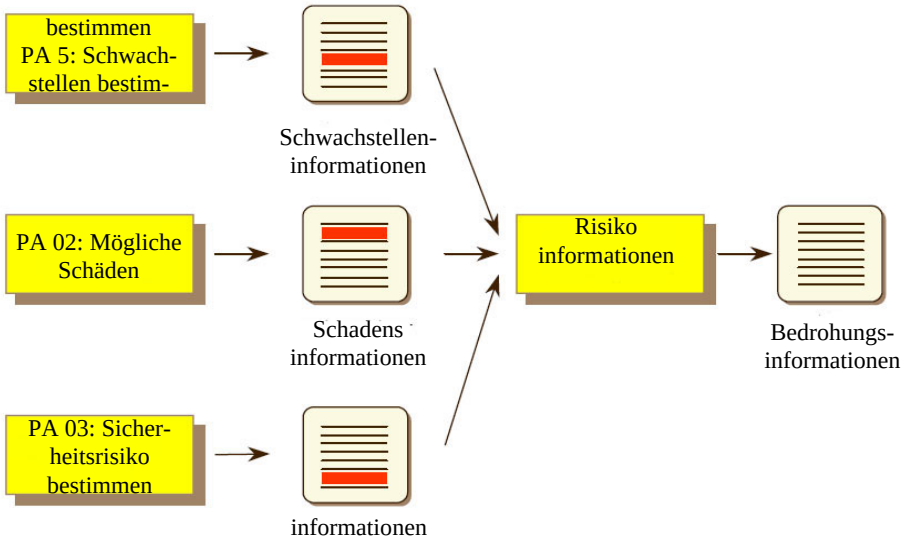


Abbildung 3-4. Der *Risikoprozess* betrachtet *Bedrohungen, Anfälligkeiten/Schwachstellen und Auswirkungen* des Risikos [6:34]

Gibt es keine Bedrohungen für das System oder keine Schwachstellen innerhalb des Systems, so wird es kein ungewolltes Risiko geben und damit wird auch kein Vorfall auftreten. Das hier praktizierte Risikomanagement ist die Bewertung und Quantifizierung von Risiken und dem damit verbundenen akzeptablen Restrisiko für die Organisation.

Engineering-Prozess
Bei diesem Prozess steht eine Lösungsfindung – unter Einbeziehung der verschiedenen beteiligten Disziplinen – für die gefundenen Risiken und Probleme im Zentrum des Interesses. Auch das Security Engineering durchläuft die klassischen Lebenszyklusphasen der Ingenieurswissenschaften (Anforderungsanalyse, Entwurf, Realisierung, Test, Installation, Betrieb, Wartung und Außerbetriebnahme).

Um sicherzustellen, dass die Sicherheit ein Teil des übergeordneten Prozesses ist, stehen die Security-Ingenieure in engem interdisziplinärem Kontakt zu Ingenieuren anderer Disziplinen. Dabei werden die Informationen des Risikoprozesses und andere Informationen genutzt, um Sicherheitsbedürfnisse zu spezifizieren, die dann gezielt vom Sicherheits-Ingenieur verfolgt werden. Bei der Auswahl von Realisierungsalternativen spielen die Kosten, die Realisierbarkeit, die technischen Alternativen und andere Fragestellungen eine Rolle.

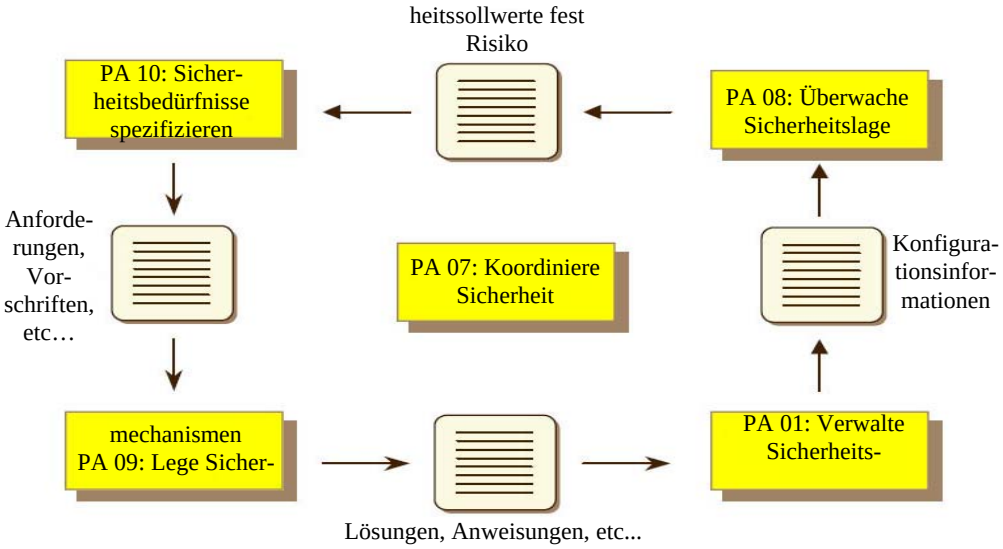


Abbildung 3-5. Sicherheit ist ein integraler Bestandteil des Engineering Prozesses [6:35]

Gewährleistungsprozess

Die Verlässlichkeit der Lösungen nachzuweisen und dem Kunden Vertrauen darin zu vermitteln, ist die Aufgabe dieses Prozesses. Hier steht zum Beispiel das Vertrauen in die Wiederholbarkeit stark im Mittelpunkt. Es werden zwar keine Gegenmaßnahmen angeboten, aber es wird versucht das Vertrauen zu stärken, dass die Kontrollmaßnahmen das vorhergesehene Risiko in ein akzeptierbares Restrisiko mindern. Wichtig ist hierbei, dass Sicherheitsvorkehrungen immer korrekt arbeiten, demzufolge ist es Aufgabe des Prozesses, zum Beispiel die korrekte Funktionalität der Sicherheitsvorkehrungen nachzuweisen. Des Weiteren kann die Gewährleistung auch als eine Art „Wachmann“ interpretiert werden, der bei kritischen Werten geeignete Gegenmaßnahmen vornimmt.

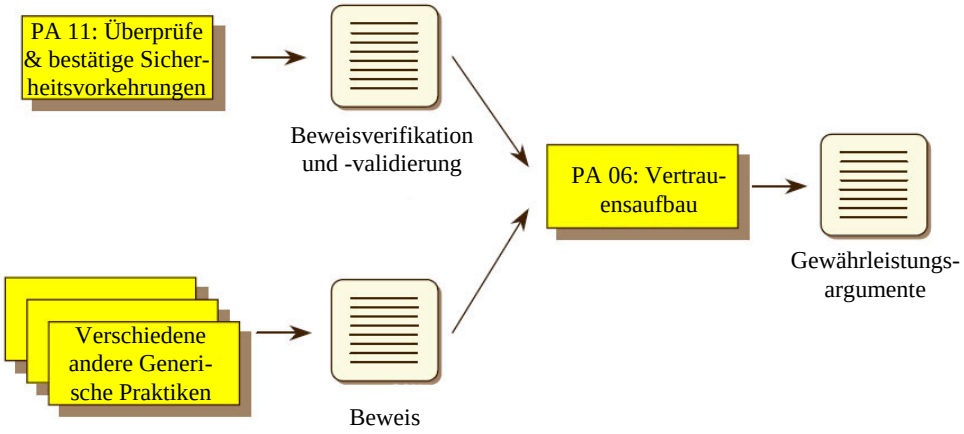


Abbildung 3-6. Der Gewährleistungsprozess führt zur Vertrauensbildung [6:36]

3.2. Root Cause Analyse (RCA) [4]

Ausgangssituation der RCA ist ein konkreter Zwischenfall, der schwerwiegende oder häufig wiederkehrende Schäden nach sich zieht. Die RCA versucht den Vorfall ganzheitlich zu analysieren und den so genannten „Root Cause" zu finden. Durch die Vorfallzentrierung der Methode, die den Betrieb voraussetzt, findet diese Methode hauptsächlich in späten Lebenszyklusphasen Anwendung.

Die ganzheitliche Betrachtungsweise versucht allen Betrachtungsebenen ausreichend Rechnung zu tragen. D.h. Sicherheitsaspekte, Prozessfehler, Methodenmängel und Produktmängel sollen durch die systematische Durchführung zumindest aufgedeckt werden können. Durch die Beseitigung des „Root Cause" kann ein ganzes Bündel möglicher Folgefehler eliminiert werden, auch solche, die bisher noch nie aufgetreten sind. Dies ist die besondere Stärke dieser Methode. Das daraus gewonnene Wissen kann dann als Feedback – zum Beispiel in Form einer Checkliste – an die frühen Lebenszyklusphasen in neuen Projekten zurückgegeben werden.

Eine gute Informationsinfrastruktur ist für den erfolgreichen Einsatz dienlich und kann zum Beispiel durch Schulungen der Mitarbeiter gefördert werden. Vor dem Auftreten des Vorfalls sollte deshalb sinnvoller weise solch eine Informationsinfrastruktur errichtet worden sein, denn wenn die Informationen nicht rechtzeitig erfasst werden, stehen sie im Krisenfall nicht zur Verfügung.

Der folgende Analysevorgang stellt eine *mögliche* Vorgehensweise dar und ist keinesfalls als starrer Vorgang zu sehen. Er sollte daher immer den individuellen Bedürfnissen der Organisation angepasst werden.

Analysevorgang. Ist es zu einem Vorfall gekommen, muss ein RCA-Team gebildet werden. Idealerweise sollte man zwei Teamleiter benennen. Als Moderator könnte zum Beispiel ein Experte für den RCA-Prozess und die RCA–Sitzung ernannt werden und als Fachleiter beispielsweise ein Experte für die betroffene Domäne. Das Team selbst sollte mindestens einen Mitarbeiter der jeweils betroffenen Abteilung integrieren, im Zweifel sogar mehr als nur einen, denn auch scheinbar unbeteiligte Personen können effektive Lösungen mit einbringen. Entscheidend für den Erfolg der Analyse ist, dass die Mitarbeiter den RCA-Prozess als wichtiges Mittel gegen das Auftreten von Wiederholungsfehlern verstehen und nicht, dass schuldige Mitarbeiter gefunden und bestraft werden sollen. Dieses Vertrauen muss gestärkt werden, indem vor allem die vermeintlich „Schuldigen" immer gut informiert und im Analyseprozess involviert sind.

Zum ersten Treffen muss dann jedes Team-Mitglied seinen individuellen schriftlichen chronologischen Ablauf des Vorfalls angefertigt haben, der die folgenden Fragestellungen beantwortet [4:140]:

- Was geschah der Reihe nach?
- Was waren die eigenen Beobachtungen?
- Was ging dem Beobachter dabei durch den Kopf?"

Die zwei Teamleiter versuchen außerdem zusätzliche Informationen, wie zum Beispiel Log-Daten und Betriebsunterlagen, Systemhandbücher und Arbeitsanweisungen, Zeugenaussagen oder Fachliteratur, für die 1. Sitzung zu recherchieren und zu dieser mitzubringen.

1. Sitzung

Zusammen mit allen Sitzungsteilnehmern wird dann eine Rekonstruktion des Vorfalls vorgenommen. Um den Vorgang für alle übersichtlich visualisieren zu können, können Flipcharts, Tafeln, Projektoren und RCA-Software zur Unterstützung hinzugezogen werden. Hat man sich auf einen ungefähren Ablauf geeinigt, soll festgestellt wer-

den, welche Gegenmaßnahmen während des Vorfalls tatsächlich ergriffen wurden. Negative Ereignisse, die in irgendeiner Form begünstigend für den Vorfall waren, müssen notiert werden. Anschließend wird durch ein Brainstorming alles geäußert, was in irgendeinen Zusammenhang mit dem Vorfall gebracht werden kann. Speziell wird dabei nach Barrieren gefragt, die versagt haben oder überhaupt nicht vorhanden waren.

Die daraus gewonnenen Faktoren werden auf Karten notiert und – nachdem Duplikate eliminiert wurden – auf einer Tafel nach selbst gewählten Oberbegriffen gruppiert. Eine Umgruppierung können die Teilnehmer so lange vornehmen, bis alle damit zufrieden sind. Im Abstand von ca. einer Woche findet die nächste Sitzung statt.

Zwischen 1. und 2. Sitzung
Moderator und Fachleiter stellen daraufhin mit den aus der 1. Sitzung gruppierten Oberbegriffen einen „Faktorenbaum“ auf, an dessen Spitze der Root Cause steht und auf der 2. Ebene die eben genannten Oberbegriffe. Darunter, auf der 3. Ebene, werden die Faktoren jeweils unter den zugehörigen Oberbegriffen gleichberechtigt angeordnet. Die zu jedem Faktor gegebenen Antworten auf die Fragen „warum der Faktor Einfluss auf den Schadensverlauf genommen hat und in welcher Weise die Wirkung zustande kam“[4:143] werden in den unter den Oberbegriffen liegenden Ebenen eingetragen. Dabei werden die Blätter des Baumes mit „begünstigend“, „irrelevant“ oder „fehlende Information“ gekennzeichnet, um somit die jeweilige Bedeutung des Faktors hervorzuheben. Blätter mit „fehlender Information“ werden bis zur nächsten Sitzung um ausreichende Informationen ergänzt, indem man Verantwortliche für die Beschaffung der fehlenden Information bestimmt.

2. Sitzung
Die nun gesammelten fehlenden Informationen werden dem Faktorenbaum ergänzend hinzugefügt und dieser wird erneut präsentiert. Korrekturen und Ergänzungen werden unter den Teilnehmern diskutiert, mit besonderem Augenmerk auf die begünstigenden und irrelevanten Blätter, wobei Letztere besonders hinterfragt werden müssen, ob sie wirklich als irrelevant markiert bleiben können. Auch wenn diese Blätter daraufhin ihren Status nicht ändern, müssen sie aus Managementgründen in der Dokumentation besonders begründet werden. Um gute Resultate zu erzielen, können domänenspezifische Checklisten von Vorteil sein.

In einem Maßnahmenplan (siehe Tabelle 3-3) können Verantwortliche zu den verschiedenen Begünstigungsfaktoren ernannt werden, welche die Gegenmaßnahmen bis zu einem gewissen Zeitpunkt durchgeführt haben müssen. Nach diesem Zeitpunkt kann eine Erfolgskontrolle vorgenommen werden und bei Bedarf zum Beispiel eine spezielle Task Force gebildet werden. Die dritte und meist letzte Sitzung findet wieder im Abstand von ca. einer Woche statt.

Begünstigungs-faktor	**Gegen-maßnahme**	**Verant-wortlicher**	**Durchzuführen bis …**	**Erfolgs-kontrolle**
Auskommentierter Code in alten Software-Versionen	Alle auskommentierte Code-fragmente löschen	Erik Muster-mann	17. KW	✓
…	…	…	…	…

Tabelle 3-3. Grundgerüst eines *Maßnahmenplans* [4:145]

3. Sitzung
Jetzt ist es erneut Aufgabe der Teamleiter, den bereinigten Faktorenbaum, die Ursachen des Vorfalls und den komplettierten Maßnahmenplan den anwesenden Teil-

nehmern zu präsentieren, woraufhin alle Mitglieder erneut Einwände und Korrekturen diskutieren können. Die Ideen, die während des gesamten Analysevorgangs entwickelt und dokumentiert wurden, werden nun nochmals durchgegangen, um sicher zu gehen, ob jede Idee ausreichend bedacht wurde. Dies geschieht mit Hilfe der Fragen [4:146]:

- „Wurden alle Ideen angemessen verwertet?“
- „Gibt es noch interessante Punkte, die man aufgreifen sollte?“

Das Analyseergebnis sollte von einem ausgewählten Verantwortlichen, vor allem an den Melder des Vorfalls, an die betroffenen Abteilungen und – sofern vorhanden – an eine Abteilung für Risikomanagement verteilt werden. Eine Kontrollinstanz muss die erfolgreiche Durchführung des Maßnahmenplans überprüfen und gewährleisten können, auch damit den Teammitgliedern der Nutzen und Sinn des Analysevorgangs näher gebracht wird.

3.3. Bewertung des SSE-CMM und der RCA

Zuallererst sollte man sich vor Augen führen, in welchem Kontext des Seminars die hier vorgestellten Modelle und Methoden zu sehen sind. Sicherheitskritische Software-Systeme stehen dabei im Mittelpunkt des Interesses. Es geht hier also um Systeme, bei denen Menschen in Gefahr gebracht werden können. Dies kann von einfachen Softwaresteuerungen innerhalb des Autos bis hin zum komplexen Echtzeitbetriebssystem eines Atomkraftwerks reichen. Überall dort, wo Fehler in der Software immer mit dem Verlust von Menschenleben oder kostenintensiven Vorfällen verbunden sein können, macht es Sinn, über geeignete Maßnahmen nachzudenken.

Aus diesem Gedanken heraus sind die Kosten und der Aufwand der Implementierung dieser Modelle und Methoden, im Vergleich zu den Folgekosten eines Vorfalls, eher gering einzustufen. Nichtsdestotrotz müssen die Ansätze immer an den Anwendungsbereich angepasst werden, was sie für den einmaligen und spontanen Einsatz disqualifiziert.

SSE-CMM. Die eindeutige Prozessorientierung dieses Modells, und die damit verbundene Vernachlässigung der Methoden- und Produktebenen, stellt das Modell auf den ersten Blick in einem schlechten Licht dar. Bei genauerer und umfassenderer Betrachtung scheint dieses Modell jedoch als Framework für eine Organisation äußerst geeignet zu sein. Denn es kann als eine Art „ideologische Rahmenbedingung“ für die Organisation verstanden werden.

Der hauptsächliche Blick des SSE-CMM auf die Prozessebene lässt den „Anwendern“ in Bezug auf die Realisierung innerhalb der mikroskopischen Betrachtungsebene ausreichend Gestaltungsspielraum, um das Modell den individuellen Anforderungen der Organisation anzupassen.

Das Modell versucht mit dem Gedanken der „Qualitätsverbesserung in allen Bereichen“ eine unweigerlich daraus resultierende Sicherheit zu erreichen. Theoretisch spricht dafür einiges. Dies liegt vor allem an der systematischen und peniblen Abdeckung aller organisatorischen Bereiche durch die 2-dimensionale Aufteilung zwischen „Domäne“ und „Reife“. In der Praxis ist die konsequente Umsetzung aber relativ kostenintensiv.

Der Mangel an konkreten Handlungsanweisungen kann in Hinblick auf die spezielle sicherheitskritische Ausrichtung der Organisation als positiv bewertet werden. Denn eine Organisation, die sich mit der Entwicklung sicherheitskritischer Software-Systeme beschäftigt, muss ihre Prozesse den individuellen Bedürfnissen anpassen und kann sich keine „Lösungen von der Stange“ erlauben.

Die alle Lebenszyklusphasen überspannende Anwendung des Modells eignet sich als grundlegendes Konzept für die Umsetzung sowohl innerhalb einer herstellenden Organisation als auch beim Kunden selbst. Verschiedene Formen und Größen der anwendenden Organisationen haben keine Auswirkung auf die Effektivität des Modells, spielen aber in der Kostenfrage eine erhebliche Rolle, denn dieses Modell erzeugt zum Beispiel für die sinnvolle Dokumentation der Aktivitäten einen großen Personenaufwand.

Als abschließende Bemerkung zum SSE-CMM muss der interdisziplinäre Charakter dieses Modells hervorgehoben werden. SSE-CMM bietet sich gerade als ein Modell für sehr große sicherheitskritische Systeme an, die die Software-Entwicklung mit anderen sicherheitsrelevanten Ingenieurdisziplinen verknüpfen kann.

RCA. Die Root-Cause-Analyse kann sicherlich als effektive Methode zur Findung der Ursachen eines Zwischenfalls beschrieben werden. Das Vorhandensein einer gut funktionierenden Infrastruktur in Bezug auf Informationen kann hilfreich sein, oft führen aber entscheidende Zeugenaussagen oder Einschätzungen erfahrener Praktiker schon zu einem raschen Ergebnis. Ein gutes Ergebnis kann somit nur erreicht werden, wenn genügend Daten und genaue Störmeldungen auftreten und gesammelt werden können. So kann also nur mit ausreichender Betriebserfahrung das vorhandene Potential anhand der RCA abgeschöpft werden. Jedoch darf die Latenz zwischen Vorfall und Ausführung der Analyse nicht zu groß sein, da zum einen zu viele unnötige und irritierende Informationen auftreten können, und zum anderen wichtige Informationen verloren gehen können. Eine Selektion der zu untersuchenden Vorfälle anhand der Schadensrelevanz kann die RCA effektiver gestalten, denn trotz relativ geringer Kosten kann eine willkürliche Anwendung der Methode in end- und sinnlosen Sitzungen enden.

Neben dem Post-Mortem-Aspekt berücksichtigt die RCA ebenso die Idee der kontinuierlichen Verbesserung. Durch die gesammelten und gewonnenen Erkenntnisse aus dem Analyseverfahren können Materialien erstellt werden, die neuen Projekten Feedback über laufende oder schon beendete Projekte geben. Wichtiger ist aber noch: Das Niveau der laufenden Prozesse wird durch Abstellen potentieller Root Causes immer weiter angehoben.

Zusammenfassung. In Anbetracht der Tatsache, dass es sich bei der Einsatzumgebung um sicherheitskritische Software-Systeme handelt, können die Kosten der Ansätze hier zumindest als zweitrangig betrachtet werden. Ein Beispiel der Anwendung findet sich in [7:93ff], bei dem die NASA das CMM für die Entwicklung der Flug-Software (Primary Avionics Software System), die während allen Phasen des Fluges im Shuttle aktiv ist, einsetzt. Das Leben der Crew und die enormen Entwicklungskosten rechtfertigen den Einsatz des Modells und zeigen ein mögliches Einsatzgebiet auf.

Die Auswahl der beiden – in diesem Abschnitt ausführlich erläuterten – Ansätze wurde vom Autor aus folgenden Gründen gewählt.

Ein Grund ist zum Beispiel, dass in dieser Seminararbeit eine große Abdeckung der in Tabelle 3-1 aufgeführten Kategorisierung erreicht wird. Das SSE-CMM deckt die „horizontale“ Prozessebene über alle Lebenszyklusphasen ab, die RCA hingegen deckt die „vertikalen“ späten Lebenszyklusphasen umfassend ab.

Als weiterer Grund kann angeführt werden, dass das SSE-CMM ein theoretisches Konzept mit großem „Spielraum“ darstellt, im Gegensatz dazu die RCA eine konkrete praktische Methode repräsentiert, die keine großen Interpretationen zulässt.

Weiterhin steht die „kontinuierliche Verbesserung“ in engem Kontakt zum SSE-CMM, und der vorfallzentrierte Ansatz der RCA trägt den Post-Mortem-Ansätzen Rechnung.

Eine allgemeingültige Beurteilung der hier erwähnten Methoden und Modelle kann nicht gegeben werden. Das Anwendungsgebiet der sicherheitskritischen Software-Systeme ist zu komplex und kann daher nicht von einem Ansatz alleine bewältigt werden. Nur durch eine ausführliche Befassung mit dem Einsatzgebiet, in dem man sein (Software-)System einsetzen oder entwickeln möchte und mit einer sinnvollen und angepassten Kombination verschiedener Ansätze, kann diese Komplexität annähernd bewältigt werden.

4. Fazit und Ausblick

Die Implementierung eines ganzheitlichen und allumfassenden Ansatzes zur Erhöhung der Sicherheit in sicherheitskritischen Software-Systemen kann als vorteilhaft, aber auch als sehr kostenintensiv beurteilt werden. Diese Abwägung muss jede Organisation, die in diesem Bereich tätig ist, für sich selbst treffen. Das wichtigste Kriterium dabei stellt sicherlich das Risiko, Menschenleben zu verlieren, dar. Nach [1:68] kann im Allgemeinen die Risikoanalyse und das Risikomanagement aber als sinnvoll erachtet werden, wenn die Kosten des Risikos die Kosten der Einführung eines solchen Systems übersteigen. Für eine erfolgreiche Umsetzung muss ausreichend Fach- und Domänenwissen vorhanden sein, denn alle für das Risikomanagement geeigneten Ansätze bieten den Experten genügend Flexibilität, sich die Methoden und Modelle nach ihren Bedürfnissen anzupassen.

Problem aller Ansätze ist aber der Mangel an allgemeingültigen Ausführungsanweisungen, was aus der Individualität und Komplexität der Projekte resultiert. Soweit es konkrete Anweisungen gibt, sind diese nicht auf jeden Problemfall anwendbar. Auch wird gerade in der Literatur mehr über den Begriff „Qualität“ als über den Begriff „Sicherheit“ geschrieben. Dies kann daran liegen, dass der Begriff der Qualität „eindimensional“ gesehen werden kann – also Qualität kann verbessert oder verschlechtert werden. Der Begriff der Sicherheit – gerade in der Informationstechnologie-Branche – hat hingegen viele Facetten: Wie die beiden Begriffe Safety und Security verdeutlichen, können beide nicht an einem einzelnen Merkmal festgemacht werden, sondern nur an einem Bündel von „Do`s“ und „Dont`s“. Der Einsatz von Software im täglichen Leben bis hin zum Einsatz in sicherheitskritischen Software-Systemen zeigt die vielfältige Interpretationsmöglichkeit des Sicherheitsbegriffs.

Gerade in Bezug auf dieses Seminarthema und seine hier vorgestellten Methoden und Modelle sollte das Verständnis für den Nutzen dieser Ansätze, gerade beim Anwender, gestärkt werden, anstatt sich in Schuldzuweisungen zu üben oder Probleme zu ignorieren. Die steigende Nachfrage nach gutem und sicherem Risikomanagement für Software-Systeme wird in Zukunft sicher steigen, denn der steigende Einsatz von Software in allen Bereichen führt zu einem wachsenden Sicherheitsbedürfnis. Die Geschichte hat gezeigt, dass blindes Vertrauen in Software oft genug zu Katastrophen geführt hat.

Referenzen

[1] R. N. Charette, Software Engineering Risk Analysis and Management. New York: McGraw-Hill, 1989.

[2] The Federal Aviation Administration. System Safety Handbook: Practices and Guidelines for Conducting System Safety Engineering and Management. Dezember 2000 Erhältlich: www.asy.faa.gov/RISK/SSHandbook/contents.htm

[3] P. Dilg, Praktisches Qualitätsmanagement in der Informationstechnologie: von ISO 9000 zum TQM. München, Wien: Hanser, 1995.

[4] R. Schwarz, Methoden zur systematischen Bekämpfung von Sicherheitsschwachstellen, Fraunhofer Institut Experimentelles Software Engineering (IESE), Kaiserslautern, Report Nr. 087.00/D, Dezember 2000
[5] The System Safety Society (SSS) System Safety Analysis Handbook. Kapitel 3, August 1999
Erhältlich: www.system-safety.org
[6] Carnegie Mellon University/Software Engineering Institute. (Juni 2003). [Online]. Systems Security Engineering Capability Maturity Model SSE-CMM Model Description Document Version 3.0. Erhältlich: www.sse-cmm.org/docs/ssecmmv3final.pdf
[7] Carnegie Mellon University/Software Engineering Institute, The Capability maturity model: guidelines for improving the software process. Reading, Massachusetts, USA: Addison-Wesley, 1995.

Kapitel 4. Human Factors

Sebastian Weber

Universität Kaiserslautern
s_weber@informatik.uni-kl.de

Kurzfassung. Human Factors, dieser Begriff bezeichnet die Wissenschaft der menschlichen Fähigkeiten, Beschränkungen und deren Einbringung in Prozesse und Produkte. Mit der Entwicklung zunehmend komplexerer Systeme wird dieses wissenschaftliche Feld immer weiter in den Fokus von Industrie und Forschung gerückt. Insbesondere bei sicherheitskritischen Systemen kann ein Fehler im Design der Mensch-Maschine Schnittstelle verheerende Folgen haben. Sicherheit und einfache Bedienbarkeit sind nur zwei mit Human Factors assoziierte Konzepte die häufig im Widerspruch zueinander stehen und deren Vor- und Nachteile es gegeneinander abzuwägen gilt. Dieser Beitrag beschreibt, welche Rolle die menschliche Wahrnehmung und die damit in Verbindung stehenden geistigen Verarbeitungsprozesse beim Entwurf von Mensch-Maschine-Schnittstellen spielen. Des Weiteren wird eine Entwurfsmethodik vorgestellt, die diesen Anforderungen Rechnung tragen kann.

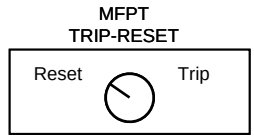
1. Einführung

Traditionelle Human Factors (HF) Erwägungen beim Entwurf von Mensch-Maschine-Schnittstellen (engl. Human Machine Interfaces, HMI) gibt es bereits seit der Zeit nach dem 1. Weltkrieg.

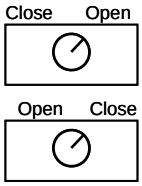
Diese unter dem Begriff Ergonomie bekannten Erwägungen zielten fast ausschließlich auf den Bereich der physischen Beschränkungen des Bedienpersonals, etwa Erreichbarkeit und Sichtbarkeit der Kontrollen. Es existiert eine Vielzahl an Richtlinien und Standards die sich mit diesem Gebiet beschäftigen. So etwa die Richtlinien der Verwaltungs-Berufsgenossenschaft für Bildschirmarbeitsplätze BGI 650, oder auch die Technical Standard Orders der US-amerikanischen FAA, um nur zwei zu nennen.

Trotz dieser Bemühungen menschlichen Beschränkungen Rechnung zu tragen kommt es auch in diesem lange bekannten Bereich noch zu geradezu katastrophalen Fehlentwürfen: Zusammen gehörende Anzeige- und Kontrollelemente werden an verschiedenen Enden eines Raumes installiert, Beschriftungen und Nummerierungen auf Kontrollboards scheinbar willkürlich angebracht (weitere Beispiele siehe Abbildung 4-1).

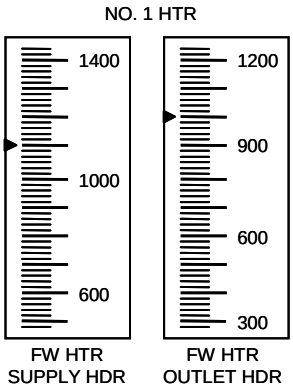
Mit der Einführung von Computern in Kontrollräumen in den 60er Jahren wurde der Entwurf der Mensch-Maschine-Schnittstelle mit einem mal um ein Vielfaches komplizierter. Mit Hilfe der computerisierten Steuerung konnten auch physikalisch weit voneinander getrennte Bereiche einer Anlage von einem einzigen zentralen Punkt aus überwacht werden. Statt zahlreicher verteilter, an die Erfordernisse vor Ort angepasster Kontroll- und Regelelemente wurden auf begrenzter Fläche plötzlich etliche hundert dieser Elemente installiert, mit der Folge, dass die einst wirksamen Warnmechanismen in der Masse untergingen. Viele der traditionellen Erkenntnisse wurden damit wertlos, unzählige neue Erfordernisse kamen auf. So musste beispielsweise die durch den Computer verursachte Indirektion der anfallenden Daten ausgeglichen werden: Konnte etwa Personal vor Ort anhand von Geräuschen oder Vibrationen eine Fehlfunktion einer Maschine diagnostizieren, so fehlten dem Operator im Kontrollraum im Regelfall derartige Signale.



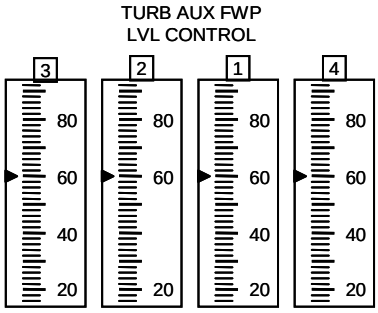
(a) Trip und Reset Einstellung vertauscht



(b) Inkonsistente Belegung



(c) Heizaggregat Druckanzeige. In Eile kann leicht angenommen werden, dass der Ausgangsdruck höher ist als der Eingangsdruck



(d) Befremdliche Nummerierung

Abbildung 4-1. Beispiele für einen mangelhaften Entwurf der Mensch-Maschine-Schnittstelle [1]

Wie bereits erwähnt ermöglicht der Computer die Überwachung einer Vielzahl von Prozessen von einem zentralen Ort aus. Das Bedienpersonal muss, häufig unter Zeitdruck, kritische Entscheidungen auf der Grundlage von Informationen treffen die der Computer auswählt, aufbereitet und präsentiert.

Um diesen neuen Anforderungen Rechnung zu tragen ist es nicht mehr ausreichend, die Kontrollelemente lediglich innerhalb der Reichweite eines menschlichen Operators zu platzieren. Bereits beim Entwurf des HMI muss berücksichtigt werden wie Menschen ‚funktionieren', wie sie Entscheidungen treffen und Alternativen bewerten und welche Informationen sie hierfür benötigen.

Das von Jens Rasmussen entwickelte und in Abschnitt 2 vorgestellte 3-Ebenen Modell der menschlichen Wahrnehmung und Steuerung kann bei der Realisierung der Kontrollfunktionen der Mensch-Maschine-Schnittstelle helfen [2]. Das in Abschnitt 3 beschriebene Operator Function Modell, eine Alternative zur traditionellen Task Analysis Methode, stellt ein Beschreibungs- und Vorhersagemodell für menschliche Verhaltensweisen in Interaktion mit komplexen Systemen zur Verfügung [3].

2. Menschliche Wahrnehmung

Der Hauptgrund, warum Menschen in automatisierten Systemen nach wie vor eine zentrale Rolle spielen, ist ihre Anpassungsfähigkeit. Ein Computer kann in Situationen für die er programmiert wurde zumeist schneller und effizienter reagieren als ein Mensch. Tritt jedoch ein Ereignis ein, welches beim Systementwurf nicht vorhergesehen wurde und für dessen Bewältigung keine Maßnahmen vorgesehen sind, führt

dies zwangsläufig zu einem Scheitern der Maschine. Derartige Situationen treten bei komplexen Systemen jedoch mit an Sicherheit grenzender Wahrscheinlichkeit ein.

Menschen betten vorhandene Informationen in einen Kontext ein. Sie versehen Daten mit zusätzlicher Semantik und bewerten neue Informationen oder Zustandsänderungen innerhalb eines Systems im Rahmen dieses Kontextes. Auf diese Weise ist es ihnen möglich auch auf Unvorhergesehenes zu reagieren und potentiell gefährliche Situationen zu entschärfen. Das kreative Verhalten bei der Problemanalyse und –bewältigung macht den Menschen für komplexe automatisierte Systeme unentbehrlich. Es ist häufig der ‚Leim' [3], der es verschiedenen Teilsystemen überhaupt erst ermöglicht ihre Aufgabe zu erfüllen.

Diesem Entscheidungsfindungsprozess liegen Muster zugrunde die Rasmussen in einem Modell zusammengefasst hat (siehe Abbildung 4-2).

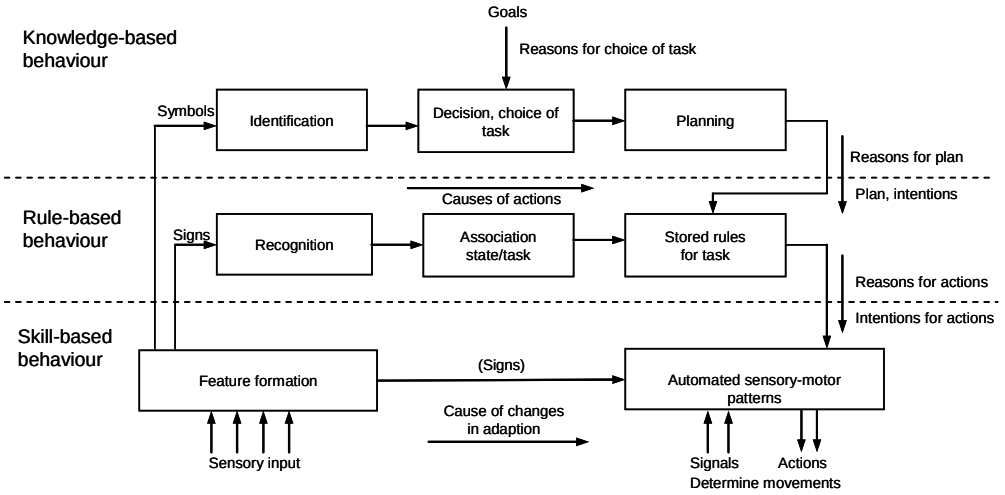


Abbildung 4-2. Rasmussens Drei-Ebenen Modell [2]

2.1. Rasmussens Drei-Ebenen Modell

Basierend auf seiner Untersuchung von Fehlerberichten in Atomkraftwerken identifizierte Rasmussen drei Ebenen der kognitiven Steuerung: Fertigkeitenbasiertes Verhalten (*skill-based behaviour*) welches durch Bewegungsmuster in einem dynamischen, gedanklichen Modell der Umwelt charakterisiert ist; regelbasiertes Verhalten (*rule-based behaviour*) welches auf impliziten in prozeduralen Regeln eingebetteten Modellen der Umgebung fußt; sowie wissensbasiertes Verhalten (*knowledge-based behaviour*) welches auf strukturellen Modellen der Umwelt aufbaut [1].

2.2. Fertigkeitenbasiertes Verhalten

Unter fertigkeitenbasiertem Verhalten versteht Rasmussen fließendes, hoch integriertes Wahrnehmungs-Bewegungs-Verhalten (*sensory-motor behaviour*) wie es für das beinahe unbewusste Durchführen von Routineaufgaben charakteristisch ist. Etwa dem Fahren eines Autos auf einer bekannten Strecke.

Grundlage für dieses Verhalten ist eine Feedforward-Kontrolle, bei der unerwünschte Veränderungen in einem Prozess vorausgesehen werden und diesen entgegengewirkt wird. Eine derartige Kontrolle erfordert sehr flexible und effiziente Modelle der Umgebung sowie den Einsatz von speziellen Wahrnehmungsindikatoren (*sensory-motor inputs* oder *signals*).

Die Güte derartiger Handlungen wird durch eine Feinanpassung des menschlichen Verhaltens an die Prozessumgebung erhöht. Bevor es zu einer derartigen Anpassung kommen kann ist es jedoch nötig, die Grenzen des Systems bzw. der eigenen Leistungsfähigkeit auszuloten. Dieses Ausloten wird erst durch ein überschreiten selbiger Grenzen möglich. Zum Erlangen einer Fertigkeit bzw. zu ihrem Erhalt auf einem bestimmten Level ist also ein Trial-&-Error-Lernprozess notwendig. Versucht man diese notwendigen Fehler zu unterbinden nimmt man dem Menschen zugleich die Möglichkeit effizient zu reagieren.

2.3. Regelbasiertes Verhalten

Mit regelbasiertem Verhalten meint Rasmussen ein bewussteres Lösen von bekannten Problemen. Auf dieser Ebene werden Handlungsfolgen in vertrauten Situationen von vorgefertigten Regeln (*stored rules*) oder Prozeduren (*heuristics*) gesteuert.

Die Auswahl der für eine Situation geeigneten Regeln gestaltet sich schwierig. In komplexen Systemen ist es dem Benutzer selten möglich alle Einflußgrößen zu beobachten und in den Entscheidungsprozess mit einzubeziehen. Aus diesem Grund wird lediglich eine kleine Anzahl von Variablen als Indikator für den Zustand eines Systems herangezogen (*signs*). Wichtige Informationen werden unter Umständen außen vor gelassen, das Vertrauen in die Richtigkeit bestimmter Anzeigen wird überbewertet und folglich wird eine Handlung entsprechend einer falschen Regel durchgeführt.

Um effiziente Regeln zu entwickeln und die jeweils wichtigen Indikatoren zu identifizieren muss der Mensch Erfahrungen mit dem System sammeln. ‚Experimente' mit dem System liefern notwendiges Wissen über Zustandsänderungen und Verhaltensweisen in Reaktion auf bestimmte Operationen. Derartige Experimente führen unter Umständen zu Fehlersituationen im System, sind für den Erfahrungserwerb jedoch unerlässlich.

2.4. Wissensbasiertes Verhalten

Wird ein Mensch mit einer unbekannten oder unerwarteten Situation konfrontiert so versucht er laut Rasmussen die aufkommenden Probleme auf einer höheren konzeptuellen Ebene anzugehen. Wissensbasiertes Verhalten ist durch eine starke Zielorientierung gekennzeichnet. Basierend auf einer Analyse der Umgebung und dem jeweiligen Ziel wird ein Plan zur Erreichung dieses Ziels erstellt.

Im Unterschied zum regelbasierten Verhalten werden Informationen über den Systemzustand nicht als Indikatoren benutzt die Regeln auslösen, sondern eher als Ausdrücke einer stärker formalen Logik (*symbols*). Dieser Übergang gestaltet sich laut Rasmussen schwierig. Bei regelbasierter Entscheidungsfindung ist von einer recht starken Voreingenommenheit des Menschen auszugehen. Die Indikatoren werden in einen Kontext von bekannten Situationen eingebettet, abweichende Anzeichen werden fälschlicherweise ignoriert, da sie nicht zu diesen bereits bekannten Situationen passen. Wissensbasiertes Verhalten hingegen erfordert eine sehr viel stärkere Objektivität bei der Analyse des Systemzustandes und das Überwinden dieser Voreingenommenheit.

Eines der Werkzeuge, die der Mensch im Umgang mit unbekannten Situationen und dem Lösen von Problemen auf der wissensbasierten Ebene einsetzt ist auch hier ein Experimentieren mit dem System. Hypothesen werden formuliert und bewertet, Reaktionen des Systems gegen die Hypothesen getestet.

Einige dieser Tests können in ihrer Folge zu Systemfehlern führen. Schließt man derartige Experimente jedoch von vorneherein aus, so beraubt man den Menschen

zugleich der Möglichkeit aufkommende Probleme mit der ihm eigenen Kreativität und seiner Fähigkeit unbekannte Situationen zu handhaben zu bewältigen.

2.5. Von der Theorie zum System

Allen drei Verhaltensebenen ist gemein, dass dem Menschen zum Erlangen und zum Erhalt seiner Fähigkeiten auf einem bestimmten Level eine Möglichkeit gewährt werden muss mit dem System zu experimentieren und Erfahrungen zu sammeln. Je vertrauter ein Benutzer mit einem System wird, desto weiter sinken seine Fertigkeiten im Umgang damit von der zielorientierten wissensbasierten in Richtung der unbewussten fertigkeitenbasierten Ebene.

Kommt es schließlich zu einer unerwarteten Situation muss das System es dem Benutzer ermöglichen mittels dieser Experimente die notwendigen Informationen für alle drei Verhaltensebenen zu sammeln.

Die Notwendigkeit für diese Experimente führt zu einem neuen Designparadigma, dem der *fehlertoleranten Systeme*. Der Benutzer muss in die Lage versetzt werden seine Fehler jederzeit erkennen und rückgängig machen zu können. Hierzu ist es erforderlich, ihn zu jederzeit mit ausreichend Informationen über seine Handlungen zu versorgen. Potentiell gefährliche Operationen müssen in mehrere Schritte unterteilt werden um plötzliches, irreversibles Eintreten von Fehlersituationen zu vermeiden. Eine Fehlentscheidung darf nicht sofort zu einer Katastrophe führen.

In ihrem Buch *Safeware, System Safety and Computers* stellt Nancy Leveson zahlreiche Konzepte und Designempfehlungen für die Entwicklung fehlertoleranter Mensch-Maschine Schnittstellen vor [1].

3. Aufgabenmodelle

Beim Systementwurf werden die Anforderungen an Hard- und Software genauestens spezifiziert. Fragt man einen Entwickler jedoch nach den Aufgaben des Benutzers eines solchen Systems erntet man häufig nur ein Schulterzucken oder eine Antwort der Form „Der Benutzer tut was auch immer nötig ist damit das System seine Aufgabe erfüllen kann“.

Mit einer derart ungenauen Vorstellung von den Aufgaben eines Benutzers ist der Systementwurf bestenfalls schwierig. Im schlechtesten Fall ist das Ergebnis ein nicht vernünftig bedienbares und somit unter Sicherheitsaspekten bedenkliches System.

Entwickler und Benutzer verwenden beim Umgang mit dem System unterschiedliche gedankliche Modelle. Während Entwickler fast ausschließlich auf der wissensbasierten Ebene agieren verwendet der Benutzer Modelle aller drei Ebenen (siehe Abschnitt 2). Um diesem Umstand Rechnung zu tragen muss der Entwickler in der Lage sein die Handlungen der Anwender erklären und vorhersagen zu können. Zu diesem Zweck gibt es verschiedene Analysetechniken. Abschnitt 3.1 gibt einen kurzen Überblick über die Task-Analysis Methodik, Abschnitt 3.2 enthält eine detailliertere Beschreibung des Operator Function Model.

3.1. Task-Analysis

Eine Technik, die häufig eingesetzt wird, ist die Task-Analysis Methode. Diese, typischerweise statische Analysemethode, lässt sich grob folgendermaßen zusammenfassen: Finde und erstelle eine Liste mit allen vom Menschen durchgeführten Aktionen und ihrer Beziehung zu Systemaufgaben.

Ein Verantwortlicher sammelt zunächst alle zu einem bestimmten Prozess gehörenden Aufgaben. Dies kann entweder durch direkte Beobachtung oder durch Fragebögen und Interviews geschehen.

Anschließend werden die gesammelten Daten ausgewertet und aufbereitet und aus ihnen werden so genannte *job descriptions* erstellt. Diese job descriptions beschreiben die benötigten Fertigkeiten, Verantwortlichkeiten, Anforderungen und Arbeitsbedingungen der jeweiligen Aufgaben.

Anhand dieser Beschreibungen ist nun eine genaue Einschätzung der anfallenden Aufgaben sowie der damit verbundenen Risiken und Gefahren möglich.

Gerade bei komplexen Prozessen und Interaktionsszenarien ist der Ansatz der Task-Analysis Methode jedoch nur schwer umzusetzen. Die Größenordnung der durchzuführenden Erfassung und Auswertung sprengt schnell jeden Rahmen. Eine sinnvolle Durchführung ist dann nicht mehr möglich, obwohl die Methode an sich für einen eingearbeiteten Mitarbeiter umkompliziert ist [4].

3.2. Operator Function Model (OFM)

Mit Blick auf die Mängel der herkömmlichen Task Analysis wurde das Operator Function Model entwickelt. Es zeichnet sich durch eine dynamische und analytische Repräsentation der menschlichen Kontrollaktivitäten innerhalb eines Systems aus.

Die dynamische Repräsentation ermöglicht die Modellierung menschlicher Kontrollaktivitäten innerhalb wechselnder Systemzustände. Die analytische Repräsentation muss leicht in Software umzusetzen sein und anschließend eingesetzt werden um die Erfordernisse und Inhalte (*semantics*) des Mensch-Maschine-Interfaces zu charakterisieren [3].

Das OFM versucht in mathematischer Form darzustellen wie ein Benutzer ein komplexes System in einfachere Teile unterteilt und Kontrollaktionen und Systemkonfigurationen aufeinander abstimmt um eine annehmbare Systemleistung zu erreichen.

3.3. Struktur des Operator Function Model

Das OFM besteht aus einem Netzwerk von endlichen Automaten. Grundlage seiner Struktur ist ein hierarchisch/heterarchisches (nebengeordnetes) Schema (siehe Abbildung 4-3).

Die Hierarchie des Modells erlaubt es, Komplexität überschaubar zu machen indem es Knoten auf einer höheren Ebene als ursächliche Auslöser für die Aktivitäten auf niedrigeren Ebenen spezifiziert. Knoten auf der höchsten Ebene repräsentieren die wichtigsten Benutzerfunktionen (*operator functions*); Knoten auf der niedrigsten Ebene stehen für Benutzeraktionen (*operator actions*).

Jede Funktion (*function*) wird in eine Reihe von Unterfunktionen (*subfunctions*) unterteilt. Diese wiederum bestehen aus einer Reihe von Aufgaben (*tasks*), welche durch eine oder mehrere Aktionen (*actions*) erfüllt werden. Die genaue Anzahl von Unterteilungsebenen ist abhängig von der jeweiligen Anwendung. Es existieren sowohl Fälle in denen eine Unterfunktionsebene nicht notwendig ist, als auch solche in denen eine ganze Hierarchie von Unterfunktionsebenen vorkommt.

Mit der Heterarchie des Modells wird der nebenläufigen Natur überwachender Kontrolle Rechnung getragen: Zu jedem Zeitpunkt kann der Benutzer mehrere Aktionen aufeinander abstimmen.

Kanten auf derselben Ebene des Modells stehen für vom System ausgelöste Aktionen oder den erfolgreichen Abschluss von Benutzeraktivitäten. Diese Kanten können entweder als Vorbedingungen oder als auslösende Ereignisse betrachtet wer-

den. Sie können auch den erwarteten zeitlichen Ablauf von Kontrollaktivitäten darstellen.

Die Zustandsübergänge in den endlichen Automaten sind nicht notwendigerweise deterministisch. Es existiert möglicherweise eine ganze Reihe von möglichen Folgezuständen. Durch diese nicht-deterministischen Übergänge wird die Flexibilität des Benutzers bei anstehenden Entscheidungen ausgedrückt. Will der Benutzer Aktionen durchführen, die nicht in der Liste zulässigen Folgeknoten enthalten sind, ist dies möglicherweise ein Zeichen für einen Benutzerfehler.

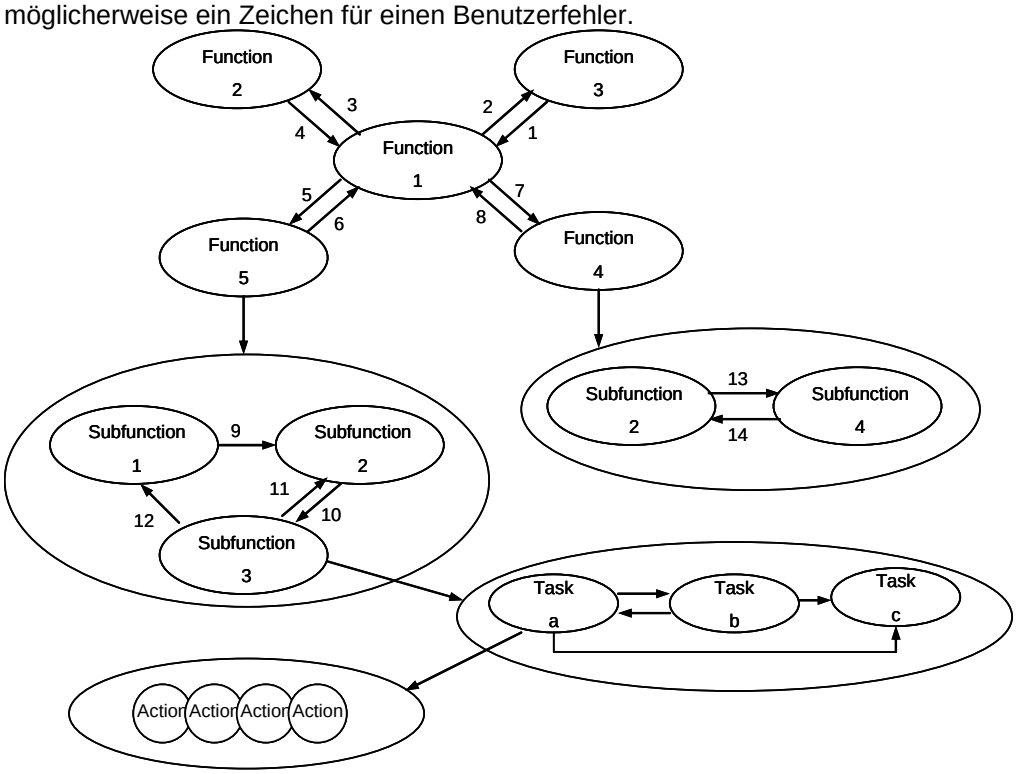


Abbildung 4-3. Struktur des Operator Function Model

3.4. Erstellung eines Operator Function Model

Die Erstellung eines OFM ist normalerweise ein iterativer Prozess, der in Abhängigkeit vom gewünschten Ziel *bottom-up* oder *top-down* durchgeführt werden kann.

Bei der bottom-up Vorgehensweise basiert die Struktur des Modells häufig auf der Konfiguration von Kontrollelementen (Schaltern, Anzeigen etc.). Diese bilden mit den zugehörigen Benutzeraktionen die Knoten auf der untersten Modellebene. Diese Knoten werden dann so lange in Gruppen zusammengefasst, bis eine zum untersuchten Prozess passende Menge von Knoten auf der höchsten Modellebene gefunden ist. Dieses Vorgehen wird häufig gewählt, um zu erklären warum ein Benutzer Probleme auf eine bestimmte Art und Weise angeht.

Der top-down Ansatz verfolgt ein anderes Ziel: Er soll bestimmte Verhaltensweisen vorschreiben. Dazu wird auf der obersten Ebene eine Reihe von Benutzerfunktionen (*operator control functions*) spezifiziert, diese Funktionen werden anschließend in die einzelnen Bestandteile zerlegt. Danach kommt manchmal eine *meet-in-the-middle* Vorgehensweise zum Einsatz, bei der die untersten Ebenen (zumeist Eingaben über ein Computer-Interface) und eine weitergehende Verfeinerung der höheren Schichten abwechselnd stattfinden.

3.5. Benutzeraktionen

Die Knoten der untersten Ebene des Operator Function Model bestehen aus den Benutzeraktionen (*operator actions*). Diese können im OFM zweierlei Formen haben: *manuelle* und *kognitive*.

Ein Beispiel für manuelle Aktionen wäre etwa: „stelle Schalter A auf Position 1“ oder auch „ersetze Komponente A durch eine verfügbare Komponente derselben Klassen“. Kognitive Aktionen stehen für Benutzeraktionen wie Monitoring und Situationsbeurteilung, z.B. „beurteile den Systemzustand und stelle fest ob Komponenten A korrekt funktioniert“.

Diese kognitiven Aktionen machen einen wichtigen Teil der Rolle des Benutzers als Systemüberwacher aus. Seine Aufgabe besteht häufig darin, einen selbst regelnden automatisierten Prozess zu überwachen. Diese Überwachung besteht jedoch nicht darin, nichts zu tun, sondern erfordert in bestimmten Intervallen eine Überprüfung der angezeigten Informationen auf mögliche Fehlerzustände. Im Gegensatz zu manuellen Aktionen haben kognitive nicht notwendigerweise Einfluss auf den Systemzustand oder die Konfiguration der Kontrollelemente.

3.6. Verbindung zwischen Informationen und Aktionen

Zu den kognitiven Aktionen zählen auch das Sammeln von Informationen, deren Verarbeitung und das Fällen von Entscheidungen. Zusätzlich zu den manuellen und kognitiven Aktionen verbindet das Operator Function Model häufig auch eine Reihe von Informationen, die der Benutzer in einer bestimmten Situation benötigt, mit den kognitiven Knoten des Modells.

Diese Verbindung zwischen kognitiven und manuellen Aktionen in Verbindung mit den benötigten Informationen ist ein wichtiges Merkmal des Operator Function Model. Die Modellierung sowohl der kognitiven als auch der manuellen Aktionen erlaubt eine sehr genaue Beschreibung der Interaktionen des Benutzers mit dem System. Nutzt man das Modell zur Vorhersage können auf diese Weise die vom Benutzer benötigten Informationen identifiziert werden. Wird es zur Beschreibung eingesetzt kann so erklärt werden warum bestimmte Informationen vom Benutzer angefordert wurden. Kommt es auf schlussfolgernde Weise zum Einsatz kann es Benutzeraktionen, oder auch das Fehlen einer scheinbar notwendigen Aktion, erklären.

Eine ähnliche Verbindung wie sie zwischen benötigten Informationen und den kognitiven Aktionsknoten besteht, existiert auch zwischen Benutzeranweisungen (*operator commands*) und manuellen Benutzeraktionen. Das OFM verwendet eine semantische Interpretation von Aktionen auf der Ebene der Aktionsknoten. Anweisungsknoten (*command nodes*) spezifizieren die Syntax die nötig ist um eine bestimmte Aktion durchzuführen.

3.7. Zusammenfassung Operator Function Model

Das Operator Function Model stellt mit seiner fein granularen Darstellung der Benutzeraktivitäten und deren Interaktion mit dem System für Entwickler ein wichtiges Mittel zum Verständnis des menschlichen Teils eines Systems bereit. Diese Darstellung ist ebenso wohl definiert und formal wie es typische Beschreibungen für die Hard- und Software eines Systems sind.

Mit Hilfe des OFM ist es möglich, kritische Fragen zur Systemautomatisierung zu beantworten:

- Aus welchen Aktivitäten setzt sich eine Benutzerfunktion zusammen?
- Wie wird diese Funktion ausgeführt?
- Welche Alternativen hat ein Benutzer um den gewünschten Systemzustand herbeizuführen?
- Bei welchen Aktionen handelt es sich in einem bestimmten Kontext höchstwahrscheinlich und einen Benutzerfehler?
- Ist der Benutzer überfordert / unterfordert?
- Mit welchen Auswirkungen ist zu rechnen, wenn eine bestimmte Unterfunktion automatisiert wird?

Benutzerfehler können mit den Mitteln des Operator Function Model genauestens untersucht werden. Mit einem präzisen Verständnis dafür, was ein Benutzer tun sollte und wann und wie er es tun sollte ist es möglich unerwartete oder falsche Aktionen zu analysieren um festzustellen ob es sich dabei tatsächlich um einen zufälligen Fehler des Benutzers handelt oder ob es nicht vielmehr eine logische Folge einer schlecht entworfenen Mensch-Maschine-Schnittstelle ist.

Des Weiteren bietet das OFM eine Unterstützung bei der Entwicklung ‚intelligenter' Systeme. Das bedeutet, dass das System erkennen kann, was der Benutzer tun möchte und ihn bei der Erreichung seiner Ziele unterstützt.

Durch seine flexible Struktur ist das Operator Function Model in einer Vielzahl von Kontexten einsetzbar und bietet die Möglichkeit zur Anpassung an verschiedenste Entwicklungsprozesse [5], [6], [7].

4. Schlusswort

Mit Rasmussens 3-Ebenen Modell steht ein theoretisches Modell für das Verständnis menschlichen Handelns zur Verfügung. Das Operator Function Model bietet eine formalisierte Methode zur Vorhersage und Beschreibung der Interaktionen des Benutzers mit dem System.

Beide stellen in gewissem Umfang das Rüstzeug bereit, das Entwickler zum Entwurf qualitativ hochwertiger Mensch-Maschine-Schnittstellen benötigen. Mit einem besseren Verständnis dafür, wie Menschen in einer gegebenen Situation handeln, welche Informationen sie zur Entscheidungsfindung benötigen und wie diese Informationen aufbereitet und dargestellt werden müssen, wird der Weg zu fehlertoleranten Kontrollsystemen und intelligenten Unterstützungssystemen bereitet.

Der Mensch spielt auch im Zeitalter hoch automatisierter Systeme weiterhin eine zentrale Rolle. Ohne ihn ist es diesen Systemen fast unmöglich, ihre Aufgaben zu erfüllen. Um ihren sicheren Betrieb zu ermöglichen muss den menschlichen Fähigkeiten und Einschränkungen Rechnung getragen werden.

Ebendies ist das Ziel der weltweiten Anstrengungen der Forschungsdisziplin der Human Factors.

Referenzen

[1] N. Leveson, *Safeware: System Safety and Computers*. Boston: Addision-Wesley, 1995.
[2] J. Rasmussen, "What Can Be Learned from Human Error Reports?" in *Changes in Working Live*, K. Duncan et al. New York: John Wiley & Sons, 1980.

[3] C. Mitchell. (2003, February 28). [Online]. Task-Analytic Models of Human Operators: Designing Operator-Machine Interaction. Available: http://www.isye.gatech.edu/~cm/papers/task-analytic_design.html

[4] Stephans et al. *System Safety Analysis Handbook, 2nd Edition*. New York: System Safety Society, 1997

[5] Cohen, S. M., Govindaraj, T., and Mitchell, C.M. "Analysis and aiding the human operator in electronics assembly" in *Design for Manufacturability: A Systems Approach to Concurrent Engineering and Ergonomics*, M. Helander and M. Nagamachi (Eds.). London: Taylor & Francis, pp.361-376, 1992.

[6] Lee, J. and Sanquist, T. "A systematic evaluation of technological innovation: A case study of ship navigation". *Proceedings of the 1993 IEEE International Conference on Systems, Man, and Cybernetics, Vol. 1*, pp.102-107, Le Touquet, France, 1993.

[7] Mitchell, C. M. and Saisi, D. S. "Use of model-based qualitative icons and adaptive windows in workstations for supervisory control systems". *IEEE Transactions Systems, Man, Cybernetics*. 17(4), pp.594-607, 1987.

Kapitel 5. Software ist anders – Warum Risikomanagement eine Herausforderung für das Software Engineering ist

Florian Munz

Universität Kaiserslautern
f_munz@informatik.uni-kl.de

Kurzfassung. Risikomanagement ist in traditionellen Ingenieursdisziplinen ein verbreiteter und akzeptierter Teil des Projekt-Managements und auch verantwortlich für die Produkt-Qualität. Im Software Engineering setzt sich das Risikomanagement dagegen erst langsam durch und zielt bisher hauptsächlich auf Projekt-Risiken wie Budget und Zeit. Gerade aber bei sicherheitskritischen Systemen – in denen Software eine immer größere Rolle spielt – muss das Risikomanagement mehr leisten und auch die Produkt-Qualität beeinflussende Aspekte betrachten. Diese Ausarbeitung stellt dar, inwiefern Software anders ist und welche Herausforderungen deswegen für Risikomanagement von Software-Projekten entstehen. Des Weiteren werden die Risikomanagement-Methoden des Software Engineerings dargestellt und die Möglichkeiten der Übertragung von schon existierenden Risikomanagement-Methoden der traditionellen Disziplinen untersucht.

1. Einleitung

Die Frage wie man das Software Engineering verbessern kann stellt sich sehr schnell, wenn man sich den hohen Anteil an fehlgeschlagenen Software-Projekten vor Augen führt. Ein wesentlicher Ansatz zur Verbesserung ist das Risikomanagement. Während es in traditionellen Ingenieurdisziplinen schon etabliert ist, hat sich die Erkenntnis, dass Risikomanagement ein wesentlicher und notwendiger Aspekt im Software Engineering ist, noch nicht allgemein durchgesetzt.

Die wenigen existierenden Risikomanagement Ansätze für das Software Engineering, z.B. vom SEI [12], fokussieren auf die Risiken des Softwareprojekts an sich: Endtermine zu halten und im finanziellen Budget zu bleiben. Da sich Software aber z.B. im Embedded-Systems-Umfeld immer mehr und weitestgehend unbemerkt in weite Teile unseres Alltags ausbreitet, müssen Entwicklungsmethoden gefunden werden, die in Garantien für die Qualität der erstellten Software münden. Risikomanagement muss daher über die nahe liegenden Projektaspekte hinausgehen und vor allem die Folgen des Softwareeinsatzes betrachten (z. B. im Umfeld der Produkthaftung oder beim Einsatz in sicherheitskritischen Bereichen).

Da das Software Engineering eine noch relativ junge Disziplin ist, stellt sich die Frage, ob man nicht bezüglich des Risikomanagements von der Vorgehensweise und den Konzepten etablierter Ingenieurdisziplinen lernen kann. Ob es dort Wissen und Methodik gibt, die man ganz oder teilweise auf Softwareprojekte und Softwareprodukte übertragen kann oder ob für Softwaresysteme andere Gegebenheiten gelten, soll hier untersucht werden.

Diese Ausarbeitung ist in die folgenden Abschnitte aufgeteilt. Abschnitt 2 beschreibt, warum Software anders ist. In Abschnitt 3 werden die Risikomanagement-Methoden des traditionellen Ingenieurwesen dargestellt und in Abschnitt 4 die existierenden Methoden des Software Engineerings. Im Abschnitt 5 wird der Frage nachgegangen, ob die traditionellen Disziplinen von den Risikomanagement-Methoden des

Software Engineerings etwas lernen können. Schließlich folgt eine Zusammenfassung in Abschnitt 6.

2. Software ist anders

Ob und wodurch sich Softwaresysteme von anderen mit Ingenieurswissen gebauten Systemen unterscheiden ist natürlich eine wesentliche Frage für den Ansatz, die Methoden anderer Disziplinen zu übertragen.

Die intuitive Annahme, dass neuere Ingenieurdisziplinen das Wissen und die Techniken etablierter und reiferer Disziplinen übernehmen, gilt natürlich auch für das Software Engineering. Auch wenn die Erstellung von Softwaresystemen erst seit wenigen Jahren als Ingenieurdisziplin begriffen wird, ist die Übernahme existierender Techniken zum Risikomanagement z. B. im Bereich sicherheitskritischer Systeme nahe liegend.

Warum aber, hat sich diese Idee noch nicht stärker durchgesetzt? Warum lassen sich bekannte Techniken nicht so ohne weiteres übernehmen? Sind Softwaresysteme wirklich anders oder ist das nur eine Ausrede von gescheiterten Softwareprojektleitern? Diese Fragestellung ist in der jungen Geschichte der Informatik und insbesondere des Software Engineerings schon häufig diskutiert worden. Im Folgenden werden die wichtigsten Argumente vorgestellt und diskutiert.

Brooks teilt in [1] die Probleme und Eigenschaften von Software in zwei Kategorien ein. Zum einen gibt es Eigenschaften der Software, die zum Wesen der Software gehören. Dies sind inhärente Schwierigkeiten, die auch mit besseren Methoden nicht behoben werden können. Zum anderen gibt es Eigenschaften von Software, die als temporäre Probleme zu verstehen sind. D.h., sie treten eine Zeit lang auf, können aber mit den richtigen Methoden behoben werden. Sie sind also für Software nicht inhärent.

Auch wenn sich irgendwann alle temporären Probleme durch geeignete Methoden behoben sind, das Wesen der Software bleibt und es wird laut Brooks kein Silver Bullet geben, das die inhärenten Schwierigkeiten von Software lösen kann. In der Vergangenheit haben z.B. High-Level-Programmiersprachen und die Objekt-Orientierte Programmierung die Schwierigkeiten in einigen Phasen des Software-Entwicklungsprozesses reduziert. Diese besseren Werkzeuge nutzen wir jetzt aber in Umgebungen und Einsatzgebieten deren Komplexität und Anforderungen wesentlich stärker gestiegen sind als die Qualität unserer Methoden und Werkzeuge. Brooks behauptet daher:

“I believe the hard part of building software to be the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the rep-resentation. We still make syntax errors, to be sure; but they are fuzz compared with the conceptual errors in most systems.”

2.1. Das Wesen von Software

Brooks beschreibt in [1] vier inhärente Attribute von Software. Aufgrund vielschichtiger Bedeutungen und der deshalb schwierigen Übersetzung sind die Attribute im Folgenden im englischen Original:

Kapitel 5: Software ist anders –
Warum Risikomanagement eine Herausforderung für das Software Engineering ist

Complexity. Große Softwaresysteme sind momentan eines der komplexesten menschlichen Konstrukte überhaupt. In anderen komplexen Gebilden, wie z.B. Gebäuden und Autos, liegt ein Teil der Komplexität in der physischen Wiederverwendung identischer Teile. Bei Software dagegen werden identische Teile zu einem Modul zusammengefasst, so dass die Größe und Komplexität eines Programms nicht in der Wiederverwendung identischer Komponenten begründet ist, sondern durch das Zusammenwirken vieler unterschiedlicher Teile entsteht.

Schon die Computer-Hardware ist mit ihrer hohen Anzahl an unterschiedlichen internen Zuständen sehr komplex. Softwaresysteme hingegen liegen von der Größe ihres Zustandsraums her nochmals mehrere Größenordnungen darüber, was das Entwerfen, Beschreiben und Testen so schwer macht.

Nach Brooks ist diese Komplexität im Wesen von Software enthalten. Im Gegensatz zur Mathematik oder Physik, wo komplexe Phänomene durch einfachere und abstraktere Modelle erklärt und bewiesen werden können, gibt es (noch) keine Möglichkeit von der Komplexität in Software zu abstrahieren. Newtons Gesetze erklären auf simple Weise einen sehr komplexen Sachverhalt, aber nur weil alles Unwesentliche weggelassen wurde. Der Software ist aber eine Komplexität inhärent, die man nur eingeschränkt geeignet abstrahieren kann. Abstrahiert man die Komplexität komplett weg, so verschwindet das Wesentliche von Software.

Beizer hofft in [2], dass in der Zukunft eine Gesetzmäßigkeit gefunden werden kann, mit dem man die Auswirkungen dieser Komplexität berechnen und daraus resultierende „Safety-Limits“ bestimmen kann. Bis allerdings solch ein Gesetz gefunden wird, gibt es nur die Empfehlung, behutsam mit der Komplexität umzugehen. Konservatives Software Engineering bedeutet sich selbst Grenzen vorzugeben. Auch wenn es so scheint, als wäre Software einfach und leicht zu ändern, darf man es gerade in komplexen Systemen nur sehr vorsichtig und durch umfassende Regressionstests kontrolliert machen. Wenn man die Besonderheiten von Software akzeptiert, sollte man Methoden und Regeln verwenden, die auf ihr Wesen Rücksicht nehmen.

Zusätzlich zu den Problemen, die aus der internen technischen Komplexität resultieren, sind bei einem erfolgreichen Projekt auch noch organisatorische und zwischenmenschliche Probleme zu lösen. Für einen Projekterfolg müssen alle Dimensionen gleich gut beherrscht werden.

Conformity. Nun sind Informatiker natürlich nicht die Einzigen, die sich mit komplexen Fragestellungen beschäftigen. Physiker müssen schon auf fundamentaler Ebene mit sehr komplexen Objekten arbeiten, sie haben aber den Glauben allgemeingültige und vereinigende Prinzipien zu finden. Brooks sagt: "Einstein argued that there must be simplified explanations of nature, because God is not capricious or arbitrary. No such faith comforts the software engineer."

Denn die zu handhabende Komplexität ist kein Design der Natur, sondern eine Folge von vielen Anpassungen an unterschiedlichste System-Interfaces und menschliche Schnittstellen, nach der die Software sich richten soll. Die Änderungen sind selten konsistent, da sie von unterschiedlichen Personen zu unterschiedlichen Zeiten vorgenommen werden. Des Weiteren muss sich Software häufig anpassen, weil es die letzte Komponente ist die hinzugefügt wird. In anderen Fällen wird angepasst, weil es leicher scheint die Software zu verändern, als eine bestehende Hardware-Komponente.

Changeability. An Software möchte man wie bei jedem anderen Produkt auch – z.B. Computer oder Autos – nach Inbetriebnahme Änderungen vornehmen. Tatsache ist aber, dass Autos nur selten nach Inbetriebnahme geändert werden, eine Änderung in der Software dagegen schnell durchgeführt ist. Es scheint, dass Software sehr einfach zu ändern ist.

Erfolgreich eingesetzte Software unterliegt einem dauernden Änderungsprozess. Entweder weil sich die Anforderungen oder die Umgebung ändert, oder weil die Software von den Benutzern in neuen Szenarien eingesetzt wird.

Invisibility. Software ist abstrakt und nur in Teilaspekten visualisierbar. Der Grundriss eines Gebäudes lässt sich dagegen mit Hilfe von geometrischen Abstraktionen visualisieren und Architekten und dem Kunden bekommen ein Gefühl und Eindruck von dem Objekt. Mechanische Teile werden mit den technischen Zeichnungen des Maschinenbaus präzise beschrieben.

Um eine geometrische Abstraktion zu bekommen, müssen geometrische Realitäten vorliegen. Da Software nicht räumlich ist, gibt es keine nahe liegende Repräsentation. Die vielen verschiedenen Diagramm-Typen, die es in der Software gibt stellen immer nur einen Teilaspekt dar, sei es der Kontrollfluss, der Datenfluss, die Verwendungsabhängigkeiten oder die Interaktionen zwischen Komponenten und leisten keine vollständige oder umfassende Visualisierung von Software.

2.2. Vergleich von Software mit anderen Produkten

Mit dem Wissen über das Wesen von Software lassen sich die Eigenschaften von Softwaresystemen mit denen von anderen Produkten vergleichen. Anhand der klassischen Beispiele wie Autos, Gebäude und Computer-Hardware hat Beizer in [2] die Unterschiede von Software betont. Neben dem Lokalitäts-Prinzip und der Dekomposition, die im Folgenden noch genauer beschrieben werden, hat Beizer – ähnlich wie Brooks – die Komplexität als ein wesentliches Unterscheidungsmerkmal herangezogen. Des Weiteren betont er die Probleme synergetischer Bugs, womit er das Fehlen eines eindeutigen Schuldigen meint. Fehler, die nur durch die unvorhersagbare und unvorhersehbare Interaktion der ansonsten korrekten scheinenden Komponenten bei ihrer Integration entstehen, machen es schwer, eine einzelne Fehlerursache zu finden und irgend jemand als Person oder irgendetwas als Softwarekomponente die Schuld zu geben. Fehlende oder unzureichende allgemeine Qualitätsmaße für Software und die Schwierigkeit der Quantifizierbarkeit von Softwareeigenschaften sind weitere Punkte die Beizer anführt.

Lokalitäts-Prinzip. In der physischen Welt liegen der Auslöser und die Auswirkung einer Aktion zeitlich und räumlich sehr nahe bei einander. Beizer nennt dies das Lokalitäts-Prinzip. In der Welt der Software liegt der Auslöser, z.B. eine Fehlerursache, und seine Auswirkungen zeitlich und räumlich beliebig weit voneinander entfernt. Hinzu kommt, dass jeder noch so kleine Fehler in einem komplexen System große Auswirkungen an einer ganz anderen Stelle haben kann. Je komplexer die Software wird, desto gravierender wird dieses Lokalitäts-Prinzip verletzt.

Wenn man in seinem Auto Probleme mit dem Scheibenwischer hat, erwartet man nicht, dass das Probleme mit dem Kofferraumschloss verursachen könnte. Diese in der physischen Welt gefunden Raum-Lokalität, also das Auftreten der Fehler nahe zu ihrem Entstehungsort trifft in der Software höchstens auf ganz einfache Fehler zu. Da diese aber in den meisten Fällen entweder direkt vom Compiler oder in den ersten Test-Phasen gefunden werden, sind es komplexere Fehlersituationen, die der Benut-

zer zu spüren bekommt. Aber auch wenn Konzepte wie die Objekt-Orientierte Programmierung dabei helfen, das Lokalitätsproblem zu verringern, eine vollständige räumliche Lokalität bezüglich der Fehlerbilder wird damit nicht erreicht. Um Software und die Probleme der Software-Qualität verstehen zu können muss man also akzeptieren, dass Fehler beliebig weit entfernt von ihrem Entstehungsort auftreten können.

Das gleiche gilt für die Zeit-Lokalität. Bei einem explodierenden Reifen bricht das Auto immer sofort aus. Das Auftreten eines sichtbaren Fehlverhaltens direkt zum Zeitpunkt der Fehlerursache kann man bei Software allerdings nicht voraussetzen. Der Zeitpunkt des Auftretens hängt häufig von der spezifischen Aktions-Reihenfolge des Benutzers ab. Ein Fehler kann mit einem hohen Zeitverzug erst spätere Verwender beeinträchtigen (Caching, Persistenz). Kein Entwickler kann die Kombinatorik aller möglichen Nutzungsszenarien vorhersehen und durch vollständige Tests abdecken. In Extremfällen mit hoher Last (= Ressourcenmangel) und hoher Nebenläufigkeit kommt noch eine Vielzahl von neuen Problemen mit sporadischer Natur dazu.

In der physischen Welt ist man gewöhnt, dass Gegenstände allmählich verschleißen. Ein Messer z.B. wird nicht von einem Moment auf den anderen unscharf. Das „Fehlverhalten“ des Messers kündigt sich über einen Zeitraum hinweg an. In diesem analogen Verständnis der Dinge geht man im Allgemeinen auch von proportionalen Reaktionen auf Aktionen aus. Diese Proportionalität findet man bei Software nicht. Ein Software-Bug hat keine „angemessen“ Folgen: selbst das kleinste Fehlverhalten kann ohne Ankündigung katastrophale Auswirkungen haben.

Komposition und Dekomposition. Das Kompositions-Prinzip der Ingenieurwissenschaften besagt, dass man ausgehend von den Eigenschaften einer einzelnen Komponente mit den geeigneten Regeln die Eigenschaften des gesamten Systems berechnen kann. So wird z.B. die Belastbarkeit einer Brücke aus der Belastbarkeit der Pfeiler und Träger berechnet.

Sollte so ein Prinzip auch für Software gelten, so ist es leider noch nicht gefunden worden. Die einzige Möglichkeit die Zuverlässigkeit eines Softwaresystems zu bestimmen, ist es im laufenden Betrieb zu überwachen und zu vermessen. Selbst wenn wir es schaffen sollten, ein fehlerfreies Softwaresystem zu schreiben, existiert keine Möglichkeit dies zu bestätigen. Hierbei gibt es auch eine Erwartungshaltung bei den Anwendern eines Softwaresystems, die nur schwer zu erfüllen ist. Auf der einen Seite wird immer mehr Funktionalität und Verfeinerung nachgefragt und auf der anderen Seite wird erwartet, dass die nun komplexerer gewordenen Software noch genauso zuverlässig und schnell wie die vorherige, einfachere Version arbeitet, wenn nicht noch verlässlicher und schneller.

Beizer schließt daraus: *“Composability, which is fundamental to traditional engineering dis-ciplines, cannot be assumed for software”.*

Die Dekomposition oder auch „Divide & Conquer“ genannt kann sicherlich als eines der Schlüsselkonzepte der Informatik angesehen werden, um Komplexität beherrschbar zu machen. Es gibt aber in der Informatik keine aus den traditionellen Ingenieurwissenschaften übernommene formale Methode dafür. Die Software Ingenieure setzten in der Vergangenheit diese Vorgehensweise nicht anhand einer formalen oder mathematischen Methodik ein, sondern nur mittels Heuristiken und durch Erfahrung.

Das größte Problem der Dekomposition in der Informatik ist aber, das wir nicht davon ausgehen können, dass zwei korrekte einzelne Komponenten auch im Zusammenspiel korrekt sind. Auch umgekehrt können zwei in einem funktionierenden System zusammengesetzte Komponenten einzeln dennoch fehlerhaft sein, weil sie zufällig gegenseitig ihre Fehler kompensieren.

3. Ansätze traditioneller technischer Systeme

In jeder neuen Disziplin ist es üblich, von den Methoden und Prinzipien der schon existierenden Disziplinen zu lernen. Die Entwicklung einer Wissenschaft vollzieht sich dabei in Synthese- und Analyse-Phasen. Am Anfang überwiegt die Synthese, es geht nur darum etwas herzustellen bzw. zu erzeugen. Erst wenn der Erzeugungsprozess einigermaßen stabil ist, beginnt man mit der Analyse. Es werden die zugrunde liegenden Prinzipien, Gesetzmäßigkeiten und Mechanismen entdeckt. Mit diesem Wissen beginnt eine weitere Synthese-Phase in der mit dem jetzt erworbenen Wissen der Erzeugungsprozess automatisiert und häufig industrialisiert wird. Diese beiden Phasen wechseln sich ab und treiben eine Ingenieursdisziplin immer weiter.

In der Softwareentwicklung, die nun gerade mal 50 Jahre alt ist, überwog nach Beizer bisher die Synthese-Phase. Nur langsam beginnt man den Software-Erstellungsprozess zu analysieren. Genauso wie Anfang des 18. Jahrhunderts das Einstürzen vieler Stahl-Brücken zwangsläufig die Statik-Berechnung voran gebracht hat, scheiterten Ende der 60er Jahre die ersten großen Software-Projekte und machten deutlich, dass eine systematische Softwareentwicklung Software Engineering zwingend erforderlich ist. Die weiter zunehmende Komplexität, Größe und Kritikalität heutiger Softwaresysteme hat das Bewusstsein dafür nur noch verschärft.

Am Beispiel der existierenden Risikomanagement-Ansätze in traditionellen Disziplinen wird nun die Frage erläutert, inwieweit bereits bekannte Ansätze auf die Informatik übertragen werden können.

Der erste Unterschied wird schon im Fokus deutlich. Wenn Software-Ingenieure von Risikomanagement sprechen, verstehen sie darunter einen Ansatz um den Entwicklungsprozess unter Kontrolle zu halten. Da lange Zeit Kostenüberschreitungen und nicht eingehaltene Termine das größte und offensichtlichste Problem bei der Erstellung von Softwaresystemen waren, beschäftigt sich Risikomanagement im Software Engineering hauptsächlich mit Kosten- und Zeitrisiken. Es liegt sicherlich zu einem Großteil an der historischen Entwicklung – betriebliche Informationssysteme waren der erste Bereich in dem große Softwaresysteme breit eingesetzt wurden – dass sicherheitskritische Aspekte oder generell die Produktqualität beim Risikomanagement eine eher untergeordnete Rolle spielt.

In den traditionellen Ingenieurwissenschaften wurden mittlerweile viele Risikomanagement-Ansätze entwickelt, in deren Fokus Produktrisiken stehen. D.h. Risiken, die vom Produkt ausgehen und den Benutzer oder die Umwelt in Gefahr bringen können. Die Ansätze kann man in zwei Kategorien unterteilen: Strukturorientierte und problemorientierte Methoden.

Strukturorientierte Methoden gehen von einer Zerlegung (Dekomposition) des Systems aus und aggregieren die Eigenschaften einzelner Systemkomponenten zu einer Gesamtaussage. Diese Methoden lassen sich nicht ohne weiteres auf Softwaresysteme übertragen. Das Fehlen eines strikten Dekompositionsprinzips in der Informatik – insbesondere das Fehlen formaler Methoden und die geringe Beherrschung des komplexen Zusammenwirkens von Systemkomponenten – verhindern, dass man von der Qualität einer einzelnen Komponente auf die Gesamtqualität des Systems schließen kann. Aus diesem Grund lassen sich Risiken nicht bottom-up – aus dem System zum Fehler – aus den einzelnen Komponenten für das Gesamtsystem hochrechnen.

Problemorientierte Methoden sind aufgrund ihres allgemeineren Ansatzes eine genauere Betrachtung wert. Ausgehend von Domänenwissen werden Risiken und Gefahren abgeleitet. Dieser Top-Down-Ansatz erfordert spezielles Wissen über die Domäne, die der Software-Entwickler selbst nicht besitzt. Diese Methoden erfordern deswegen eine enge Zusammenarbeit mit dem Anwendungs-Experten. Das spezifische Domänenwissen des Anwenders muss mit dem Systemwissen des Software-Ingenieurs zusammengebracht werden und von dieser Kommunikation hängt die Güte des Risikomanagements ab. Insofern sind problemorientierte Methoden breiter

anwendbar, gleichzeitig aufgrund ihrer Allgemeinheit aber auch schwieriger einzusetzen.

Es zeigt sich, dass die Methoden des traditionellen Ingenieurswesens nicht ohne weiteres eins-zu-eins übernommen werden können. Zu spezifisch sind die Ansätze und die Voraussetzungen berücksichtigen die Eigenheiten von Software nicht. Nichtsdestotrotz sind natürlich die zugrunde liegenden Ideen und Ansätze der Methoden sehr wohl übertragbar. Der Schlüssel zum erfolgreichen Risikomanagement liegt demnach in der sinnvollen Kombination der Methoden – das vorhandene Wissen der traditionellen Disziplinen ist zu wertvoll um es nicht zu berücksichtigen. Eine spezielle Anpassung unter Berücksichtigung der Eigenheiten von Software ist aber auf jeden Fall nötig.

Die Übertragung gut entwickelter Methoden aus traditionellen Ingenieurdisziplinen ist möglich, in Teilbereichen und speziellen Fällen sogar angemessen und wäre durchaus in der Lage einen echten Verbesserungsbeitrag zu liefern.

4. Risikomanagement-Methoden des Software Engineering

Als Beispiele für existierende Methoden im Software Engineering werden nachfolgend zwei Ansätze vorgestellt.

4.1. Risikomanagement nach Boehm

Barry Boehm hat als einer der ersten das Risikomanagement in das Bewusstsein des Software Engineerings gebracht und seine bis dahin unterschätzte Wichtigkeit betont. Die Methodik von Boehm [3], die den Schwerpunkt auf Kosten und Zeit legt, ist heute der Grundstein der meisten Risikomanagement-Methoden im Software Engineering. Da die ersten größeren Softwaresysteme meist betriebliche Informationssysteme waren, sind die Risikomanagement-Methoden auch auf sie ausgelegt. Sie betrachten daher weniger die Risiken, die von der Software ausgehen als vielmehr die Risiken, durch die eine Software-Entwicklung beeinträchtigt werden kann. Im Wesentlichen sind das Risiken, die höhere Kosten oder eine längere Projekt-Laufzeit zur Folge haben.

Risikomanagement besteht für Boehm aus zwei übergeordneten Tätigkeitsblöcken Risiko-Bewertung und Risiko-Kontrolle, die wiederum in untergeordnete Tätigkeiten unterteilt sind

Risiko-Bewertung. Die Einschätzung der Risiken umfasst drei Schritte: In der Risiko-Identifikation werden mögliche Risiken aufgezählt. Verschiedene Techniken, wie Checklisten oder eine Analyse der Annahmen, helfen dabei Risiken zu identifizieren, die am wahrscheinlichsten den Projektverlauf gefährden können. Im zweiten Schritt werden diese Risiken analysiert und ihre Wahrscheinlichkeit sowie Auswirkung des Schadens jedes einzelnen Risikopunkts geschätzt. Techniken wie Performanz- oder Kosten-Modelle kommen hierbei zum Einsatz. Die Priorisierung der Risiken erfolgt im letzten Schritt der Risiko-Bewertung. Mittels einer abschließenden Gesamtanalyse der Bedrohungen werden dann alle Risiken in eine Reihenfolge entsprechend ihrer Signifikanz gebracht.

Risiko-Kontrolle. Sind die Risiken identifiziert und analysiert hört das Risikomanagement nicht auf, sondern muss auch im kompletten Projektverlauf verfolgt werden. Dafür sind nach Boehm drei Schritte vorgesehen:

Um nicht überraschend in ein Problem zu laufen, wird in einer Risiko-Management-Planung jedes potentielle Risiko aufbereitet und die Risiken miteinan-

der und bezüglich des gesamten Projektplans eingeschätzt. Im zweiten Schritt – der Risiko-Auflösung – wird für jedes Risiko eine Lösung gefunden, in der das Risiko vermieden oder anderweitig umgangen wird. Der dritte Schritt beinhaltet nun das Beobachten der Risiken im Projektverlauf und ein eventuelles Eingreifen bei ihrem Auftreten. Die so genannte Risiko-Überwachung beinhaltet z.B. ein Milestone-bezogenes Risiko-Tracking oder eine Top-10-Risiko-Liste.

4.2. Riskit

Die Riskit-Methode hebt sich durch ihre solide theoretische Basis hervor, sowie durch ihren Fokus auf das qualitative Verstehen der Risiken bevor sie quantifiziert werden. Diese Methode versucht Defizite und Unzulänglichkeiten des Risikomanagement-Ansatzes nach Boehm zu beheben und hat deshalb teilweise andere Ansätze. Riskit ist im Gegensatz zu Boehms Ansatz ein vollständig spezifizierter Prozess nachdem Risikomanagement durchgeführt werden kann und wird auch durch ein konkretes Werkzeug unterstützt.

Mit der Riskit-Methode sind mehrere Schwachpunkte in existierenden Risikomanagement-Methoden adressiert worden, insbesondere die Simplifizierung der Risiken und die unzureichende Quantifizierung.

Um Unklarheiten durch einen großen Bedeutungs-Umfang des Worts "Risiko" zu vermeiden, wird zuerst eine präzise und eindeutige Definition von Risiko eingeführt [4]:

“When we use the term risk on its own, we are using it in its general meaning: risk is de-fined as a possibility of loss, the loss itself, or any characteristic, object or action that is as-sociated with that possibility.”

Diese Definition erklärt die Eintritts-Wahrscheinlichkeit und das Verlustausmaß als Hauptattribute eines Risikos, wobei unter Verlust ein nicht die Erwartungen erfüllendes Ergebnis verstanden wird. Des Weiteren werden explizite Definition von „Objectives“, „Constraints“ and anderen Einflussfaktoren eingeführt, die das Risiko als relatives Konzept beschreiben, das von vielen Einflüssen abhängig ist.

Im Gegensatz zur rein quantitativen Beschreibung werden Risiken in Riskit qualitativ dokumentiert und modelliert. Dafür werden konzeptionelle und grafische Tools zur Verfügung gestellt und das Risiko formal mittels eines Riskit-Analysegraphen definiert. Der Riskit-Analysegraph dient der expliziten und formalen Definition von Risiko-Aspekten und Risiko-Szenarien. Er zerlegt das Risiko in klar definierte Risiko-Komponenten, sogenannte Risiko-Elemente.

Die Priorisierung der Risiken kann durch skalierte Angaben geschehen, um eine gewisse Verlässlichkeit zu bekommen. Dadurch wird das Problem der richtigen Einschätzung verkleinert, da nicht unrealistische Schätzungen einfließen werden, sondern nur soweit geschätzt werden muss, um das Risikomanagement durchführen zu können.

In Riskit wird das Konzept des „Utility Loss“ verwendet. Es erfasst, wie stark die möglichen Verluste aufgrund eines Risikos für die verschiedenen Interessensvertreter sind, anstatt wie sonst üblich dafür Attribute wie Kosten oder Zeitverlust zu benutzen. Das Konzept basiert auf dem Nutzen-Begriff der Ökonomie, bei dem Entscheidungen zwischen alternativen Produkten aufgrund des Nutzens, als Maß für die Fähigkeit die Bedürfnisse zu befriedigen, getroffen werden.

Der Risikomanagement-Prozess eines Projektes besteht dann aus folgenden Aktivitäten:

- Auftrags-Definition: Warum, wann und in welchem Umfang Risikomanagement durchgeführt werden soll.

- Ziel-Bewertung: Explizite Definition der Ziele aus Sicht der unterschiedlichen Interessensvertreter.
- Risiko-Identifikation: Mögliche Risiken für das Projekt identifizieren.
- Risiko-Analyse: Zusammenfassen und Klassifizieren der Risiken, Riskit-Analysegraph für alle Risiken erstellen.
- Planung der Risiko-Kontrolle: Auswahl von Kontroll-Maßnahmen für die wichtigsten Risiken mit dem Ziel der „Entschärfung" der Risiken
- Risiko-Kontrolle: Durchführen der Maßnahmen und dadurch Verringerung der Risiken.
- Risiko-Überwachung: kontinuierliche Überwachung der definierten Risiken.

Als ein wichtiger Punkt der Riskit-Methode ist die Modellierung der Perspektiven der verschiedenen Interessensvertreter zu sehen. So werden ihre unterschiedlichen Prioritäten und Erwartungen in Betracht gezogen und für jede Perspektive die Risiken und die damit verbunden möglichen Schäden individuell dokumentiert.

4.3. Bewertung von Boehm und Riskit

Wie alle existierenden Risikomanagement-Ansätze im Software Engineering sind die Methode von Boehm und Riskit hauptsächlich für das Anwendungsgebiet Betriebliche Informationssysteme ausgelegt. Dementsprechend werden hauptsächlich Risiken adressiert, die Kosten und Zeit beeinflussen. Die Betrachtungsebene ist in beiden Methoden der Entwicklungsprozess, so dass sie projekt-begleitend – mit einem Schwerpunkt auf einer Analyse am Anfang des Projekts – eingesetzt werden. Riskit ragt durch eine qualitative Risiko-Bewertung heraus, wohingegen Boehm – wie ein Großteil der anderen bekannten Methoden – die Risiken quantitativ bewertet.

Die Risiko-Bewertung findet in beiden Methoden in einem Top-Down-Ansatz ausgehend von der Problemdomäne und der Betriebserfahrung statt. Da die Kosten und Zeit betreffenden Risiken in vielen Software-Projekten die gleichen sind, wird ein Großteil der Risiken anhand der Erfahrung und der Durchführung früherer Projekte erkannt. Riskit unterscheidet sich von Boehm in der Sichtweise: Während die Methode von Boehm bedrohungszentriert ist, also anhand von potentieller Gefahren vorgeht, werden in Riskit zusätzliche unterschiedliche Interessen analysiert und so unterschiedliche Bedürfnisse der Beteiligten berücksichtigt.

Auf Basis der Methode von Boehm wurde das *Continous Risk Management* vom SEI entwickelt. Alle drei Methoden erfüllen die Anforderungen für das Risikomanagement von Softwareprojekten so wie es CMM Level 2 vorschreibt. Riskit lässt sich durch die explizite Beschreibung des Prozesses gut in der Praxis umsetzen.

5. Übertragung der SE-Ansätze auf traditionelle Disziplinen

Wenn man akzeptiert, dass Software anders ist und deshalb komplexe und neue Probleme lösen muss, könnte man sich eine Übertragung der Ansätze des Software Engineering auf andere Disziplinen vorstellen.. Vor dem Hintergrund des immer stärkeren Software-Anteils in vielen traditionellen Disziplinen keine ganz abwegige Idee. Viele Automobil-Hersteller kämpfen z.B. mit dem extrem hohen Software-Anteil und der daraus resultierenden Komplexität in aktuellen Fahrzeugmodellen.

Von dieser Seite könnte die Frage kommen, welche speziellen Methoden das Software Engineering bietet, um sie in den vorhandenen Prozess einzubauen. Schließlich kann es nur von Vorteil sein, sich weitgestreut qualitätssichernde Methoden für extrem komplexe Systeme genauer anzusehen und in Teilen geeignet zu übertragen. Mag dieser Grundgedanke noch stimmen, so ist der heutige Stand des Risikomanagements im Software Engineering aber noch nicht weit genug, anderen

Ingenieurdisziplinen praktikable und hilfreiche Methoden anbieten zu können. Das liegt zum einen an dem teilweise fehlenden Bewusstsein für Risikomanagement außerhalb der Kategorien Budget und Zeit, zu einem großen Teil aber an den ausgereiften Risikomanagement-Methoden des traditionellen Ingenieurwesens. Vor allem die Auswahl an genau zugeschnittenen und praktikablen Methoden hat gegenüber den im Großteil allgemein gehaltenen Ansätzen des Software Engineerings viele Vorteile.

6. Zusammenfassung

Diese Ausarbeitung beschrieb, warum Software anders ist und welche Herausforderungen und Probleme daraus – insbesondere für das Risikomanagement – resultieren. Es wurden die Risikomanagement-Methoden der traditionellen Disziplinen und die des Software Engineerings dargestellt und versucht Möglichkeiten zur Übertragung zu finden.

Viele hoffen auf ein Silver Bullet, eine Wunderdroge zur Lösung der z.B. hohen Komplexität und fehlenden Visualisierbarkeit von Software; eine bahn brechende neue Technologie oder Methodik, die das Software Engineering revolutioniert und alle Probleme bei der Erstellung komplexer Softwaresysteme vermeidet. Aber wie schon Brooks 1989 in [1] gesagt hat: „There is no Silver Bullet“. Die zu lösenden Probleme liegen im Wesen von Softwaresystemen und sind damit inhärent. Aber Silver Bullets hat es auch in den traditionellen Disziplinen nie gegeben. Auch sie mussten sich mühsam graduell weiterentwickeln. Dass das Risikomanagement im Software Engineering Nachholbedarf hat, liegt weniger an seiner zu langsamen Entwicklung als an der überproportional schnellen Entwicklung der Hardware gemäß Moore’s Law. Noch nie hat sich eine neue Technologie so schnell in ihren Eigenschaften (CPU-Leistung, Speicher) weiterentwickelt und ihre Kosten so dramatisch reduziert wie die Computer-Hardware. Damit wurde das Kosten-/Nutzen-Verhältnis für neue Anwendungen immer wieder neu definiert und ein bis heute nicht beherrschbares Momentum für neue Anwendungsideen ausgelöst. Auf eine Pause, die dem Software Engineering ein Aufholen ermöglicht, darf man auch die nächsten Jahre nicht hoffen.

Nichtsdestotrotz ist die Kenntnis von existierenden Methoden zum Risikomanagement und zur Qualitätssicherung in anderen Ingenieurwissenschaften hilfreich. Auch wenn die simple Übertragung kompletter Methoden nicht funktioniert, einzelne Elemente und vor allem das zugrunde liegende Denken können übernommen werden und helfen weiter.

Referenzen

[1] F.P. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering", *IEEE Computer*, Vol. 20 No. 4, pp. 10-19., Apr 1987.
[2] B. Beizer, “Software is different”, *Annals of Software Engineering*, Volume 10, Issue 1 - 4, pp. 293 – 310, Mar 2000.
[3] B. Boehm, “Software Risk Management: Principles and Practices”, *IEEE Software*, pp. 32-41, Jan 1990.
[4] J. Kontio, “The Riskit Method for Software Risk Management, version 1.00”, Computer Science Technical Reports, University of Maryland, MD, Tech. Rep. CS-TR-3782, 1997.
[5] T. DeMarco and T. Lister, *Waltzing with Bears: Managing Risk on Software Projects*. New York: Dorset Hous Publishing, 2003.
[6] W.S. Humphrey, “Software — A performing science?”, *Annals of Software Engineering*, Volume 10, Issue 1 – 4, pp. 261 - 271, Mar 2000.
[7] G.A. King, “Qualitiy technique transfer: Manufactoring and software”, *Annals of Software Engineering*, Volume 10, Issue 1 – 4, pp. 359 - 372, Mar 2000.

[8] R. Fairley, “Risk Management for Software Projects”, *IEEE Software*, pp. 57-67, May 1994.
[9] B. Boehm, T. DeMarco, "Software Risk Management", IEEE Software, pp. 17-19, May 1997.
[10] N. Leveson, *Safeware: System Safety and Computers*. New York: Addison-Wesley, 1995.
[11] R.N. Charette, *Software Engineering Risk Analysis and Management*. New York: McGraw-Hill, 1989.
[12] *SEI Risk Management*: URL: http://www.sei.cmu.edu/programs/sepm/risk/

Foliensätze zu Kapitel 1 bis 5

Problemorientierte Vorgehensweise

Bernd Burckhardt

WS 2004/05 Seminar „Risikomanagement für sicherheitskritische Softwaresysteme“
TU Kaiserslautern, AG Software Engineering, Prof. Dr. Dieter Rombach
Kaiserslautern, 3. Februar 2005

Fraunhofer Institut Experimentelles Software Engineering

Gliederung

Problemorientierte Vorgehensweise

- Warum das Ganze
- Methoden
- Toolunterstützung
- Übersicht

Warum das Ganze – kleine Anekdote

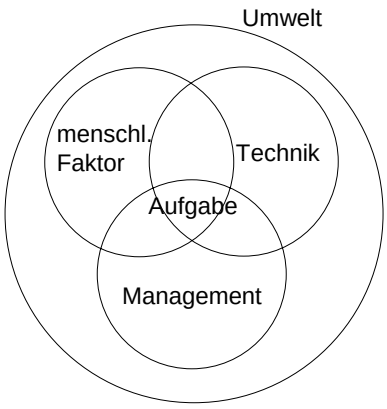
Gliederung
- Warum das Ganze
- Kleine Anekdote
- Systemmodell
- Risikomanagement
- Risikoeinordnung
- Methoden
- Toolunterstützung
- Übersicht

The vessel Baltic Star, registered in Panama, ran aground at full speed on the shore of an island in Stockholm waters on account of thick fog. One of the boilers had broken down, the steering system reacted only slowly, the compass was maladjusted, the captain had gone down into the ship to telephone, the lookout man on the prow took a coffee break, and the pilot had given an erroneous order in English to the sailor who was tending the rudder. The latter was hard of hearing and understood only Greek.

LeMonde

Warum das Ganze - Systemmodell

Gliederung
- Warum das Ganze
- Kleine Anekdote
- Systemmodell
- Risikomanagement
- Risikoeinordnung
- Methoden
- Toolunterstützung
- Übersicht



Warum das Ganze – Risikomanagement

- Mögliche zukünftige Probleme erkennen
  => Risiken erkennen
- Risiken vermeiden
  => Potentielle Probleme verringern
- Komplexe Systeme haben auch versteckte Risiken
- Nicht alle Risiken sind vermeidbar
  => Risiken einordnen

Warum das ganze - Risikoeinordnung

- Schwere der Risiko-Konsequenz

Description	*Category*
Catastrophic	I
Critical	II
Marginal	III
Neglible	IV

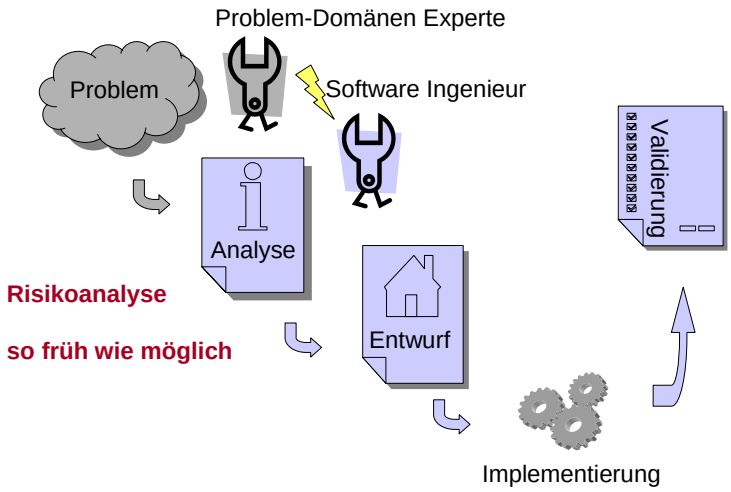
- Wahrscheinlichkeit des Eintretens

Description	*Level*
Frequent	A
Probable	B
Occasional	C
Remote	D
Inprobable	E

Quelle: MIL-STD-882c

Methoden – Wo anfangen

Gliederung
- Warum das Ganze
- Methoden
- Wo anfangen
- Wie anfangen
- What-If
- Checklist
- Software Hazard
- Toolunterstützung
- Übersicht

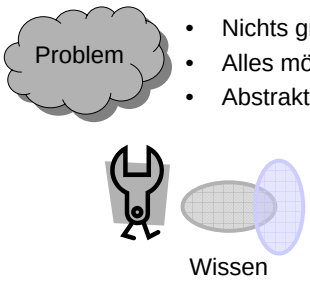


Risikoanalyse

so früh wie möglich

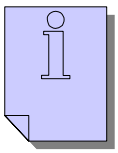
Methoden – Wie anfangen

Gliederung
- Warum das Ganze
- Methoden
- Wo anfangen
- Wie anfangen
- What-If
- Checklist
- Software Hazard
- Toolunterstützung
- Übersicht



- Nichts greifbar
- Alles möglich
- Abstraktes, virtuelles Konstrukt

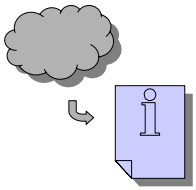
- Kreativ sein
- Alles in Frage stellen
- Mut zu ‚gewagten' Fragen
- „Ich weis, dass ich nichts weis"



- Beschreibung
- Anforderungen

Methoden – What-If

- Brainstorming ähnliche Methode
- Einsatzgebiet
  - Analysephase und initiale Problemfindung
- Anforderungen
  - Erfahrung mit Ingenieur- und Prozessabläufen
  - Experte aus der Problem-Domäne
  - Experte in der Analysemethode



Methoden – What-If

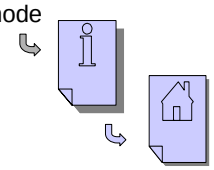
- Vorgehen
  - Wenig strukturierte Methode
  - Team analysiert Problem und Anforderungen
  - Sucht nach What-If's – mögliche Risiken
    - Konsequenzen, Präventionen und Alternativen
  - Unbewertete Resultaterfassung

What IF	Concequences	Protection	Scenario	Comments
… Seitenwind bei der Landung ?	Ungleiche Gewichtsverteilung auf Fahrwerk	keine		
… Regen bei der Landung ?	Aquaplaning	Spezialreifen	3a	

Methoden – Checklist Analysis

Gliederung
- Warum das Ganze
- **Methoden**
  - Wo anfangen
  - Wie anfangen
  - What-If
  - **Checklist**
  - Software Hazard
- Toolunterstützung
- Übersicht

- Einfache, bewährte, strukturierte Analysemethode
  - Aufbauend auf What-If
  - Checklisten sorgen für Struktur



- Einsatzgebiet
  - Weites Spektrum
    - Analyse, Design, Implementierung
- Anforderungen
  - Erfahrung mit Ingenieur- und Prozessabläufen
  - Sehr gute Projektkenntnisse
    - über Systementwurf, Projektziel, Ablauf
  - Keine Erfahrung mit Analysemethode notwendig
    - kreative Vorstellungskraft

Methoden – Checklist Analysis

Gliederung
- Warum das Ganze
- **Methoden**
  - Wo anfangen
  - Wie anfangen
  - What-If
  - **Checklist**
  - Software Hazard
- Toolunterstützung
- Übersicht

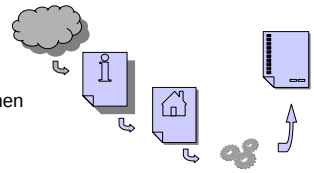
- Vorgehen
  1. Vorbereitung zum Review
  2. Liste von What-If's und Ursachen erstellen
  3. mit Checkliste Lücken überbrücken
  4. Auswerten der Ergebnisse
  5. Dokumentieren der Ergebnisse

  - Dokumentation wie What-If
    - Bewertung durch Checklisten möglich
  - Teamleiter kann auch Teilaufgaben erstellen (Divide & Conquer)
  - Schritt 2 und 3 können auch iterativ verfeinert werden
  - Checklisten erstellen und pflegen

Methoden – Software Hazard Analysis

Gliederung
- Warum das Ganze
- **Methoden**
- Wo anfangen
- Wie anfangen
- What-If
- Checklist
- **Software Hazard**
- Toolunterstützung
- Übersicht

- In Entwicklungsprozess integrierte Software-Sicherheit
  - Methoden-Paket
    - strukturiert analytisches Vorgehen
    - Schwächen identifizieren und bewerten
    - Schwächen beseitigen oder abschwächen



- Einsatzgebiet
  - Kompletter SE-Prozess
- Anforderungen
  - Erfahrung mit Ingenieur- und Prozessabläufen
  - Erfahrung mit Problem- und Systemdomäne
  - Kenntnisse in theoretischen und praktischen Verifikationstechniken

Methoden – Software Hazard Analysis

Gliederung
- Warum das Ganze
- **Methoden**
- Wo anfangen
- Wie anfangen
- What-If
- Checklist
- **Software Hazard**
- Toolunterstützung
- Übersicht

- Vorgehen
  - Anwendung verschiedener Methoden
    (nicht beschränkt auf folgende:)
    - Preliminary Hazard Analysis
    - Fault Hazard Analysis applied to software
    - Soft-Tree (Fault Tree Analysis applied to software)
    - Software Sneak Circuit Analysis
    - Software Checklists
    - Software design guidelines
    - Software coding standards
    - Petri Net Analysis
  - Unterscheidung: Software & Hardware Safety
    - „Software does not fail“
      SW altert nicht
    - „Hardware is predictable“
      HW verschleißt

Toolunterstützung - SpecTRM

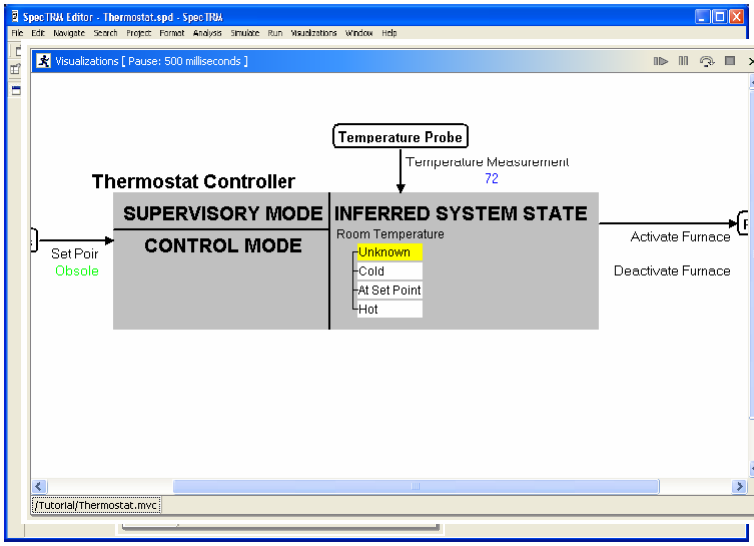
Gliederung
- Warum das Ganze
- Methoden
- Toolunterstützung
- Übersicht

- SpecTRM = Specification Tools and Requirements Methodology
  - Toolset to support the specification and development of safe systems and software
  - Development environment supports
    - Assurance through inspections
    - Formal validation tools
    - Simulation
  - Finding errors early in development
    - Fixed with the lowest cost and impact on system design
  - Tracing not only requirements
    - Design rationale including safety constraints throughout system design and documentation
  - Building required system properties into the design from the beginning
    - At the end effective response is limited and costly

Quelle: Safeware Engineering

Toolunterstützung - SpecTRM

Gliederung
- Warum das Ganze
- Methoden
- Toolunterstützung
- Übersicht



Übersicht – Stärken Schwächen

Methode	Vorteile	Nachteile
What-If	-einfach -benutzerfreundlich -billig	-unstrukturiert -eher für kleine Probleme -Ergebnisqualität hängt stark von Erfahrung ab
Checklist Analysis	-strukturiert -> aufteilbar -vielseitig -Methodenkenntnisse nicht nötig -Lerneffekt durch Checklisten	-zeit- und arbeitsintensiv bei komplexen Problemen -Ergebnisqualität wird durch Checklisten beeinflusst
Software Hazard Analysis	-Paket -> skalierbar -in kompletten SE-Prozess integrierbar -SW-angepasst (NASA, Militär)	-großes Paket -hohe Team-Anforderungen

Danke

Tool-URL: http://www.safeware-eng.com/index.php/products

Toolunterstützung - SpecTRM

- MIT Aero/Astro
  Software Engineering Research Laboratory (SERL)
  - Case Studie - Use of SpecTRM in Space Applications
    (by Masafumi Katahira, Prof Nancy G. Leveson)
- Potential uses of this modeling approach
  - Conceptual system design, which is done at the beginning of the development phase
  - Reverse engineering including accident investigations or problem corrections
  - Shadow process as the IV&V
  - A part of the development process itself
- Conclusion
  - The accident model analysis by SpecTRM is very significant for space application.

Quelle: MIT
http://sunnyday.mit.edu/papers.html

Applicability of SpecTRM to NASA SW-safety standard

NASA Software Safety Standard (NASA-STD-8719.13A)	Intent Specification / SpecTRM tool
System safety analysis Preliminary Hazard Analysis, software safety analysis to define safety-critical software (potential cause, or supporting the control of a hazard) and software safety requirements through the project life cycle.	In level 1 of the intent specification, the hazard analysis data can be described including software fault tree analysis. The software safety requirements can be described in level 1 as safety constraints
Software safety The objective is to ensure that safety is considered throughout the software life cycle.	All life cycle activities can be treated as the appropriate level of intent, refinement, or decomposition.
Software safety tasks by life cycle phase Planning shall be documented in Software Management plan or Safety Management Plan	The planning can be included or referenced in the "Program Management Plan" or "System Safety Plan" parts of the intent specification.

Quelle: MIT

Software safety tasks by life cycle phase

- **Software safety planning**
  - Planning shall be documented in Software Management plan or Safety Management Plan
- **Software Requirements Specification Developments**
  - -0 Development of software safety requirements
  - -1 Analysis of software requirement for potential hazards.
  - -2 Test planning is begun.
- **Software architectural design**
  - This design process shall include identification of safety design features and methods
- **Software detailed design**
  - Low-level design for the software units, safety-related information shall be into all user manuals.
  - Development of test procedure
- **Software implementation**
  - Code, which shall implement the safety feature and methods developed during the design process
- **Software integration and acceptance testing**
  - Testing to verify of software safety requirements including hazard verification, Acceptance testing verify in conjunction with system hardware and operators.
- **Software operations and maintenance**
  - When changes are made, performing hazard analysis, updating software safety requirements and specification, design, and operator document, regression testing

Quelle: MIT

Strukturorientierte Vorgehensweisen im Risikomanagement sicherheitskritischer Software-Systeme

Mathias Int-Veen

WS 2004/05 Seminar „Risikomanagement für sicherheitskritische Softwaresysteme“
TU Kaiserslautern, AG Software Engineering, Prof. Dr. Dieter Rombach
Kaiserslautern, 3. Februar 2005, Raum 57/528

Fraunhofer Institut Experimentelles Software Engineering

Aufbau

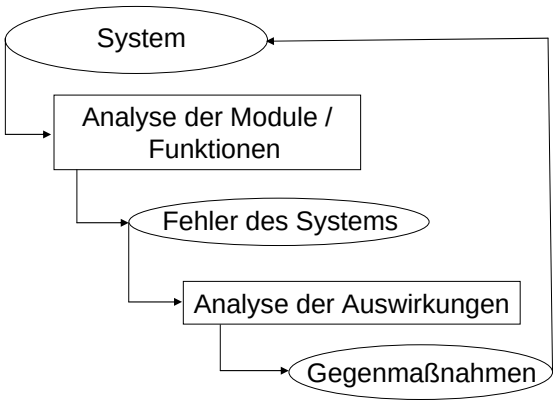
Gliederung
- Aufbau
- Grundlagen
- FMEA
- SFMEA
- Diskussion
- Zusammenfassung

- Strukturorientierte Vorgehensweisen
  - Grundlagen / Merkmale
  - Vertreter
- FMEA - Hardwaresysteme
  - FMEA
  - Beispiel
  - FMECA
- SFMEA – Softwaresysteme
- Diskussion
- Zusammenfassung

Strukturorientierte Vorgehensweisen

Gliederung
- Aufbau
- Grundlagen
- FMEA
- SFMEA
- Diskussion
- Zusammenfassung

Vom System zum Fehler und seiner Auswirkung



Vertreter strukturorientierter Vorgehensweisen

Gliederung
- Aufbau
- Grundlagen
- FMEA
- SFMEA
- Diskussion
- Zusammenfassung

- (S)FMEA – Identifikation der Fehlermodi der Systemmodule
- Energy Analysis – Identifikation aller Energieflüsse und ihrer Auswirkungen
- Network Logic Analysis – Boolsche Repräsentation des Systems
- Petri Netze – Umsetzung in Petri Netze

⇨ Oft weitere Formalisierung des System notwendig als Grundlage

FMEA - Failure Modes and Effects Analysis

Gliederung
- Aufbau
- Grundlagen
- FMEA
- SFMEA
- Diskussion
- Zusammenfassung

- Prozedur, bei der jeder Fehlerzustand analysiert wird, um den Effekt auf das Gesamtsystem zu bestimmen
- Deduktives Vorgehen:
  Evaluation des Systems → Risikoidentifikation
- 1980 als Ausfalleffektanalyse in die DIN 25448 aufgenommen
- Failure Modes, Effects and Criticality Analysis (FMECA)
  – Erweiterung um bewertete Rangordnung der Fehlermodi
  – Abgrenzung nicht trennscharf

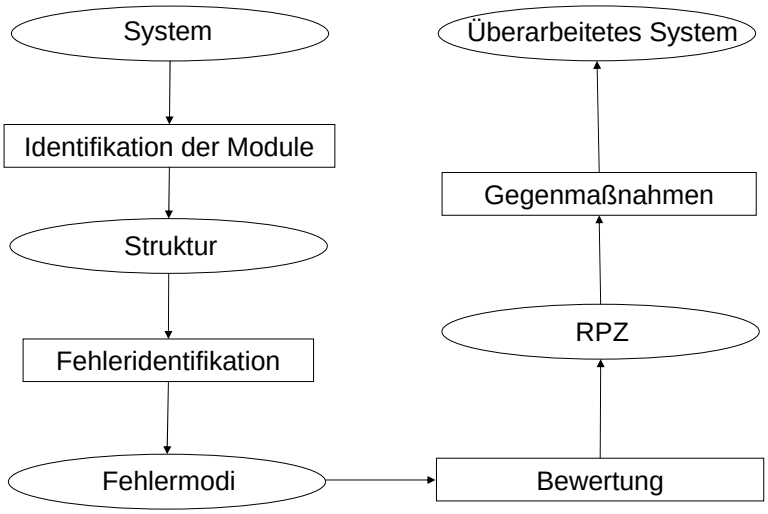
FMEA

Gliederung
- Aufbau
- Grundlagen
- FMEA
- SFMEA
- Diskussion
- Zusammenfassung

- Verschiedene Arten der FMEA, je nach Betrachtungszeitpunkt und Umfang
  – System-FMEA: Betrachtung des funktionsgerechten Zusammenwirkens der Systemkomponenten
  – Konstruktions-FMEA: Betrachtung der Merkmalsebene für Bauteile, Baugruppen
  – Prozess-FMEA: Betrachtung der Gesamtheit der Abläufe mit dem Ziel die Einhaltung der Merkmale zu beachten

FMEA

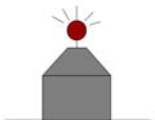
Gliederung
- Aufbau
- Grundlagen
- FMEA
- SFMEA
- Diskussion
- Zusammenfassung



Beispiel (I)

Gliederung
- Aufbau
- Grundlagen
- FMEA
- SFMEA
- Diskussion
- Zusammenfassung

Wegführung der Autopiloten durch sensorengesteuertes Radarsystem



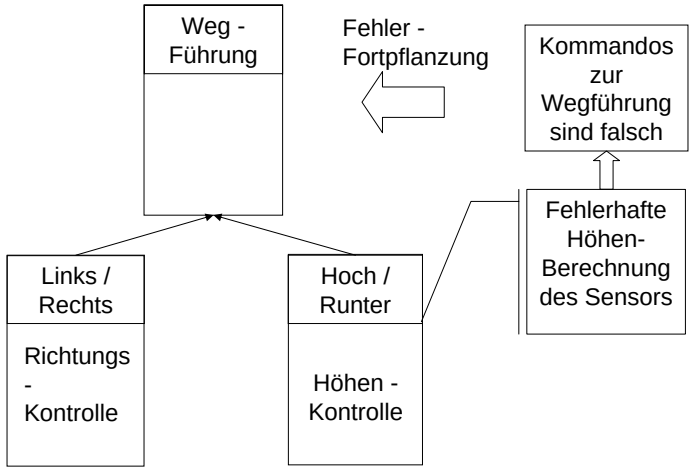
Beispiel (II)

Gliederung
- Aufbau
- Grundlagen
- FMEA
- SFMEA
- Diskussion
- Zusammenfassung

- Fehler im Sensor
  - Höhe der Flugzeuge wird falsch bestimmt
  - Sensor fällt aus, Höhe kann nicht bestimmt werden
  - Sensor erkennt Flugzeug nicht

Beispiel (III): Module → Fehlererkennung

Gliederung
- Aufbau
- Grundlagen
- FMEA
- SFMEA
- Diskussion
- Zusammenfassung



Beispiel (IV) - Fehlerbewertung

- Keine einheitlich anerkannten Regelungen
- Prinzipielles Vorgehen: Bewertung durch Zahlen, hier: je kleiner desto besser (1-10)

Bedeutung des Fehlers / Criticality :		Auftritts-Wahrscheinlichkeit :		Entdeckungs-Wahr-scheinlichkeit		
10	*	9	*	10	=	900

Risikoprioritätskennzahl

Beispiel (V): Ursache und Auswirkung

User Req. (UR)	Failure Modes (FM)	Be-troffene UR	FM höherer Funktion	Effekte System Level	RPZ
Aus-weichen: hohes Flugzeug nach oben, tieferes nach unten	Be-rechnung der Höhe falsch	XX	Kommandos zur Wegführung sind falsch	Kollisions risiko	900

⇨ Gegenmaßnahmen definieren

Vollständigkeit

Gliederung
- Aufbau
- Grundlagen
- FMEA
- SFMEA
- Diskussion
- Zusammenfassung

- Die Vollständigkeit der FMEA hängt von drei Faktoren ab
  - Aufwand um Fehlermodi zu identifizieren
  - Aufwand um Fehlerpropagierung zu verfolgen
  - Aufwand mit dem Zusammenhänge identifiziert werden
- Probleme bei multiplen Fehlern:
  - Fehler die sich aus verschiedenen, möglicherweise richtigen Aspekten zusammensetzen
  - Zusammengesetzte Auswirkungen von mehreren Fehlern für übergeordnete Komponenten
- Einsatz eines interdisziplinären Teams um verschiedene Perspektiven zu gewährleisten

FMECA

Gliederung
- Aufbau
- Grundlagen
- FMEA
- SFMEA
- Diskussion
- Zusammenfassung

- Erweiterung der FMEA um Fehlerraten
- Betrachtet werden:
  - Ausfallwahrscheinlichkeit über Zeitspanne
  - Fehler über Zeitspanne
- Qualitativer Ansatz: wenn keine genauen Fehlerraten für Komponenten bekannt sind
- Quantitativer Ansatz: wenn genaue Werte bekannt sind
- Geschieht anhand von Listen in denen Vergleichswerte angegeben sind

SFMEA – Software Failure Modes and Effects Analysis

Gliederung
- Aufbau
- Grundlagen
- FMEA
- SFMEA
- Diskussion
- Zusammenfassung

- Übertragung von FMEA auf Software
- Bewertet das Gefahrenpotential das durch das Versagen von Software-Modulen entsteht
- Anforderungs – und Designphase, noch vor dem Code (Pseudocode ist aber hilfreich)
- Vorgehen prinzipiell wie bei FMEA
- Erlaubt den Schritt zum Gesamtsystem

Fehlerpropagierung

Gliederung
- Aufbau
- Grundlagen
- FMEA
- SFMEA
- Diskussion
- Zusammenfassung

- Fehler können sich im System nur entlang von Kontroll- / Datenströmen fortpflanzen
- Nachricht
  - Zu spät
  - Zu früh
  - Gar nicht
  - fehlerhaft

SFMEA

Gliederung
- Aufbau
- Grundlagen
- FMEA
- SFMEA
- Diskussion
- Zusammenfassung

- (S)FMEA ist beschränkt im Bereich multipler Fehler, jeder Failure Mode ist prinzipiell unabhängig
- Erfahrungswerte aus der FMEA lassen sich im Allgemeinen nicht übertragen
- Risikoprioritätskennzahlen müssen unabhängig bewertet werden
- Hohe qualitative Anforderungen an die Dokumente des Entwurfs als Grundlage der (S)FMEA

Kombinierte Hard- und Softwaresysteme

Gliederung
- Aufbau
- Grundlagen
- FMEA
- SFMEA
- Diskussion
- Zusammenfassung

- Software erhöht die Kontrollmöglichkeiten von Systemen, aber auch die Fehleranfälligkeit
- SFMEA bietet Ansätze zur Generierung von Testfällen für das Gesamtsystem
- Einzeltechniken können aber keine umfassende Lösung für die Probleme hybrider Systeme sein

  ⇨ System-Wechselwirkungen müssen betrachtet werden, z.B. Zeitaspekte

Vor- und Nachteile strukturorientierter Maßnahmen

Vorteile:

- (fast) alle Fehler im System können gefunden werden
- Kosteneinsparung durch frühzeitige Fehlererkennung in allen Phasen
- Tw. auch für komplexe Systeme geeignet (z.B. FMEA)

Nachteile:

- Keine menschlichen / domänenorientierten Fehler
- Tw. hoher Aufwand / Kosten
- Tw. hoher Lernaufwand vorrausgesetzt
- Tw. auf bestimmte Phasen des Entwurfs beschränkt

Zusammenfassung

- Strukturorientierte Maßnahmen stellen das System in den Mittelpunkt der Betrachtung
- Vorwärtsbetrachtung anhand von Ereignisketten
- Exogene Beeinträchtigungen und Fehler werden nicht betrachtet
- Gut geeignet um Fehler des Systems selber festzustellen
- Keine vollständigen Ergebnisse
  ⇨ eingeschränkte Perspektive

Gliederung

- Aufbau
- Grundlagen
- FMEA
- SFMEA
- Diskussion
- Zusammenfassung

Ich danke für ihre Aufmerksamkeit, und stehe für Fragen zur Verfügung

Post-Mortem-Ansätze/ Kontinuierliche Verbesserung (Betriebserfahrung)

Pascal Hagedorn

WS 2004/05 Seminar „Risikomanagement für sicherheitskritische Softwaresysteme“
TU Kaiserslautern, AG Software Engineering, Prof. Dr. Dieter Rombach
Kaiserslautern, 3. Februar 2005, Gebäude 57 Raum 528

Fraunhofer Institut Experimentelles Software Engineering

Eingliederung in das Seminar

„Risikomanagement für sicherheitskritische Softwaresysteme“

1. Übersichtsvortrag (Safety/Security Engineering)
2. Problemorientierte Vorgehensweise (Domänenwissen)
3. Strukturorientierte Vorgehensweise (Systemwissen)
4. **Post-mortem-Ansätze/Kontinuierliche Verbesserung (Betriebserfahrung)**
5. Human Factors
6. Software-spezifische Ansätze

Gliederung

1. Betrachtungsweisen
2. SSE-CMM (System Security Engineering Capability Maturity Model)
3. Root-Cause-Analyse
4. Zusammenfassung

"We installed little monitors because they make all of our problems look smaller."

Betrachtungsweisen

- Makroskopische Ansätze
  - Aspektebene und
  - Prozessebene
- Mikroskopische Ansätze
  - Methodenebene und
  - Produktebene

Beide Ansätze ergänzen sich gegenseitig

- Kontinuierliche Verbesserung
  - Alle Lebenszyklusphasen
  - Kontinuierlicher Prozess
  - Bsp. SSE-CMM
- Post-Mortem-Ansätze
  - Späte Lebenszyklusphasen
  - Vorfälle (Unfall, Katastrophe, …)
  - Bsp. Root-Cause-Analyse

(SSE-CMM)
Systems Security Engineering Capability Maturity Model

- Hauptentwicklungsorganisationen
  - National Security Agency (NSA) - USA
  - Office of the Secretary of Defense (DoD) - USA
  - Communications Security Establishment – Kanada
- Mehr als 50 weitere Organisationen (USA, Kanada und Europa)
  - Cisco Systems
  - Motorola
  - Oracle Corporation
  - …
- Chronologie der Entstehung
  - NSA ruft im April 1993 SSE-CMM Initiative ins Leben
  - 1. Security Engineering Workshop im Januar 1995
  - März 1995 Bildung von Arbeitsgruppen
  - 1. Version im Oktober 1996
  - 2. Version 1999
  - 3. Version 15. Juni 2003

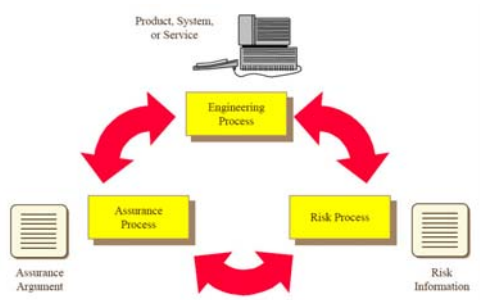
Security Engineering

- Ziele 
  - Sicherheitsrisiken in Organisationen erkennen
  - Ausgewogene Sicherheitsmaßnahmen zu den erkannten Sicherheitsrisiken einführen
  - Umsetzen eines Bewusstseins für die Notwendigkeit und Effektivität von Sicherheitsmaßnahmen
  - Sicherstellung, dass Vorfälle toleriert werden können (akzeptierbares Risiko)
  - Vereinigung aller ingenieurswissenschaftlichen Disziplinen und Besonderheiten, um ein vertrauenswürdiges und zuverlässiges System zu schaffen
- Security Engineering Aktivitäten werden über alle Lebenszyklusphasen durchgeführt

Security Engineering Prozess

3 Hauptgebiete des Security Engineering

- Risiko
- Engineering
- Gewährleistung



SSE-CMM Framework Teilbereiche

- Risiko-Prozesse
  Beurteilung von Bedrohungen, Schwachstellen und Schadwirkungen

- Engineering-Prozesse
  Identifikation von Sicherheits-anforderungen, sowie Auswahl von Realisierungsalternativen und deren technischer Umsetzung

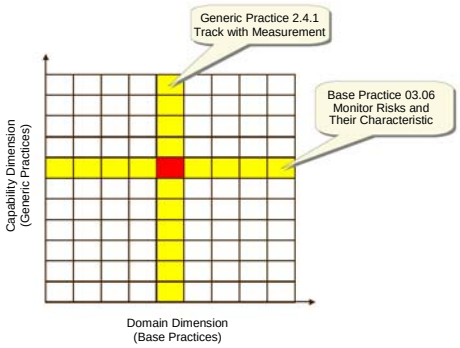
- Gewährleistungs-Prozesse
  Verifizierung und Validierung von Sicherheitsmechanismen und Bereitstellung überzeugender Nachweise

SSE-CMM Architektur

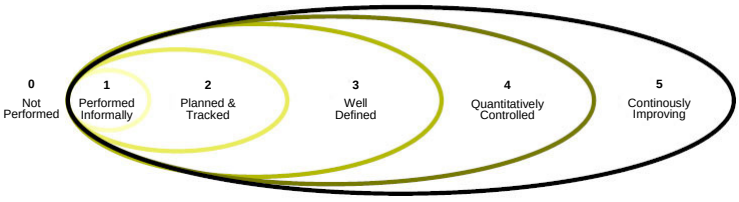
Klare Trennung zwischen

- Grundcharakteristiken des Security Engineering Prozess
- Management und institutionale Charakteristiken

⇒ 2 Dimensionen
- Domain
- Capability



Reifegrade (Capability Levels)



1. „You have to *do it* before you can manage it."
2. „*Understand what's happening* on the project before defining organization-wide processes."
3. „*Use the best of what you've learned* from your projects to create organization-wide processes."
4. „*You can't measure it until you know what ‚it' is.*" und „Managing with measurement is only meaningful when you're measuring the right things."
5. "A *culture of continuous improvement* requires a foundation of sound management practice, defined processes, and measurable goals."

Grundpraktiken (Base Practices)

Gliederung
- Betrachtungsweisen
- SSE-CMM
- Root-Cause-Analyse
- Zusammenfassung

- **Grundpraktik**
  - Überlappen sich nicht gegenseitig
  - Repräsentiert eine „Best Practice“ der Security Community
  - Reflektiert nicht nur „state-of-the-art“
  - Durch die Benutzung mehrerer Methoden, in mehreren Geschäftszusammenhängen anwendbar.
  - Spezifiziert keine bestimmte Methode oder bestimmtes Werkzeug

- **Prozessbereich** (Process Area)
  - Zusammenfassung ähnlicher Aktionen
  - Kann in mehreren Produkt- und Organisationszusammenhängen umgesetzt werden
  - Kann als einzelner Prozess verbessert werden
  - Beinhaltet alle Grundpraktiken, um die Ziele des jeweiligen Prozessbereichs zu erreichen

Beispiel Grundpraktiken (Base Practices)

Gliederung
- Betrachtungsweisen
- SSE-CMM
- Root-Cause-Analyse
- Zusammenfassung

- BP.03.06 – Monitor Risks and Their Characteristic
  Monitor ongoing changes in the risk spectrum and changes to their characteristics.
  - **Description**
    The risk spectrum applicable to any location and situation is dynamic. New risks can become relevant and the characteristics of existing risks can change. It is therefore important to monitor both existing risks and their characteristics, and to check for new risks on a regular basis. This base practice is closely linked to the generalized monitoring activity in BP.08.02 Monitor changes in threats, vulnerabilities, impacts, risks and the environment.
  - **Example Work Products**
    - risk monitoring reports – reports describing the current risk spectrum
    - risk change reports – describes the operational capabilities of a system and their importance to the objective of the system.
  - **Notes**
    Because risks can change, the risk assessment activity can be conducted multiple times in the defined environments. However, risk assessment repetition should not supplant risk monitoring.

Generische Praktiken (Generic Practices)

Gliederung
- Betrachtungsweisen
- SSE-CMM
- Root-Cause-Analyse
- Zusammenfassung

- Auf alle 3 Prozesse anwendbare Aktivitäten
- Zielen auf
  - das Management,
  - die Messbarkeit und
  - die Institutionalisierung

  eines Prozesses ab
- Im allgemeinen werden sie während einer Bewertung eingesetzt um den Reifegrad einer Organisation in der Ausführung von Prozessen zu bestimmen
- Zusammenfassung in „Common Features“
  - Attribute die erfüllt sein müssen um einen bestimmten Reifegrad zu halten oder zu erreichen
  - werden durch die generischen Praktiken definiert

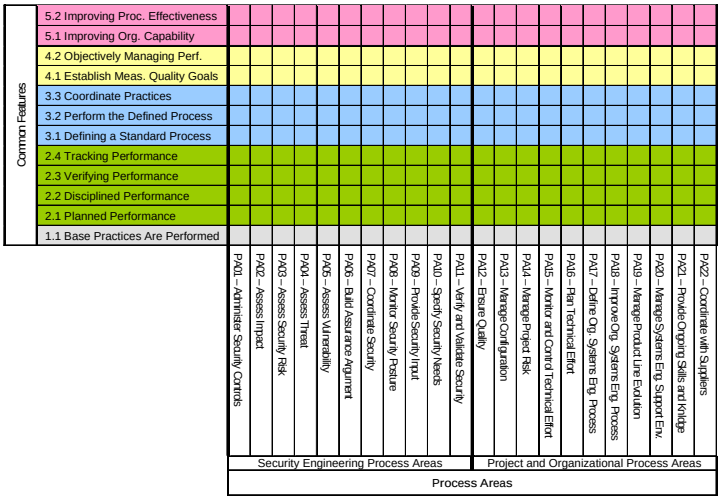
Beispiel Generische Praktiken

Gliederung
- Betrachtungsweisen
- SSE-CMM
- Root-Cause-Analyse
- Zusammenfassung

- GP 2.4.1 – Track with Measurement
  - **Description**
    Track the status of the process area against the plan using measurement.
  - **Notes**
    Building a history of measures is a foundation for managing by data, and is begun here.
  - **Relationships**
    Relationship to other generic practices: The use of measurement implies that the measures have been defined and selected in 2.1.3 and 2.1.6, and data have been collected in 2.2.1.

    Project tracking is described in process area PA13 Monitor and Control Technical Effort.

Summary Chart



SSE-CMM Fazit

- Pro
  - Geeignet für verschiedene Organisationsformen und Organisationsgrößen
  - Umspannt alle Lebenszyklusphasen
  - Für Hersteller und Anwender
  - Hilft Kompetenzdefizite zu identifizieren
  - Hilft Risiken zu eliminieren oder akzeptabel zu machen
- Contra
  - Wenig hilfreich auf Produkt- und Methodenebene
  - Mangel an konkreten Anweisungen
  - Nur Prozess- und Aspektebene
  - Fast „nur" Evaluierung

Root-Cause-Analyse

Gliederung
- Betrachtungsweisen
- SSE-CMM
- Root-Cause-Analyse
- Zusammenfassung

- Ganzheitlich analytische Methodik
- Ausgangssituation
  - Vorfall
  - Schwerwiegende oder häufige Qualitätsmängel
  - Sinnvoll ab Test-Phase
- Alle Betrachtungsebenen
  - Sicherheitsaspekte
  - Prozessfehler
  - Methodenmängel
  - Produktmängel
- Ziel
  - Beseitigung von Fehlerbündel
  - Vermeidung zukünftiger Vorfälle
  - Qualitätsverbesserung
  - Feedback

RCA Vorgehensweise [1]

Gliederung
- Betrachtungsweisen
- SSE-CMM
- Root-Cause-Analyse
- Zusammenfassung

- Vor dem Vorfall
  - Infrastruktur
  - Informationen
  - Schulungen
- RCA-Team bilden
  - 2 Teamleiter
    - Experte für RCA-Prozess und RCA-Sitzungen **Moderator**
    - Experte für Domäne als **Fachleiter**
  - Team
    - Mindestens 1 Repräsentant jeder Gruppe
    - Im Zweifel lieber mehr Teilnehmer
- Individueller chronologischer Ablauf des Vorfalls
  - Was geschah der Reihe nach?
  - Was waren die eigenen Beobachtungen?
  - Was ging dem Beobachter dabei durch den Kopf?

RCA Vorgehensweise [2]

Gliederung
- Betrachtungsweisen
- SSE-CMM
- Root-Cause-Analyse
- Zusammenfassung

- Hintergrundinformationen
  - Log-Daten und Betriebsunterlagen
  - Systemhandbücher und Arbeitsanweisungen
  - Zeugenaussagen
  - Fachliteratur
- 1. Sitzung
  - Rekonstruktion des Zwischenfalls
  - Tatsächlich angewandte Gegenmaßnahmen identifizieren
  - Negative Ereignisse und Maßnahmen
  - Brainstorming nach „Faktoren“
  - Gruppierung der „Faktoren“ und Benennung eines Oberbegriffs

RCA Vorgehensweise [3]

Gliederung
- Betrachtungsweisen
- SSE-CMM
- Root-Cause-Analyse
- Zusammenfassung

- Zwischen 1. und 2. Sitzung
  - Erstellung eines „Faktorenbaums“
  - Frage zu jedem Einflussfaktor
    - Warum hat er Einfluss auf den Schadensverlauf genommen?
    - In welcher Weise kam die Wirkung zustande?
  - Blätter (‚fehlende Information‘, ‚begünstigend‘ und ‚irrelevant‘)
- 2. Sitzung
  - Ergänzungen und Korrekturen des „Faktorenbaums“
    - ‚fehlende Informationen‘ werden ergänzt
    - ‚begünstigend‘ und ‚irrelevant‘ werden diskutiert
    - Eventuell domänenspezifische Checklisten
  - Maßnahmeplan:

Begünstigungs-faktor	Gegenmaß-nahme	Verantwortlicher	Durchzuführen bis …	Erfolgs-kontrolle

RCA Vorgehensweise [4]

Gliederung
- Betrachtungsweisen
- SSE-CMM
- Root-Cause-Analyse
- Zusammenfassung

- 3. Sitzung
  - Übersichtliche, bereinigte Präsentation der Ereignisfolge, der identifizierten Begünstigungsfaktoren, sowie des vervollständigten Maßnahmeplans.
  - „Ideenspeicher“ überprüft
    - Wurden alle Ideen angemessen verwertet?
    - Gibt es noch interessante Punkte, die man aufgreifen sollte?
  - Überprüfungsinstanz für Umsetzung des Maßnahmenplans
  - Rückmeldung an alle RCA-Beteiligten (Besonders „Melder“)
- RCA ist wirkungsvoll, wenn
  - Geringe Latenz zwischen Vorfall und RCA
  - Viele Daten und Informationen
  - Systematische Überwachung von IT-Systemen
  - Gute Infrastruktur

RCA-Fazit

Gliederung
- Betrachtungsweisen
- SSE-CMM
- Root-Cause-Analyse
- Zusammenfassung

- Pro
  - „Geringe“ Kosten
  - Ganzheitliche Betrachtung der Zwischenfälle
  - Hilft Verbesserungspotentiale zu erkennen
  - Nutzung durch Hersteller und Anwender
  - Ergebnisse hilfreich für frühe Phasen zukünftiger Projekte (Feedback)
- Contra
  - Nur für etablierte Systeme nutzbar
  - Ausreichend Betriebserfahrung
  - Genaue Störmeldungen und Daten
  - Selektion der Vorfälle

Zusammenfassung

Gliederung
- Betrachtungsweisen
- SSE-CMM
- Root-Cause-Analyse
- Zusammenfassung

- SSE-CMM
  - Prozessorientierung
  - „Grundideologie“
  - Daten und Informationen beschaffen
  - Konstruktive Methode
  - Interdisziplinär
  - ⇒ **Kontinuierliche Verbesserung**
- Root-Cause-Analyse
  - Kann gesammelte Daten von SSE-CMM nutzen
  - Ganzheitlichkeit
  - Akribische Aufklärung der Ursache
  - ⇒ **Post-Mortem-Ansatz**

⇒ Nutzbar durch Hersteller und Anwender

⇒ Abdeckung der Makro- und Mikroskopischen Ansätze

⇒ Aus Fehlern lernen, um Risiken zu minimieren

"'Be Careful'! All you can tell me is 'Be careful'?"

The Human Factor

Die Menschliche Komponente

Sebastian Weber

WS 2004/05 Seminar „Risikomanagement für sicherheitskritische Softwaresysteme“
TU Kaiserslautern, AG Software Engineering, Prof. Dr. Dieter Rombach
Kaiserslautern, 3. Februar 2005, Raum 57-528

Fraunhofer Institut Experimentelles Software Engineering

Übersicht

Gliederung

➔ Einführung ins Thema

- Aufgabenverteilung in automatisierten Systemen
- Mensch-Maschine-Schnittstelle
- Schlusswort

Einführung: Human Factors – Was ist das eigentlich?

Gliederung

- Einführung
- Aufgabenverteilung in automatisierten Systemen
- Mensch-Maschine-Schnittstelle
- Schlusswort

‘Human Factors’ (HF) ist eine interdisziplinäre Anstrengung um Informationen über menschliche Fähigkeiten und Beschränkungen zu sammeln und diese Informationen auf Ausrüstung, Systeme, Software, Anlagen, Prozeduren, Aufgaben, Arbeitsumgebung, Training, Stellenbesetzung und Personal-Management anzuwenden um sichere, angenehme und effektive menschliche Leistung zu ermöglichen.

[Quelle: FAA, System Safety Handbook, Chapter 17]

Einführung: Wo liegt das Problem?

Gliederung

- Einführung
- Aufgabenverteilung in automatisierten Systemen
- Mensch-Maschine-Schnittstelle
- Schlusswort

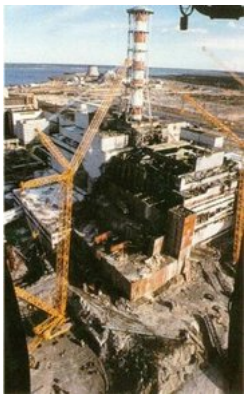
Zur Erinnerung:

- Ein System ist sicherheitskritisch, wenn eine Fehlfunktion zu folgenden Konsequenzen führen kann:
  - Verlust von Menschenleben, Verletzung oder Gesundheitsgefährdung von Menschen
  - Schwerwiegende Schädigung der Umgebung
  - Nichterfüllung einer wichtigen Aufgabe
  - Großer wirtschaftlicher oder materieller Verlust

Katastrophe in Tschernobyl

Gliederung

- Einführung
- Aufgabenverteilung in automatisierten Systemen
- Mensch-Maschine-Schnittstelle
- Schlusswort



Tschernobyl: Der Ablauf

Gliederung

- Einführung
- Aufgabenverteilung in automatisierten Systemen
- Mensch-Maschine-Schnittstelle
- Schlusswort

- Für einen Test wurde der Energieausstoß des Reaktors verringert
- Zu hohe Konzentration von Xenon-135 dämpfte Ausstoß weiter
- Die Kontrollstäbe wurden über das sichere Maß herausgefahren
- Die Reaktionsgeschwindigkeit stieg sprunghaft an
- Sicherheitsmechanismen griffen nicht schnell genug
- Die Kühlflüssigkeit verdampfte und sprengte Reaktor und Schutz-mantel

[Quelle: www.wikipedia.org/Chernobyl_accident]

Tschernobyl: Ursachen der Katastrophe

Gliederung

- Einführung
- Aufgabenverteilung in automatisierten Systemen
- Mensch-Maschine-Schnittstelle
- Schlusswort

- Offizielle Begründung: Fehlerhaftes Design des Reaktors in Verbindung mit menschlichem Versagen
  - Unzureichend ausgebildetes Personal mit mangelhaftem Wissen über die Funktionsweise des Reaktors
  - Mangelhafte Kommunikation zwischen den Sicherheitsverantwortlichen und den für das Experiment zuständigen Mitarbeitern
  - Verhaltensweisen des Reaktors unterlagen zum Teil der militärischen Geheimhaltung
  - Sicherheitssysteme wurden außer Kraft gesetzt oder umgangen um das Experiment durchführen zu können

THERAC-25

Gliederung

- Einführung
- Aufgabenverteilung in automatisierten Systemen
- Mensch-Maschine-Schnittstelle
- Schlusswort

- 'Medical Linear Accelerators' zur Tumorbehandlung
- Vorläufer THERAC-6 und THERAC-20 mit mechanischen Sicherungen
- Software-Routinen ersetzen diese Sicherungen bei THERAC-25
- Systemfehler führten zu massiver Überdosierung der Strahlung

THERAC-25: Ursachen

Gliederung
- Einführung
- Aufgabenverteilung in automatisierten Systemen
- Mensch-Maschine-Schnittstelle
- Schlusswort

- Vereinfachung der Dateneingabe
- Vollständiges Vertrauen in korrekte Funktionsweise der Software
- Kryptische Fehlermeldungen ohne Erklärung
- Häufiges Auftreten von Fehlersituationen ohne reproduzierbare Ursachen

[Quelle: N.Leveson, Safeware: System Safety & Computers]

Übersicht

Gliederung
- Einführung
- Aufgabenverteilung in automatisierten Systemen
- Mensch-Maschine-Schnittstelle
- Schlusswort

- Einführung ins Thema
- ➔ Aufgabenverteilung in automatisierten Systemen
- Mensch-Maschine-Schnittstelle
- Schlusswort

Menschliches Versagen?

Gliederung
- Einführung
- Aufgabenverteilung in automatisierten Systemen
- Mensch-Maschine-Schnittstelle
- Schlusswort

- Ergebnisse von Studien
  - Bis zu 85% aller Arbeitsunfälle durch ‚unsicheres Handeln' der Arbeiter verursacht
  - Zwischen 60 und 80% aller Unfälle durch einen Kontrollverlust des Operators über die systeminhärenten Energien verursacht
- Aber
  - In 75% der untersuchten Fälle gingen den Handlungen des Operators verschiedenste Fehlfunktionen der Produktions- und Sicherheits-kontrollsysteme voraus

[Quelle: N.Leveson, Safeware: System Safety & Computers, pp92]

Rolle des Menschen in automatisierten Systemen (1)

Gliederung
- Einführung
- Aufgabenverteilung in automatisierten Systemen
- Mensch-Maschine-Schnittstelle
- Schlusswort

- Der Mensch als Aufpasser
  - Echtzeit-Überwachung eines Computersystems
  - Wissen über korrektes Soll-Verhalten notwendig
  - Zuviel Information
  - Mangel an Aufmerksamkeit
  - . . .
- Der Mensch als Backup
  - Mangelnde Erfahrung im Umgang mit dem System
  - Zu viele Entscheidungen in zu kurzer Zeit zu fällen
  - Wechsel von passivem Beobachten zu aktivem Eingreifen erfordert 'Aufwärmzeit'
  - . . .
- Der Mensch als Partner
  - Der Mensch erhält die Aufgaben, die nicht zu automatisieren sind
  - Die verbleibenden Aufgaben sind extrem komplex
  - Für Verständnis der Funktionsweise des Systems fehlt Training
  - . . .

Rolle des Menschen in automatisierten Systemen (2)

- Warum keine vollständige Automatisierung?
  - Es ist unmöglich in einem komplexen System alle Eventualitäten bereits während des Entwurfs zu berücksichtigen
  - Computer können nur auf vorhergesehene Ereignisse reagieren
  - Menschen sind anpassungsfähig, sie können sich auf verschiedenste Situationen einstellen

Human Factors Analysis

- Experten beschreiben das System
  - Für jede Phase des Systemlebenszyklus
  - Vom Blickwinkel ihrer jeweiligen Disziplin aus
  - Unter Berücksichtigung aller anderen Disziplinen
  - Wo, Wie und Wann nimmt die Disziplin Einfluss auf das System
- HF-Verantwortlicher ordnet Material entsprechend der Relevanz zur Aufgabenteilung zwischen Mensch und Maschine zu
- Beschreibungen müssen früh eingebracht und während des gesamten Systementwurfs berücksichtigt werden!

Beispiel: Flugleitsystem

- Konzept ‘Shared Controll’:
  - Alleinige Verantwortung liegt beim Menschen
  - Computer kann auf Anordnung des menschlichen Operators zur Unterstützung herangezogen werden
  - Nur simpelste Aufgaben werden komplett automatisiert
- Aber:
  - Explizite Aufgabenzuteilung durch den Operator erhöht dessen Arbeitsbelastung
  - Keine Erhöhung der Flugdichte pro Operator

Übersicht

- Einführung ins Thema
- Aufgabenverteilung in automatisierten Systemen

➔ Mensch-Maschine-Schnittstelle

- Schlusswort

Anforderungen ans System

Gliederung
- Einführung
- Aufgabenverteilung in automatisierten Systemen
- Mensch-Maschine-Schnittstelle
- Schlusswort

- Das System muss den Benutzer...
  - ... mit notwendigen Informationen versorgen
  - ... vor möglichen (System-) Fehlern warnen
  - ... von zu hoher Arbeitslast befreien
- Das System darf den Benutzer...
  - ... nicht mit Informationen überfluten
  - ... nicht mit Fehlalarmen belästigen
  - ... nicht unterfordern, da sonst dessen Wachsamkeit und Fertigkeiten nachlassen

Mensch-Maschine-Schnittstelle (1)

Gliederung
- Einführung
- Aufgabenverteilung in automatisierten Systemen
- Mensch-Maschine-Schnittstelle
- Schlusswort

- Es existieren viele Richtlinien und Vorschläge für den Entwurf der Mensch-Maschine-Schnittstelle (engl. Human-Machine Interface, HMI)
  - Keine dieser Richtlinien ist Allgemein gültig
  - In manchen Fällen widersprechen sie einander
  - Trade-off zwischen verschiedenen Zielen (z.B. Sicherheit vs. Benutzerfreundlichkeit)
  - Einführung von Computern hat den HMI-Entwurf sehr viel komplizierter gemacht

Mensch-Maschine-Schnittstelle (2)

Gliederung
- Einführung
- Aufgabenverteilung in automatisierten Systemen
- Mensch-Maschine-Schnittstelle
- Schlusswort

- Die Anpassungsfähigkeit des Menschen führt unausweichlich dazu, dass er Fehler macht!
- Fehlertolerante Systeme
  - Fehler müssen erkennbar sein
  - Fehler müssen behoben werden können

Mensch-Maschine-Schnittstelle (3)

Gliederung
- Einführung
- Aufgabenverteilung in automatisierten Systemen
- Mensch-Maschine-Schnittstelle
- Schlusswort

- Sicherheitskritische Bedienfehler verhindern
  - Sicherheitskritische Befehlsfolgen müssen jederzeit abgebrochen werden können
  - Es sollte mehrere Wege geben um zu einem Ziel zu gelangen
  - Potentiell gefährliche Befehle sollten in mehrere Schritte unterteilt werden
  - Kritische Abfragen dürfen nicht umgehbar sein
  - Stereotypen sollten beachtet werden

Mensch-Maschine-Schnittstelle (4)

Gliederung

- Einführung
- Aufgabenverteilung in automatisierten Systemen
- Mensch-Maschine-Schnittstelle
- Schlusswort

- Feedback
  - Der Nutzer muss die Effekte seiner Aktionen erkennen können
  - Er benötigt Informationen um sein gedankliches Modell des Systems auf den neuesten Stand zu bringen
  - Er muss in der Lage sein Fehler im automatischen System zu erkennen

Mensch-Maschine-Schnittstelle (5)

Gliederung

- Einführung
- Aufgabenverteilung in automatisierten Systemen
- Mensch-Maschine-Schnittstelle
- Schlusswort

- Warnungen
  - Warnungen müssen unterscheidbar sein
  - Verschiedene Arten von Warnungen (akustisch, optisch)
  - Abstufungen entsprechend der möglichen Folgen
  - Warnungen eindeutig machen
  - Zu viele Warnungen mindern Aufmerksamkeit
  - Fehlalarme sind stark kontraproduktiv

Mensch-Maschine-Schnittstelle (6)

Gliederung
- Einführung
- Aufgabenverteilung in automatisierten Systemen
- Mensch-Maschine-Schnittstelle
- Schlusswort

- Darstellung
  - Auswahl der Informationen beeinflusst Entscheidungen
  - Anzeigen an menschliches Wahrnehmungsverhalten anpassen
  - Inhaltlich zusammengehöriges auch gruppieren
  - Gleiche Sachverhalte identisch darstellen
  - Gängige Standards beachten
  - Relative Anordnung > Form > Farbe > Beschriftung

Übersicht

Gliederung
- Einführung
- Aufgabenverteilung in automatisierten Systemen
- Mensch-Maschine-Schnittstelle
- Schlusswort

- Einführung ins Thema
- Aufgabenverteilung in automatisierten Systemen
- Mensch-Maschine-Schnittstelle

➔ Schlusswort

Schlusswort

Gliederung
- Einführung
- Aufgabenverteilung in automatisierten Systemen
- Mensch-Maschine-Schnittstelle
- Schlusswort

- Human Factors ist nicht nur ein Thema sondern ein enorm umfangreicher Themenkomplex
- Beeinflusst durch Biologie, Psychologie, Soziologie, ...
- Gestaltung von User Interfaces, Arbeitsplätzen, Prozessen, ...
- Forschung und Weiterentwicklung in allen Bereichen

Schlusswort

Gliederung
- Einführung
- Aufgabenverteilung in automatisierten Systemen
- Mensch-Maschine-Schnittstelle
- Schlusswort

Nummer Eins der 10 goldenen Regeln der FAA:

Honor the user

Technische Universität Kaiserslautern

Software ist anders — Warum Risikomanagement eine Herausforderung für das Software Engineering ist

Florian Munz

WS 2004/05 Seminar „Risikomanagement für sicherheitskritische Softwaresysteme“
TU Kaiserslautern, AG Software Engineering, Prof. Dr. Dieter Rombach
Kaiserslautern, 3. Februar 2005

AG·SE

Fraunhofer IESE Institut Experimentelles Software Engineering

„Software is a strange product”

Les Belady

Das Wesen von Software

Complexity

Conformity

Software is a strange product

Changeability

Invisibility

Lokalitäts-Prinzip

Lokalität von	Nicht-Software	Software
Raum	Auftreten des Fehlers am Entstehungsort	Beliebig weit entfernt vom Entstehungsort
Zeit	Sofort sichtbares Fehlverhalten	Unvorhersehbarer späterer Zeitpunkt
Wirkung	analoge Auffassung ⇒ proportionale Reaktion	Der kleinste Fehler kann katastrophale Auswirkungen haben

Komposition und Dekomposition

Gliederung
- Software ist anders
- Traditionelle Ansätze für Software?
- Risikomanagement des Software Eng.
- Zusammenfassung

- Kompositions-Prinzip allgemein
  - Zuverlässigkeit der einzelnen Komponenten
    → Zuverlässigkeit des Gesamt-Systems
- Divide & Conquer
  - Keine formale Methode
    - Erfahrung
    - Heuristiken
- Dieses Prinzip gilt nicht für Software-Systeme!
  - Zwei Komponenten stören sich wechselseitig oder kompensieren ihre Fehler
- Kann man die Zuverlässigkeit von Software im Voraus wissen?
  - Nur durch ein Monitoring im laufenden Betrieb
  - Begrenzte Möglichkeiten, Qualität in der Entwicklung zu beweisen!

Komplexität

Gliederung
- Software ist anders
- Traditionelle Ansätze für Software?
- Risikomanagement des Software Eng.
- Zusammenfassung

„Nur noch ein weiteres Feature ..."

- Was ist komplexer?
  - Textverarbeitung oder Supertanker
  - Datenbank oder ein 100-Stock-Hochhaus
- Wiederverwendung
  - durch Referenz anstatt Kopie desselben Bauteils
- Komplexität wächst überproportional
- Komplexitäts/Funktionalitäts-Umkehrung
  - "Easy to Use" bedeutet mehr interne Komplexität
  - Operationale Komplexität im Widerspruch mit interner Komplexität

Safety Limits

Gliederung

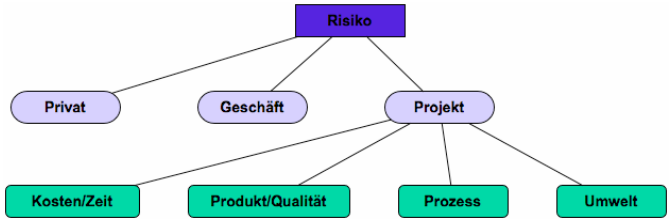
- Software ist anders
- Traditionelle Ansätze für Software?
- Risikomanagement des Software Eng.
- Zusammenfassung

- Synthese-Phase
  - Make it happen
- Analyse-Phase
  - Explain why it worked
- Gesucht: Nakamura's Law
  - Komplexität messen → Geeignete Grenzen vorausberechnen
- Warum existieren keine expliziten Safety Limits?

„Be conservative"

Können Ansätze für traditionelle technische Systeme auf Software übertragen werden?

Definitionen



- Risiko-Management
  - Kosten
  - Zeit

→ **Projekt**

- Gefahren-Management
  - Produkt/Qualität
  - Umwelt

→ **Produkt**

Traditionelle Ansätze

Gliederung
- Software ist anders
- Traditionelle Ansätze für Software?
- Risikomanagement des Software Eng.
- Zusammenfassung

- Übertragung existierender Ansätze wünschenswert
  - Ausgereiftere und detailliertere Methoden
  - Analyse-Phase häufig sehr spezifisch
- Strukturorientiert (Systemwissen)
  - Komplexes Zusammenwirken der Systemkomponenten
  - Kompositionsprinzip gilt nicht für Software
  - → In der Realität nicht so leicht übertragbar
- Problemorientiert (Domänenwissen)
  - Anwendungszentriert werden Risiken/Gefahren abgeleitet
  - Post-Mortem
  - → Erste Versuche, Fokus auf Produkt fehlt häufig

Lösung in sinnvoller Kombination der Methoden?

Welche Risikomanagement-Methoden gibt es für das Software Engineering?

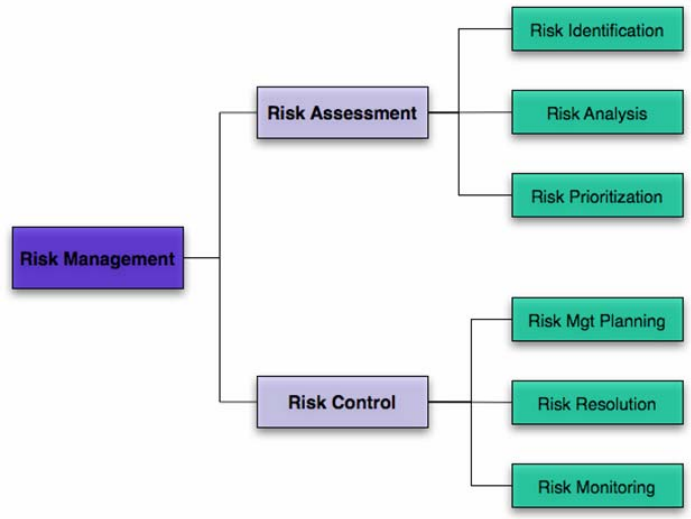
Risikomanagement im Software Engineering

Gliederung
- Software ist anders
- Traditionelle Ansätze für Software?
- Risikomanagement des Software Eng.
- Zusammenfassung

- Boehm (1989)
  - Fokus auf Kosten und Zeit
  - Grundlage vieler Methoden
- Riskit (1996)
  - qualitative Risikobewertung
  - unterschiedliche Perspektiven
  - definierter Prozess
- SEI (1990)
  - Continous Risk Management
  - ab CMM Level 2
- ...

Risikomanagement nach Boehm

Gliederung
- Software ist anders
- Traditionelle Ansätze für Software?
- Risikomanagement des Software Eng.
- Zusammenfassung



Und wie werden diese Ansätze den Besonderheiten von Software gerecht?

- Top-5 nach DeMarco
  - –Schedule flaw
  - –Requirements inflation
  - –People turnover
  - –Specifications breakdown
  - –Under-Performance

Risikomanagement des SE übertragen

- Zu allgemeine Risiko-Identifikation
  - – Checklisten, Erfahrung aus gleichartigen Projekten
- Wird Software-Systemen nicht wirklich gerecht
  - – Keine Berücksichtigung der Gefahren
- Prozesse im traditionellen Engineering ausgereifter
  - – Risikomanagement wird gelebt
  - – Spezielle Methoden für jeden Bereich
- Software-Systeme sind anders?
  - – Komplexität gibt es auch in anderen technischen Systemen
  - – Ideen und Ansätze wären übertragbar
- Solange es keine speziellen Methoden gibt, würde eine Übertragung eine Verschlechterung bedeuten

Zusammenfassung

Gliederung
- Software ist anders
- Traditionelle Ansätze für Software?
- Risikomanagement des Software Eng.
- Zusammenfassung

- Software ist anders
  - Komplexität, Lokalitäts-Prinzip, Komposition
- Traditionelle Methoden lassen sich nicht einfach übertragen
- Denkweise und Ideen können übernommen werden
  - Qualitätsbewusstsein
- Risikomanagement muss gelebt werden
  - Methoden des SE sind ein erster Schritt
- Steigende Komplexität und Einsatzbereiche erfordern eine permanente Verbesserung der Methoden

There is no Silver Bullet

Vielen Dank für Ihre Aufmerksamkeit

Dokumenten Information

Titel:	Seminar: Risikomanagement für sicherheitskritische Software-Systeme Wintersemester 2004/05
Datum:	Januar 2005
Report:	IESE-031.05/D
Status:	Final
Klassifikation:	Öffentlich