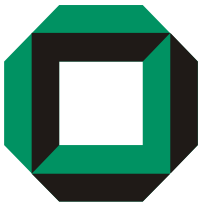
Universität Karlsruhe – Fakultät für Informatik – Bibliothek – Postfach 6980 – 76128 Karlsruhe

Software Zertifizierung

Herausgeber:
Thomas Goldschmidt, Henning Groenda, Klaus Krogmann,
Michael Kupperberg, Anne Martens, Christoph Rathfelder,
Ralf Reussner, Johannes Stammel
Autoren:
Jakob Blomer, Fabian Brosig, Andreas Kreidler, Jens
Küttel, Achim Kuwertz, Grischa Liebel, Daniel Popovic,
Michael Stübs, Alexander M. Turek, Christian Vogel,
Thomas Weinstein, Thomas Wurth

Interner Bericht 2008-4



Universität Karlsruhe

Fakultät für Informatik

ISSN 1432 – 7864

Universität Karlsruhe (TH)
Forschungsuniversität • gegründet 1825

Software-Zertifizierung

Seminar im Wintersemester 2007/2008

Universität Karlsruhe (TH)
Fakultät für Informatik
Institut für Programmstrukturen und Datenorganisation
Lehrstuhl für Software-Entwurf und -Qualität
Prof. Dr. Reussner

`http://sdq.ipd.uni-karlsruhe.de`

Jakob Blomer, Fabian Brosig, Thomas Goldschmidt, Henning Groenda, Andreas Kreidler,
Klaus Krogmann, Michael Kupperberg, Jens Küttel, Achim Kuwertz, Grischa Liebel, Anne Martens,
Daniel Popovic, Christoph Rathfelder, Ralf Reussner, Johannes Stammel, Michael Stübs,
Alexander M. Turek, Christian Vogel, Thomas Weinstein, Thomas Wurth

Vorwort

Systematische Qualitätssicherung gewinnt im Rahmen des globalen Wettbewerbs auch in der Software-Entwicklungsbranche zunehmend an Bedeutung. Vor allem auf dem Weg zur Software-Industrialisierung bzw. zu einer ingenieurmäßigen Software-Entwicklung ist eine durchgängige Qualitätssicherung unabdingbar. Zertifizierungen bieten hierbei die Möglichkeit, die Einhaltung bestimmter Standards und Kriterien durch unabhängige Dritte überprüfen und bescheinigen zu lassen, um die Qualität eines Produktes oder Entwicklungsprozesses zu belegen.

Zertifizierungen können sich sowohl auf Produkte und Prozesse als auch auf die Ausbildung und das Wissen von Einzelpersonen beziehen. Da Zertifikate durch unabhängige Prüfinstanzen ausgestellt werden, wird Zertifikaten und deren überprüfbaren Aussagen im Allgemeinen ein deutlich höheres Vertrauen entgegengebracht als Qualitätsversprechen von Software-Herstellern selbst. Unternehmen, die ihre Prozesse beispielsweise nach CMMI zertifizieren lassen, können damit ihre Fähigkeiten unter Beweis stellen, Projekte erfolgreich und mit vorhersagbarer Qualität abschließen zu können. Neben dem Nachweis entsprechender Zertifikate als Diversifikationsmerkmal gegenüber Mitbewerbern können Zertifikate über die Einhaltung von Standards auch durch den Gesetzgeber vorgeschrieben werden. Ein Beispiel hierfür sind Zertifikate aus Hochsicherheitsbereichen wie Atomkraftwerken.

Das Seminar wurde wie eine wissenschaftliche Konferenz organisiert: Die Einreichungen wurden in einem zweistufigen Peer-Review-Verfahren begutachtet. In der ersten Stufe wurde eine Begutachtung der studentischen Arbeiten durch Kommilitonen durchgeführt, in der zweiten Stufe eine Begutachtung durch die Betreuer. In verschiedenen *Sessions* wurden die Artikel an zwei *Konferenztagen* präsentiert. Die besten Beiträge wurden durch *best paper awards* ausgezeichnet. Diese gingen an Fabian Brosig für seine Arbeit *Cost Benefit Analysis Method (CBAM)*, an Jakob Blomer für die Arbeit *Zertifizierung von Softwarebenchmarks* und an Grischa Liebel für die Arbeit *SWT - Das Standard Widget Toolkit*, denen hiermit noch einmal herzlich zu dieser herausragenden Leistung gratuliert wird. Ergänzend zu den Vorträgen der Seminarteilnehmer wurde ein eingeladener Vortrag gehalten. Herr Dr. Dirk Feuerhelm von der 1&1 Internet AG gab dabei dankenswerterweise in seinem Vortrag mit dem Thema *Softskills? Ist das objektorientiert oder modellgetrieben?* einen Einblick in die Aufgaben als Leiter der Software-Entwicklung.

Gliederung

Die Themen dieses Seminars spiegeln die Frage- und Problemstellungen wieder, die sich bei der Bewertung und Zertifizierung von Software-Systemen ergeben. Dieser technische Bericht gliedert sich dabei wie folgt. Zunächst werden grundlegende Ansätze vorgestellt, welche zur Bewertung von Software-Architekturen eingesetzt werden können. Da Reifegradmodelle ein beliebtes Mittel sind, um

die Qualität anhand von Indikatoren zu bestimmen, wird im Anschluss an die Architektur-Bewertungsmethoden zunächst ein allgemeiner Überblick über Reifegradmodelle gegeben. Das Reifegradmodell SPICE wird dabei als Beispiel eines Prozess-Reifegradmodelles nochmals detailliert vorgestellt. Im Folgenden wird zunächst eine Überblick über die Vorteile der Zertifizierung von Software gegeben, welcher durch die Vorstellung mehrere Zertifizierungsansätze aus unterschiedlichen Bereichen ergänzt wird. Der beste Beitrag des parallel zum Seminar laufenden Proseminars *Software-Entwicklung mit Eclipse* über das Entwicklungs-Framework SWT bildet den Abschluss dieses Berichts.

Dank

Wir möchten uns an dieser Stelle bei allen Teilnehmern des Seminars für ihre engagierte Mitarbeit sehr herzlich bedanken. Ein mehrstufiger Begutachtungsprozess bestehend aus Peer–Reviews sowie Gutachten durch die Betreuer ermöglichte die Auswahl qualitativ hochwertiger Artikel. Insgesamt wurden 11 Ausarbeitungen für diesen technischen Bericht angenommen. Auf der Homepage[1] zu diesem Seminar sind daneben auch die Vortragsfolien der Seminarteilnehmer zu finden, die auf den Sessions der Konferenz des Seminars vorgestellt wurden.

Ganz besonders möchten wir uns bei Herrn Dr. Dirk Feuerhelm von der 1&1 Internet AG für seinen lebhaften Vortrag bedanken, der uns eine ganz andere Perspektive auf die Software-Entwicklung ermöglicht hat.

Karlsruhe, Februar 2008

Thomas Goldschmidt
Henning Groenda
Klaus Krogmann
Michael Kupperberg
Anne Martens
Christoph Rathfelder
Ralf Reussner
Johannes Stammel

[1] http://sdqweb.ipd.uka.de/wiki/Seminar_Software-Zertifizierung_WS0708

Inhaltsverzeichnis

Software-Zertifizierung

SAAM (Software Architecture Analysis Method)

Christian Vogel

Betreuer: Thomas Goldschmidt

Zusammenfassung Diese Arbeit beschäftigt sich mit der Software Architecture Analysis Method (SAAM). Mit dieser Methode analysiert man eine Softwarearchitektur und versucht Schwachstellen oder Risiken in der Architektur frühzeitig zu entdecken. In dem folgenden Artikel werden zuerst generelle Grundlagen der Softwarearchitektur und ihrer Qualität dargestellt. Im Weiteren wird die Theorie der SAAM mit ihren Zielen, ihrem Ablauf und Weiterentwicklungen detailliert beschrieben und bewertet. Zum Schluss wird die vorgestellte Methodik an einem praktischen Beispiel ausgeführt und schließlich die gewonnenen Erkenntnisse zusammengefasst.

1 Einführung

In den vergangenen Jahrzehnten ist Hardware immer kleiner und leistungsfähiger geworden. Gleichzeitig stiegen auch die Leistungsfähigkeit und Komplexität der Software, die diese Hardware ausnutzt und zum Leben erweckt. Programme mit Hunderttausenden oder Millionen Zeilen Code sind heute keine Seltenheit mehr [ClKK05]. Seit den 90er Jahren versucht man diese Komplexität in den Griff zu bekommen durch die Verwendung von Entwurfsmustern und durch das explizite Erstellen und Dokumentieren der Softwarearchitektur. Deren Verwendung allein kann aber noch nicht garantieren, dass die erstellte Software qualitativ hochwertig ist und vor allem dass sie die an sie gestellten Anforderungen erfüllt [BaCK98, S.32]. Durch verschiedene Verfahren zur Analyse der Architektur einer Software versucht man möglichst früh im Entwicklungsprozess Aussagen über die Qualität der Software und die Erreichung der an sie gestellten Anforderungen herauszubekommen. Denn je früher man Fehler oder Schwachstellen beim Erstellen oder Modifizieren einer Software findet, desto billiger und einfacher lassen sie sich beheben [ClKK05, S.23].

Diese Arbeit beschäftigt sich mit der Software Architecture Analysis Method, kurz SAAM. Die SAAM war die erste standardisierte und dokumentierte Methode zur Analyse von Softwarearchitekturen. Im den folgenden 3 Kapiteln werden die Grundlagen der Softwarearchitekturanalyse vorgestellt. Was genau versteht man eigentlich unter einer Softwarearchitektur? Mit welchen Merkmalen kann man die Qualität von Software messen? Wie kann man mit konkreten Szenarien eine Softwarearchitektur auf die Qualitätsmerkmale hin untersuchen. Im fünften Kapitel wird beschrieben, wie die SAAM genau funktioniert, welche Vor- und Nachteile sie hat und welche verschiedenen Abwandlungen es gibt. Nach der Theorie der vorangegangenen Kapitel wendet sich Kapitel sechs der Praxis zu

und zeigt ein Beispiel für den praktischen Einsatz der SAAM. Kapitel sieben fasst die gewonnenen Erkenntnisse abschließend zusammen.

2 Softwarearchitektur

2.1 Definition

Softwarearchitektur ist noch ein relativ junges Feld der Informatik. Daher gibt es auch noch keine allgemein anerkannte exakte Definition des Begriffes. Um Softwarearchitekturen bewerten zu können, muss man jedoch eine Definition bzw. ein gemeinsames Verständnis von Softwarearchitektur voraussetzen.

Bei Bass, Clements, Kazman [BaCK98] wird Softwarearchitektur folgendermaßen definiert:

„The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them."

Man kann also festhalten, dass es sich bei einer Architektur um eine Abstraktion eines Softwaresystems handelt. Das System wird nur auf der Ebene von Komponenten und Modulen betrachtet. Die Details in den Komponenten sind auf dieser Abstraktionsebene nicht relevant. Bedeutsam sind aber die Schnittstellen der Komponenten und ihre Beziehungen untereinander. Beziehungen können sowohl Abhängigkeiten, Benutzt-Beziehungen, aber auch zeitlich begrenzte Beziehungen sein.

Um die Architektur komplexer Systeme vollständig und anschaulich beschreiben zu können, benötigt man in der Regel mehrere sogenannte Sichten. Jede Sicht fokussiert unterschiedliche Aspekte des darzustellenden Systems.

2.2 Ziele

Nachdem geklärt ist, was eine Softwarearchitektur beinhaltet, stellt sich natürlich noch die Frage, welchen Sinn und Zweck eine Softwarearchitektur hat. Hier kann man drei Hauptaspekte unterscheiden:

Übersicht und Kommunikationsgrundlage

Die Softwarearchitektur bildet eine Kommunikationsgrundlage für Interessenvertreter. Als Interessenvertreter bezeichnet man in diesem Zusammenhang alle Personen(-gruppen), die mit dem Softwaresystem aus den verschiedensten Blickwinkeln zu tun haben. Das sind z.B. der Architekt der Software, die Entwickler, welche die Software entwerfen oder modifizieren müssen, die Anwender, die mit der Software arbeiten müssen, oder der Kunde als Auftraggeber der Software. All diese Personen haben unterschiedliche Hintergründe und Vokabulare. Für sie ist die Architektur eine gemeinsame verständliche Darstellung des Systems. In der Architektur können alle Beteiligten ihre Anforderungen an das System überprüfen und gemeinsam über Änderungen diskutieren. Um die Verständlichkeit

auch bei komplexeren Systemen zu gewährleisten, bedient man sich der vorher schon angesprochenen Sichten. Die wichtigsten Sichten sind:

- **Funktionale oder logische Sicht:** In dieser Sicht werden die Funktionen des Systems, ihre Abhängigkeiten und die Datenflüsse zwischen ihnen gezeigt. Die Partitionierung der Gesamtfunktionalität ist vor allem für spätere Änderungen interessant. An welcher Stelle können neue Funktionen eingebunden werden, auf welche Abhängigkeiten muss man achten, falls Funktionen entfernt werden?
- **Nebenläufige Sicht:** Sie zeigt, aus welchen Prozessen und Threads das System besteht, wie diese im Zeitablauf interagieren und auf welche Ereignisse und Signale sie reagieren. Vor allem bei Fragestellungen zu Leistung, Verfügbarkeit und Verteilung liefert diese Sicht wichtige Eigenschaften des Systems.
- **Codesicht:** Hier wird die Architektur aus Sicht eines Programmierers gezeigt. Wie verteilen sich Klassen, Objekte, Prozeduren oder Funktionen auf die verschiedenen Klassen oder Module? Wie sehen die Abhängigkeiten und Vererbungen zwischen den einzelnen Modulen aus? Im objektorientierten Fall entspricht diese Sicht UML Klassendiagrammen.
- **Entwicklungssicht:** Diese Sicht zeigt die Architektur aus der Sicht eines Code Repositories. Welche Dateien gibt es, wo liegen diese, welche Abhängigkeiten bestehen zwischen diesen Dateien? Wie der Name schon sagt, ist diese Sicht für die Entwicklungs- und Änderungsphase der Software hilfreich. Wie kann die Arbeit an den Dateien verteilt werden, welche Funktionalität kann in welcher Reihenfolge getestet und ins Gesamtsystem integriert werden?
- **Physische Sicht:** In der physischen Sicht wird die Verteilung der Software auf die verschiedenen Hardwarekomponenten und die Verknüpfung dieser Komponenten gezeigt. Hardware könnte bei einem komplexen System z.B. Sensoren, Aktoren oder Rechner sein.

Jede Personengruppe wird zwar hauptsächlich eine Sicht verwenden, kann aber auch die anderen benötigen. Erst, wenn man die verschiedenen Sichten übereinander legt, kann man das System als Ganzes erfassen. [ClKK05]

Stoßen neue Mitglieder zu einem Softwareentwicklerteam, bietet es sich ebenfalls an, diesen anhand der Architektur ein Gefühl und Verständnis für das zu entwickelnde System zu vermitteln.

Spezielle Architekturbeschreibungssprachen hatten nie die nötige Verbreitung gefunden, um sich durchsetzen. Stattdessen greift man in der Praxis heute auf die weitverbreitete Sprache UML zurück, wenn man Softwarearchitekturen beschreiben will. Ursprünglich wurde UML entwickelt, um den objektorientierten Entwurf zu beschreiben und konnte Softwarearchitekturen nur bedingt beschreiben. Seit UML Version 2 wurde dies aber erheblich verbessert [Lang06].

Festlegung der ersten Design Entscheidungen

Die Architektur ist die Abbildung der ersten und grundlegenden Design Entscheidungen eines Softwaresystems. Trotz dieser frühen Phase im Entwicklungsprozess kann man an ihr aber schon fast alle Qualitätsmerkmale der späteren Anwendung ablesen [ClKK05, S.5]. Ist zum Beispiel Leistung ein Hauptaugenmerk der Software, so muss der Architekt besonders auf die Ausführungszeiten der Komponenten und das Zeitverhalten bei der Prozesskommunikation achten. Soll die Software besonders einfach modifizierbar sein, so muss man darauf achten Komponenten so aufzuteilen, dass möglichst wenig Abhängigkeiten zwischen ihnen entstehen und man sie leicht austauschen kann. Oder stellt die Software hohe Anforderungen an die Ausfallsicherheit, müssen Komponenten in der Architektur mehrfach ausgelegt sein und ein Protokoll vorhanden sein, das regelt was bei Ausfall einer Komponente passiert. Eine gute Softwarearchitektur alleine ist noch kein Garant für eine gute Software, aber wenn die Architektur Mängel ausweist, lassen sich diese später in der Regel nur mit immensem Aufwand korrigieren.

Durch die Aufteilung der Software in Komponenten in der Architektur wird auch das Prototyping erleichtert.[BaCK98, S.33] Dazu muss man nur das Rahmenwerk, welches die Architektur liefert, implementieren. Die Komponenten fügt man als „Dummys“ hinzu, die ihre Ergebnisse nicht berechnen, sondern aus Dateien oder Datenbanken auslesen. So kann man, dank der expliziten Definition einer Architektur, dem Kunden schon einen Prototyp zeigen und damit einen ersten Eindruck der Software vermitteln. Aber auch für die eigene Entwicklung hat dies Vorteile. Die Komponenten können geschrieben werden und sofort nach Fertigstellung und Komponententest in den Rahmen eingefügt und dort getestet werden.

Die Unterteilung einer Architektur in Komponenten und Module überträgt sich nicht nur auf die spätere Software, sondern spielt auch bei der Organisation des Entwicklungsprozesses eine Rolle. So gibt die Architektur vor, in welcher Reihenfolge und in welchen Arbeitspaketen die Software entwickelt wird. Bei gutem Design gibt es innerhalb einer Komponente viele Zusammenhänge, nach außen so wenig wie möglich. In Organisationen ist es meist ähnlich. Innerhalb eines Teams tauschen sich die Mitarbeiter aus und unterhalten sich, mit anderen Teams wesentlich weniger. Wenn man die Entwicklungsteams also anhand der Architektur zusammenstellt und somit die Organisationstruktur an die Architektur anpasst, unterstützt dies den Entwicklungsprozess. [ClKK05, S.11-12]

Die Architektur eines Softwaresystems stellt ein System auf einer höheren Abstraktionsebene relativ übersichtlich dar. Aus diesem Grunde vereinfacht sie Änderungen an der Software zu planen und durchzuführen. Für große Softwaresysteme ist dies ein enorm wichtiger Punkt und Kostenfaktor. Ca. 80 Prozent der Kosten einer Software fallen nach der eigentlichen Entwicklung an, sind also hauptsächlich Kosten für Wartung und Modifikation der Software [BaCK98, S.32]. Bei einer Architektur werden Änderungen in drei Typen eingeteilt: lokale, nicht-lokale und architektonische Änderungen. Lokale Änderungen betreffen nur eine Komponente. Diese wird geändert oder ausgetauscht. Das sind die einfachs-

ten Änderungen. Nicht-lokale Änderungen erfordern Änderungen an mehreren Komponenten, die Architektur bleibt aber intakt. Bei architektonischen Änderungen wird, wie der Name schon sagt, die Architektur selbst verändert, also beispielsweise das Kommunikationsverhalten zwischen den Komponenten. Solche Änderungen ziehen sich meist durch das gesamte System und machen dort überall Änderungen oder Anpassungen nötig.

Eine Architektur sollte daher mit Weitsicht geplant werden und mögliche zukünftige Änderungen miteinbeziehen. Idealerweise ist jede zukünftige Änderung oder Erweiterung des Systems eine lokale Änderung. Ist die Architektur selbst erst einmal festgelegt, sind Änderungen an ihr, wie oben beschrieben, nur noch mit immensem Aufwand möglich. Dies muss man immer im Hinterkopf behalten. Die Entscheidung für eine Architektur ist oft auch ein Kompromiss zwischen verschiedenen Qualitätsmerkmalen. Hat man sich aber für die Architektur entschieden, muss man sich ihrer Schwachstellen bewusst sein. Man wird sie (fast) nicht mehr ändern können, wenn mit der Implementierung begonnen wurde.

Wiederverwendbarkeit

Software ist in den letzten Jahren nicht nur immer komplexer geworden, sie muss auch immer schneller entwickelt werden. Ein Beispiel hierfür sind Handys. Produktlebenszyklen werden immer kürzer, es folgen immer neue Modelle. Alle diese Modelle brauchen Software, gute Software, denn die Konkurrenz ist groß. Die komplette Software für jedes Modell neu zu entwickeln ist zeitlich und finanziell unmöglich. Auch hier kann Softwarearchitektur helfen. Softwarearchitekturen kann man bei ähnlichen Anforderungen an das System wiederverwenden. Und im Gegensatz zu anderen Formen der Wiederverwendbarkeit, ist sie bei Softwarearchitekturen ein kommerzieller Erfolg. Wenn man die Softwarearchitektur wiederverwendet „erbt“ die neue Software, über die Architektur, deren Qualitätsmerkmale. Um eine breite Modellpalette anbieten zu können verwenden Handyhersteller heute ganze Produktlinien mit sich nur relativ geringfügig unterschiedlichen Modellen. Wenn man noch einen Schritt weiter geht, hilft die Softwarearchitektur auch hier. Bei der Erstellung der Softwarearchitektur unterscheidet man zwischen Kernfunktionen und produktspezifischen Funktionen. Die Kernfunktionen werden in der Architektur festgelegt und sind für alle Produkte der Familie gleich, die produktspezifischen Designentscheidungen werden offen gelassen. Durch diesen Kompromiss wird die Software in der Regel nicht optimal an das Produkt angepasst sein, aber das Konzept erhält doch eine große Flexibilität, die es erlaubt verschiedene Produkte mit verschiedenen Funktionen auszustatten. Durch diesen Ansatz spart man Geld und vor allem Entwicklungszeit, was dazu führt, dass man Produkte schneller auf den Markt bringen kann. Trotzdem kann man möglichst viele Kunden mit individuellen Produkten ansprechen. Dies ist einer der wichtigsten Erfolgsfaktoren eines Unternehmens. [ClKK05]

3 Qualitätsmerkmale

Für die Bewertung der Qualität einer Software und ihrer Architektur gibt es verschiedene Merkmale, anhand deren Ausprägung die Qualität gemessen werden kann. Diese Qualitätsmerkmale kann man in die drei Gruppen, zur Laufzeit erkennbare Qualitätsmerkmale des Systems, nicht zur Laufzeit erkennbare Qualitätsmerkmale des Systems und wirtschaftliche Qualitätsmerkmale, einteilen. Im Folgenden werden die wichtigsten Merkmale der drei Kategorien aufgeführt. [BaCK98, S.75-90]

3.1 Zur Laufzeit erkennbare Qualitätsmerkmale des Systems

- **Leistung:** Die Leistung bezeichnet die Reaktionszeit des Systems auf ein Ereignis oder die Anzahl der Ereignisse, die in einer bestimmten Zeit abgearbeitet werden kann. Sie war früher eines der wichtigsten Kriterien für Software, hat in letzter Zeit aber, mit steigenden Ausführungsgeschwindigkeit der Hardware, in der Bedeutung etwas abgenommen. Das Dilemma der Leistung ist, dass die Erhöhung jedes anderen Qualitätsmerkmals fast immer zu Lasten der Leistung geht. In der Architektur kann man die Leistung verbessern, indem man die Interprozesskommunikation optimiert oder die Komponenten so auf die Hardware verteilt, dass ein paralleles Arbeiten möglich wird.
- **Sicherheit:** Sicherheit ist ein Maß für die Fähigkeit eines Systems Unbefugten den Zugang zu verwehren und trotz Angriffen (z.B. Denial-of-Service) für autorisierte Nutzer verfügbar zu bleiben. Der Sicherheit kann man in der Architektur mit speziellen Komponenten z.B. für Authentifizierung und Verschlüsselung Rechnung tragen.
- **Verfügbarkeit:** Die Verfügbarkeit eines Systems ist ein Maß für die Wahrscheinlichkeit, dass das System arbeitet oder auf Anfragen antwortet. Sie wird gewöhnlich angegeben als Quotient aus verfügbarer Zeit und der Gesamtzeit. Will man die Verfügbarkeit eines Systems verbessern, so kann man redundante Komponenten in die Architektur einfügen.
- **Funktionalität:** Sie gibt an, ob alle geforderten Funktionen implementiert und ausführbar sind. Dies deckt indirekt auch die Korrektheit ab. Wenn alles funktioniert wie gefordert, muss auch das Ergebnis korrekt sein.

3.2 Nicht zur Laufzeit erkennbare Qualitätsmerkmale des Systems

- **Modifizierbarkeit:** Modifizierbarkeit ist die Eigenschaft eine Software möglichst schnell und kostengünstig ändern zu können. Sie ist direkt mit der Architektur verknüpft. Je besser die Komponenten aufgeteilt und gekapselt sind, desto leichter lässt sich die Architektur ändern. Den Aufwand, den eine Modifizierung verursacht misst man in der Anzahl der Komponenten, die man ändern muss. Dies muss nicht immer korrekt sein, da drei kleine Änderungen an drei Komponenten auch weniger Aufwand sein können als

eine große Änderung an einer Komponente, aber in der Regel ist die Anzahl der Komponenten proportional zum Änderungsaufwand. Bei größeren Softwaresystemen darf man auch nicht vergessen, dass jede geänderte Komponente ausführlich getestet werden muss, was zusätzlichen Aufwand pro Komponente verursacht. Die Bewertung von Architekturen aufgrund ihrer Modifizierbarkeit ist das primäre Einsatzgebiet der SAAM.
- **Portierbarkeit:** Portierbarkeit ist eigentlich eine besondere Form der Modifizierbarkeit und sagt aus, wie schnell und mit welchem Aufwand man eine Software an eine bestimmte Hard- oder Software Plattform anpassen kann. Portierbarkeit kann man in der Architektur erreichen, indem man eine zusätzliche Schicht einfügt, die das System von der Plattform entkoppelt.
- **Wiederverwendbarkeit:** Wiederverwendbarkeit ist die Fähigkeit Komponenten eines Systems in anderen Systemen wiederzuverwenden. Gemessen wird das Merkmal anhand der Kopplung der Komponenten. Je mehr Abhängigkeiten zwischen Komponenten bestehen, desto geringer ist die Wiederverwendbarkeit. Auch die Wiederverwendbarkeit kann man als Spezialfall der Modifizierbarkeit ansehen. Wenn man 40% der Komponenten eines Systems wiederverwenden will, kann man auch sagen, man behält das System und modifiziert die restlichen 60%. [BaCK98]

3.3 wirtschaftliche Qualitätsmerkmale

- **Kosten:** Für Unternehmen sind vor allem die Kosten eines Softwaresystems interessant. Meist ist hier ein Budget vorgegeben, das eingehalten werden muss. Reduzieren kann man die Kosten z.B., durch die Wiederverwendung schon existierender Komponenten.
- **Time to Market:** Nicht minder wichtig ist die Time to Market, also die Zeitspanne, die ein Produkt von der Idee bis zur Serienreife benötigt. Je schneller ein Produkt am Markt ist, desto mehr Chancen hat es sich durchzusetzen und Gewinne abzuwerfen. Auch auf diesen Faktor kann man mit der Architektur einwirken. Eventuell kann man Komponenten schon fertig kaufen, um die Entwicklungszeit zu verkürzen oder die Komponenten so aufbauen, das mehr Entwicklungsteams parallel daran arbeiten können. An diesen Beispielen sieht man das die Kosten und Entwicklungszeit einer Software stark miteinander zusammenhängen und sich gegenseitig beeinflussen. [BaCK98]

4 Szenarien

Im vorigen Kapitel wurden die Qualitätsmerkmale vorgestellt und wie man sie durch die Architektur beeinflussen kann. Dies reicht so aber noch nicht aus um eine Architektur konkret bewerten zu können [ClKK05, S.32]. Denn wie soll eine perfekte Architektur aussehen? Sie sollte wohl alle Qualitätsmerkmale zu 100% erfüllen. Und hier fangen die Probleme an. An etlichen Stellen des vorherigen Kapitels kann man sehen, dass sich die Qualitätsmerkmale gegenseitig beeinflussen und die Verbesserung eines Merkmals gleichzeitig eine Verschlechterung

bei den anderen hervorrufen kann. Also wird man nie alle Qualitätsmerkmale erfüllen können. Das zweite Problem ergibt sich daraus festzustellen, ab wann ein Merkmal als erfüllt gilt. Idealerweise liegen die Kosten für Software bei null. Das wird in der Praxis nicht erreichbar sein. Oder muss ein System auch bei einem Erdbeben und Stromausfall noch verfügbar sein? Die Antwort ist: „Es kommt drauf an!“ Ein Webserver muss sicher nicht unter solch extremen Bedingungen funktionieren, die Software medizinischer Geräte oder in Flugzeugen aber schon. Die Qualität einer Software wird zwar anhand der Qualitätsmerkmale gemessen, aber die Qualitätsmerkmale benötigen einen Kontext, damit sie sinnvoll angewendet werden können [BaCK98, S.192]. Dieser Kontext sind die Anforderungen die an ein System gestellt werden. Sind die Anforderungen bekannt, kann überprüft werden, ob eine Architektur hierfür gut oder weniger gut geeignet ist. Eine „gute“ Architektur kann bei anderen Anforderungen komplett versagen. Dies schränkt auch die Wiederverwendbarkeit auf Fälle ein, in denen die Anforderungen an die Softwaresysteme ähnlich sind.

Szenarien, die bei der SAAM und anderen Softwarearchitekturbewertungsverfahren eingesetzt werden, sind konkrete Anforderungen die ein Interessenvertreter an eine Architektur stellt. Anhand dieser Szenarien wird die Architektur bewertet. Bei Szenarien gibt es noch eine Unterscheidung zwischen direkten und indirekten Szenarien. Ein Szenario ist direkt, wenn die Software es ohne Änderung ermöglicht. Ein Anwendungsfall ist beispielsweise ein direktes Szenario. Ein indirektes Szenario ist eine Anforderung, welche die Software nicht unterstützt und für welche die Software angepasst werden muss. Ein Beispiel für ein indirektes Szenario ist ein Webserver, der zukünftig nicht nur statische HTML Seiten, sondern zukünftig auch PHP Seiten ausliefern können soll. [BaCK98, S.191-193]

Da das Haupteinsatzgebiet der SAAM bei der Bewertung der Modifizierbarkeit von Architekturen liegt hat man es hier meist mit indirekten Szenarien zu tun. Die Güte mit der ein indirektes Szenario erfüllt wird liegt dann darin, wie wenige Komponenten angepasst werden müssen, um das Szenario zu erfüllen. Direkte Szenarien spielen hauptsächlich beim Vergleich zwischen Architekturen eine Rolle. Angenommen es sollen die Architekturen zweier Webserver verglichen werden. Dann stellt man eine Reihe von Szenarien auf die wichtig sind und überprüft bei welcher Architektur wie viele Szenarien direkt oder indirekt sind. Die eine Architektur skaliert zum Beispiel gut und erfüllt das Szenario „100 Seitenaufrufe pro Sekunde“ direkt, bei der anderen sind Änderungen nötig um dies zu erfüllen.

5 SAAM

5.1 Beschreibung

Die Software Architecture Analysis Method ist eine der ältesten dokumentierten Evaluierungsmethoden. Sie wurde von Rick Kazman, Len Bass, Mike Webb und Gregory Abowd entwickelt und 1994 publiziert [Kazm96]. Damals sollte die SAAM den Beweis erbringen, dass Softwarearchitektur maßgeblich die Qualitätsmerkmale einer Software beeinflusst [Dobr02, S.641].

Das Ziel der SAAM ist es die Architektur einer Software zu verbessern und mögliche Schwachstellen offenzulegen bzw. die Risiken beim Softwareentwurf aufzuzeigen. [ClKK05, S.213-214]

Um dieses Ziel zu erreichen verwendet man Szenarien. Erstellt werden diese Szenarien von den Interessenvertreter des Softwaresystems, anhand der jetzigen und zukünftigen Anforderungen an das System. Danach werden die Szenarien nach Priorität gewichtet und auf die Architektur angewendet. Dadurch können Problembereiche in der Architektur sichtbar werden. Wenn mehrere Szenarien Änderungen an den gleichen Komponenten erfordern, dann ist die Komponente in der Architektur sehr wahrscheinlich noch nicht detailliert genug beschrieben oder innerhalb der Komponente herrscht nur schwache Kohäsion und sie sollte daher aufgeteilt werden. Wenn andersherum ein Szenario Änderungen an vielen Komponenten bewirkt, deutet dies meist ebenfalls auf ein Problem hin. In diesem Fall sind Funktionen der Software über das System verstreut, statt zentral an einer Stelle gekapselt zu sein.

Mit der gleichen Methodik kann man natürlich nicht nur eine Architektur bewerten und verbessern, sondern man kann auch mehrere Architekturen bewerten. Über die Aufwände die nötig sind, um alle Szenarien zu unterstützen, und die Gewichtung der Szenarien werden die Architekturen direkt vergleichbar [ClKK05, S.214].

Die SAAM kann sehr früh im Softwareentwicklungsprozess angewendet werden. Einzige Voraussetzung ist das Vorliegen einer detaillierten Beschreibung der Softwarearchitektur. Sollte die Architektur unzureichend oder fehlerhaft dokumentiert sein, tritt dies in den allermeisten Fällen bei der Durchführung von SAAM zutage und kann während des Prozesses verbessert werden. Idealerweise führt man die SAAM durch, bevor die Software implementiert wird. Sie kann aber auch später noch angewendet werden. In [Kazm96] wurde die SAAM auf eine bestehende Software angewandt, die bewertet werden sollte. Falls, wie in diesem Fall, für die Software keine Architekturbeschreibung existiert, muss diese nachträglich vor der Evaluierung erstellt werden.

5.2 Ablauf

Die SAAM ist eine relativ einfache Methode und kann fast ohne Vorkenntnisse angewendet werden. Daher bietet sie sich auch für Organisationen und Teams an, die zum ersten Mal eine Softwarearchitektur evaluieren. Der Zeitaufwand ist ebenfalls nicht sehr groß. Bei einer kleineren Architektur dauert die komplette Evaluierung etwa zwei Tage. Am ersten Tag werden die Szenarien erhoben und gewichtet. Am zweiten wird die Architektur gegen die Szenarien validiert und das Ergebnis präsentiert. Bei größeren Projekten kann dieser Prozess länger dauern und sich auf einige Tage verteilen. Um die SAAM anwenden zu können braucht man einen Moderator, der die Methodik der SAAM kennt, der die anderen Teilnehmer anleitet und durch den Evaluierungsprozess führt. Des Weiteren braucht man einen oder mehrere Softwarearchitekten, die den restlichen Teilnehmern die zu untersuchende Softwarearchitektur erklären und beschreiben

können. Als dritte Gruppe benötigt man die Interessenvertreter des Softwaresystems. Um sicherzustellen, dass die Architektur aus so vielen Blickwinkeln wie möglich betrachtet wird, sollte aus jeder Interessenvertretergruppe mindestens eine Person an der Evaluierung teilnehmen. Die SAAM besteht aus sechs Schritten, die nacheinander ausgeführt werden. Die Schritte eins und zwei werden in der Regel iterativ mehrmals hintereinander ausgeführt. Dieser Ablauf ist nur eine grobe Richtlinie und keine strikte Vorgabe. Auch in den Veröffentlichungen der SAAM Erfinder findet man immer einen leicht variierenden Ablauf der SAAM. Der hier beschriebene Ablauf stammt aus [ClKK05] und [BaCK98].

Schritt 1 und 2: Szenarioerhebung und Beschreibung der Architektur
Szenarien sollen möglichst alle Anforderungen abdecken, die an die Software gestellt sind und wahrscheinlich in Zukunft gestellt werden. Daher sollen im ersten Schritt alle Interessenvertreter möglichst frei ihre Anforderungen an die Software als Szenario formulieren können. Um dies zu gewährleisten bietet es sich an, die Szenarioerhebung als Brainstorming durchzuführen und die Szenarien ohne Kritik von den Interessenvertretern zu sammeln. Verständnisfragen zu den Szenarien sind aber erlaubt. Um die Szenarioerhebung nicht schon von Beginn an zu beeinflussen, kennen die Interessenvertreter in dieser ersten Iteration die Architektur des Systems noch nicht.
Im zweiten Schritt präsentiert der Architekt den Interessenvertretern in einer verständlichen Weise die Grundzüge der Architektur, so dass diese ein grobes Verständnis vom Aufbau und Ablauf der Architektur haben. Falls ein Szenario aus Schritt eins eine genauere Architekturbeschreibung erfordert, ist es Aufgabe des Architekten, die Architektur zu verfeinern und die verfeinerten Details zu präsentieren. Die Architekturkenntnisse soll die Interessenvertreter in der Folge zu weiteren Szenarien anregen, die in Schritt eins wieder erhoben werden.
Die Szenarioerhebung und die Architekturvermittlung hängen also eng zusammen und sollten daher iterativ ausgeführt werden. Wie lange die Ausführung wiederholt werden soll, darüber gibt es in der SAAM keine Vorgabe. Die pragmatische Antwort lautet: Solange wie nötig - Bis die Interessenvertreter keine neuen Szenarien mehr finden oder bis man genügend Szenarien gesammelt hat.

Schritt 3: Klassifikation und Gewichtung der Szenarien
In diesem Schritt werden die gesammelten Szenarien klassifiziert. Zuerst wird festgestellt, ob die Architektur das Szenario ohne Änderung unterstützt, also ob es direkt oder indirekt ist. Bei direkten Szenarien kann der Architekt gleich zeigen, wie es von der Architektur unterstützt wird. Diese Maßnahme kann das Verständnis der Architektur bei den Interessenvertretern noch weiter erhöhen und ermöglicht Raum für Diskussionen, z.B. falls Interessenvertreter spezielle Einwände haben, dass es sich doch nicht um ein direktes Szenario handeln könnte. Da die SAAM hauptsächlich die Modifizierbarkeit einer Architektur prüft werden die direkten Szenarien an dieser Stelle in aller Regel aussortiert. Die indirekten Szenarien versucht man zu gruppieren.

Ist ein Szenario ein Spezialfall eines anderen oder lassen sich zwei Szenarien auf einen allgemeineren Fall zurückführen werden sie zusammengefasst. Die resultierenden Szenarien, müssen priorisiert werden, da nur die wichtigsten von ihnen genauer untersucht werden sollten und aus Zeitgründen können. Zur Gewichtung schlägt die SAAM vor, jedem Interessenvertreter eine fixe Anzahl Stimmen zu geben, die dieser frei auf die Szenarien verteilen kann.

Schritt 4: Einzelne Bewertung der Szenarien

Jedes zu bewertende Szenario wird auf die Architektur abgebildet. Dabei muss festgestellt werden, welche Änderungen an der Architektur erforderlich sind. Welche Komponenten müssen neu erstellt oder geändert werden? Für jedes Szenario müssen außerdem die Aufwände für diese Änderungen vom Architekten geschätzt werden. Die SAAM empfiehlt die Ergebnisse der Szenariobewertung übersichtlich in einer Tabelle darzustellen. Dort sollten für jedes Szenario die benötigten Änderungen, die Anzahl der betroffenen Komponenten und der zeitliche Aufwand festgehalten werden.

Schritt 5: Untersuchung von Szenariointeraktionen

Um die Architektur weiter zu verbessern werden nun noch die Szenariointeraktionen untersucht. Von Interaktion spricht man, wenn zwei Szenarien Änderungen an der selben Komponente verursachen. Falls zwei oder mehrere voneinander unabhängige Szenarien interagieren, ist dies ein Hinweis, das die betroffene Komponente für verschiedene nicht zusammengehörige Funktionen verantwortlich ist. Die Komponente sollte in diesem Fall aufgeteilt werden. Falls ähnliche Szenarien nicht oder nur in geringem Maße interagieren ist auch Vorsicht geboten, denn dies deutet darauf hin, dass zusammengehörende Funktionalität über mehrere Komponenten verstreut ist. Das Gleiche gilt meist auch, wenn ein Szenario Änderungen in sehr vielen Komponenten erfordert. Die Untersuchung von Szenariointeraktionen unterstützt also bei der sinnvollen Aufteilung der Architektur und hilft direkt bei der Erreichung des Designziels: Hohe Kohäsion innerhalb einer Komponente und nur lose Kopplung zwischen Komponenten. Hierdurch profitiert die Modifizierbarkeit des Systems.

Schritt 6: Erstellen der Gesamtbewertung

Abschließend werden alle bewerteten Szenarien nochmals gewichtet. Dieses Mal aber eher nach wirtschaftlichen Gesichtspunkten. Welchen Nutzen bringt es mir, wenn meine Software ein Szenario umsetzt und welche Kosten und Aufwände stehen dem entgegen. Damit liegt den Softwarearchitekten eine nach Wichtigkeit geordnete Liste vor, welche Funktionen die Architektur bzw. später die Software unterstützen sollte und kann dementsprechend optimiert werden. Sollten mit der SAAM mehrere Architekturen verglichen werden, so können die Nutzen und Aufwände direkt verglichen und eine Entscheidung für die präferierte Architektur getroffen werden.

5.3 Bewertung

Wenn man die SAAM bewerten will, sollte man sich als ersten Anhaltspunkt anschauen, was man in die SAAM investieren muss, was man für die Anwendung der Methode aufwenden muss und was man dafür am Ende zurückbekommt. Als Voraussetzung für die SAAM benötigt man eine Softwarearchitektur. Sofern eine Beschreibung der Architektur nicht vorhanden ist, muss sie extra für die SAAM erstellt werden. Dies bedeutet einen größeren Aufwand. Allerdings bietet eine Architekturbeschreibung auch unabhängig von einer Evaluation einen Nutzen, wie in Kapitel zwei beschrieben. Die Kosten für die SAAM selbst sind dagegen relativ gering. Das Ergebnis der SAAM ist eine Liste mit den wichtigsten Anforderungen an die Software, die von der bestehenden Architektur nicht erfüllt werden. Mit anderen Worten eine Liste mit den Schwachstellen der Architektur bezüglich der wahrscheinlichsten Änderungsanforderungen an die Software. Lässt man die Ergebnisse der SAAM in die Architektur miteinfließen erhält man also eine qualitativ hochwertigere Software, die sich wesentlich leichter an zukünftige Anforderungen anpassen lässt. Das heißt nicht zwangsläufig, dass die Software generell besser modifizierbar wird, sondern nur an den Stellen, die mit hoher Wahrscheinlichkeit auch geändert werden sollen. Wenn man sich an dieser Stelle nochmals vor Augen führt, dass bei einer Software der weitaus größte Teil der Kosten nicht durch die Erstellung, sondern durch Wartung und Modifikation entsteht [BaCK98, S.32], ist dies ein gewaltiger Kostenvorteil. Die Ergebnisse der SAAM hängt zu 100% von den Szenarien ab. Diese wiederrum hängen von den beteiligten Interessenvertretern ab. Dies ist auch die größte Gefahr. Wenn die Interessenvertreter keine guten Szenarien erheben, liefert auch die SAAM keine guten Ergebnisse. Allerdings sind in der Literatur keine Fälle bekannt, dass dieser Fall schon einmal eingetreten wäre. Denn wer sollte die Anforderungen an eine Software besser kennen, als die Personen, die an ihr teilhaben. Selbst unabhängig von den Ergebnissen der SAAM liefert schon der Prozess alleine Vorteile. Wenn der Architekt in Schritt zwei die Architektur vorstellt und beschreibt, wird der Auftraggeber der Software, implizit überprüfen, ob der Architekt alle seine Anforderungen und Wünsche, die er in der Anforderungsanalyse gestellt hat, in der Architektur realisiert hat. Des Weiteren verbessert die Kenntnis der Architektur bei allen Projektbeteiligten das Verständnis für die Software und ihre Arbeitsweise. Durch die Konfrontation mit den anderen Interessenvertretern bekommen alle Beteiligten ein umfassenderes Bild.

Die SAAM ist einfach durchzuführen, ohne viele Vorkenntnisse und Aufwand. Daher eignet sich die Methode auch für Neulinge im Feld der Softwarearchitekturevaluation. Die SAAM gibt eine Ablaufstruktur vor, an der Neulinge sich entlang hangeln können, es wird aber in der Literatur [BaCK98] darauf hingewiesen, dass man die Struktur den Gegebenheiten eines speziellen Falls anpassen kann. So erhalten erfahrene Anwender der SAAM die nötige Flexibilität die SAAM in verschiedensten Anwendungsfällen einzusetzen.

Die SAAM ist seit 1994 publiziert und hat sich in der Praxis schon in den verschiedensten Anwendungsfällen bewährt. Das heißt sie hat sich zu einem wichtigen Werkzeug entwickelt, wenn man Architekturen auf Funktionalität und die

verschiedenen Formen von Modifizierbarkeit prüfen will und hilft damit Kosten zu sparen.

5.4 Weiterentwicklungen

Inzwischen gibt es viele Weiterentwicklungen der SAAM, die sich jeweils auf spezielle Aspekte der Analyse fokussieren. Bei [Eick07] und [BaZJ04] findet man folgende SAAM Derivate:

- ASAAM - Aspectual SAAM
- SAAMCS - SAAM Founded on Complex Scenarios
- ESAAMI - Extending SAAM by Integrating in the Domain
- SAAMER - SAAM for Evolution and Reusability
- FAAM - Family Architecture Assessment Method
- ALPSM - Architecture Level Prediction of Software Maintenance
- PASA - Performance Assessment of Software Architecture
- ALMA - Architecture-Level Modifiability Analysis

Drei der bekanntesten Abwandlungen und Verfeinerungen werden im Folgenden kurz vorgestellt.

ASAAM - Aspectual SAAM

Die ASAAM erweitert das Konzept der SAAM so, dass die Architektur Aspektorientierung unterstützen kann. Aspekte oder auch „Cross-cutting concerns“ sind Funktionalitäten, die in mehreren Komponenten einer Software vorkommen oder gebraucht werden, die sich aber nicht nur zentral an einer Stelle realisieren lassen. Ein Beispiel für einen Aspekt ist z.B. Logging. Semantisch betrachtet ist Logging eine zusammengehörende Funktion, die es einer Software erlaubt, Variablen oder Zustände zu protokollieren. Im Programmcode ist Logging aber an vielen Stellen verteilt. Jede Komponente die Logging unterstützen will, muss mindestens eine Schnittstelle für Logging implementieren. Das ist unschön, da dies die Wiederverwendung dieser Komponenten, z.B. in einer anderen Software ohne Logging, sehr erschwert. Aspektorientierung strebt an, verschiedene Aspekte in einer Software getrennt voneinander zu entwickeln und sie erst zur Übersetzungszeit oder sogar erst zur Laufzeit zusammenzufügen. So wird es ermöglicht Änderungen an einem Aspekt nur an einer Stelle vornehmen zu müssen, anstatt den Code an allen verwendeten Stellen suchen und ändern zu müssen. Zudem erlaubt Aspektorientierung verschiedene Aspekte unabhängig voneinander wiederzuverwenden. Allerdings erkauft man sich dies mit einer komplizierteren Softwareentwicklung. Da der Code zusammengebaut wird ist es ungleich schwerer die sequentielle Abfolge des Befehle nachzuvollziehen. [Mill01]

Die ASAAM will Aspekte schon auf Architekturebene unterstützen und modifiziert dazu den Ablauf der SAAM etwas. Die ersten beiden Schritte, Szenarioerhebung und die Beschreibung der Architektur, sind gleich wie bei der SAAM. Im dritten Schritt, wenn es gilt die Szenarien zu klassifizieren, werden

Szenarien auf Aspekte hin untersucht. Für jedes Szenario wird geprüft, ob es mehrere Komponenten betrifft. Trifft dies zu und das Szenario lässt sich nicht auf eine betroffene Komponente reduzieren, wird das Szenario als „Aspekt Szenario“ klassifiziert und daraus dann ein Aspekt abgeleitet. Die ASAAM enthält zusätzlich noch ein Regelwerk um für jeden Aspekt die Komponenten zu ermitteln, in denen der Aspekt verwendet wird. Sind die Aspekte und die jeweils betroffenen Komponenten bekannt, kann die Architektur entsprechend angepasst werden. Der weitere Ablauf der Methode unterscheidet sich nicht von der SAAM. [Teki04]

SAAMCS - SAAM Founded on Complex Scenarios

Die SAAMCS versucht eine Risikobewertung darüber zu erstellen, wie flexibel eine Architektur ist. Aus diesem Grund wird bei der Szenarioerhebung versucht möglichst komplexe Szenarien zu finden, also Szenarien die sich nur aufwendig realisieren lassen. Die Beschreibung der Softwarearchitektur wird erweitert, so dass sie nicht nur die Anwendung selbst, sondern auch ihre Umgebung und die Interaktion damit darstellt und danach in eine Makro- und Mikroarchitektur aufgeteilt. Die SAAMCS definiert speziell für diese erweiterte Architektur und die komplexen Szenarien ein Regelwerk, um die Szenarien zu klassifizieren und festzustellen, wie sie sich auf die Architektur und die Umgebung auswirken. [Dobr02]

ESAAMI - Extending SAAM by Integrating in the Domain

Der Fokus von ESAAMI liegt auf der Wiederverwendung von Design. Wie der Name andeutet beschränkt man SAAM auf ein spezielles Anwendungsgebiet. Der Ablauf bei ESAAMI ist ähnlich wie bei der SAAM, aber man startet bei ESAAMI nicht mit einer speziellen Softwarearchitektur, sondern man hat für das gewählte Anwendungsgebiet eine Standardarchitektur und einen Menge Schablonen, mit denen man die Architektur erweitern und an die speziellen Bedürfnisse der Interessenvertreter und der Software anpassen kann. Dieses Konzept kann nur innerhalb eines Anwendungsgebietes funktionieren, in der jede Software immer bestimmte gleiche Merkmale aufweist. [Dobr02]

6 SAAM in der Praxis - WRCS Fallbeispiel

Die SAAM wurde seit 1994 in etlichen Fällen sowohl in der Industrie als auch im akademischen Umfeld praktisch eingesetzt [BaCK98]. Im Folgenden wird ein Fall dargestellt, der eine eher untypische Anwendung der SAAM ist, zeigt damit aber die flexible Einsetzbarkeit von SAAM. Der Fall beruht auf [SiKa95] und [Kazm96].

6.1 Problemstellung

Ziel dieses Falles ist es, die Architektur des Versionsverwaltungssystems WRCS zu untersuchen und mögliche Schwachstellen festzustellen, die eine einfache Anpassung der Software an neue Anforderungen behindern. Das Besondere an diesem Fall ist, dass die SAAM auf eine schon fertige Software angewendet werden soll. Da es für WRCS bislang noch keine Architekturbeschreibung gibt, muss sie erst erstellt werden. Erschwerend kommt hinzu, dass nur die Binaries der Anwendung vorliegen, aber kein Quellcode. Auf Probleme dieser Art stößt man in der Praxis leider häufig. Man hat also keine andere Wahl als die Anwendung zu untersuchen und zu versuchen ihre Architektur per Reverse Engineering mühsam abzuleiten. Als Hilfe konnte man in diesem Fall die Softwaredokumentation verwenden und hatte die Möglichkeit einige der ursprünglichen Entwickler der Software zu befragen.

6.2 WRCS

Mit WRCS kann man Versionsverwaltung für Projekte betreiben. Ein Projekt besteht aus einer Reihe zusammengehörender Dateien. Diese Dateien können von den Benutzern ausgecheckt, unabhängig voneinander bearbeitet und wieder eingecheckt werden. WRCS speichert und protokolliert die Änderungen an den einzelnen Dateien sowie die Versionen eines Projektes. Zudem bietet das System erweiterte Verwaltungsfunktionen, wie z.B. das Erstellen von Berichten. WRCS besitzt eine eigene grafische Benutzeroberfläche und eine Schnittstelle für die Einbindung der Funktionalität in externe Programme. So kann man beispielsweise direkt aus einer Entwicklungsumgebung heraus auf WRCS zugreifen, um seinen Quellcode zu verwalten.

6.3 Erstellung der Softwarearchitektur

Ausgangspunkt für die Erstellung der Architektur bildete eine grobe Aufteilung der Software in Module. In einem iterativen Verfahren versuchte man dies zu verfeinern und ein komplette Architektur daraus zu erstellen. In jeder Iteration stellte man an die Architektur bestimmte Fragen, die man dann anhand aller vorhandener Information zu beantworten versuchte. Mit den gefundenen Antworten verfeinerte man das Modell und startete den Prozess von Neuem, solange bis die Architektur detailliert genug vorlag. Abbildung 1 zeigt eine vereinfachte Version der gefundenen Architektur.

6.4 Szenarioerhebung

Die Szenarioerhebung führte zu 15 Szenarien, darunter Szenarien aus Benutzer-, Entwickler- und Administratorsicht. Die sechs wichtigsten Szenarien werden hier vorgestellt.

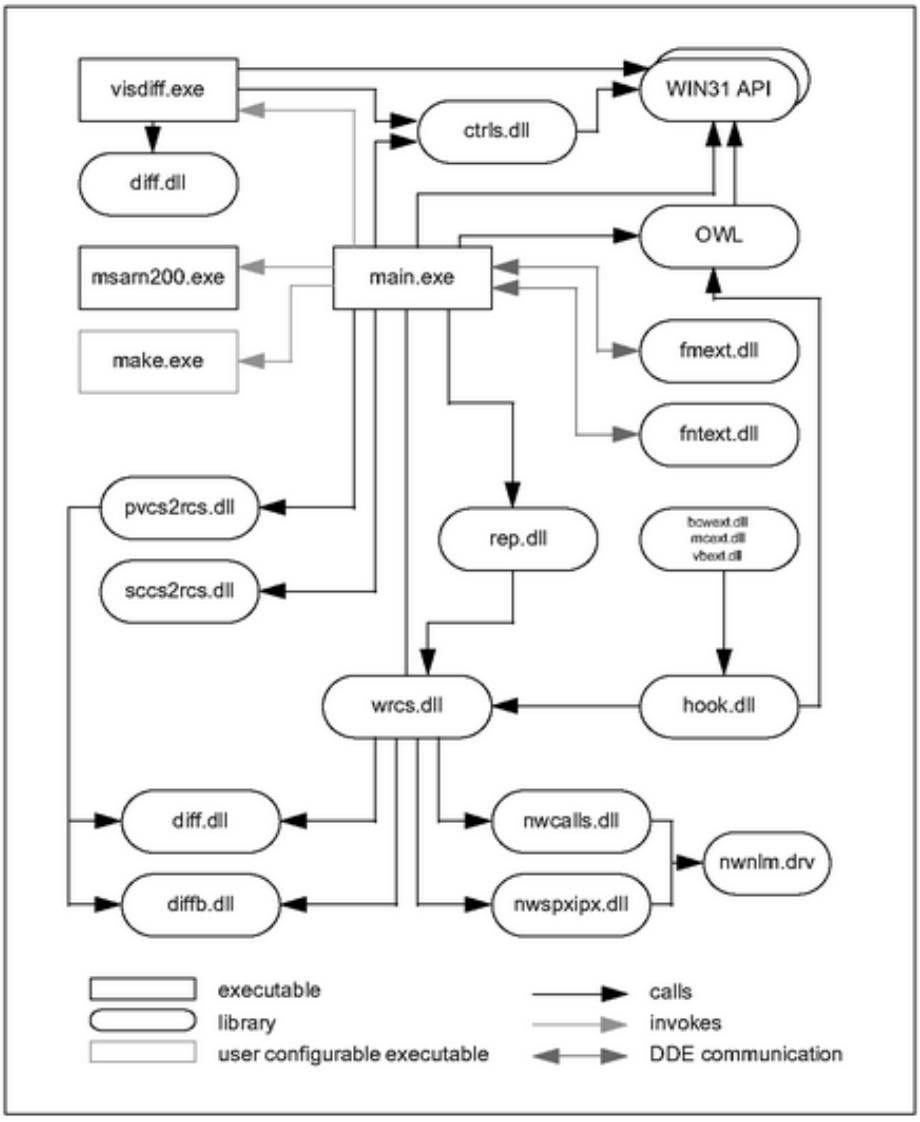


Abbildung 1. Vereinfachte Architektur von WRCS (Quelle: [SiKa95])

1. **Unterstützung verschiedener binärer Dateiformate**
   Soll beispielsweise die Änderung an einem Word Dokument dargestellt werden, soll nicht dargestellt werden, welche Bits sich in der Datei geändert haben, sondern für den Benutzer verständlich, wie diese Änderung in Word aussieht.

2. **Anpassen der Toolbar des Programms**
   Die Bilder und Befehle der Buttons auf der Toolbar, sollen angepasst werden können.
3. **Portierung auf ein anderes Betriebssystem**
4. **Kleine Änderungen an der Benutzerschnittstelle**
   Anpassungen, wie das Hinzufügen von Menüpunkten oder das Ändern der Darstellung von Dialogboxen, sollen möglich sein.
5. **Zugriffsrechte auf ein Projekt ändern**
6. **Einbinden der WRCS Funktionen in weitere Entwicklungsumgebungen**

6.5 Klassifikation und Bewertung der Szenarien

Im nächsten Schritt wurden die Szenarien in direkte und indirekte unterteilt. Bei den indirekten Szenarien wurde versucht aufzuzeigen welche Komponenten geändert werden müssen, um sie zu unterstützen. Dabei wurde gleich mit abgeschätzt, welchen Aufwand diese Änderungen verursachen würden. Folgende Übersicht liefert die Ergebnisse dieses Schrittes:

- **Unterstützung verschiedener binärer Dateiformate**
  Klasse: Indirekt
  Erforderliche Änderungen: *diff* ändern für das Vergleichen, *visdiff* anpassen für die Darstellung

- **Toolbar anpassen**
  Klasse: Direkt
  Erforderliche Änderungen: -

- **Portierung auf ein anderes Betriebssystem**
  Klasse: Indirekt
  Erforderliche Änderungen: Alle Komponenten, die die *win31* API aufrufen müssen so angepasst werden, dass sie die API des neuen Betriebssystems unterstützen. Der Einsprungpunkt in die *main.exe* muss geändert werden. Die Module für die Einbindung von WRCS in externe Anwendungen müssen umgeschrieben werden, ebenso wie die Interprozesskommunikation mit *hook.dll*. Sollte das neue Betriebssystem das dll Konzept nicht kennen, sind hier weitere Anpassungen nötig. Falls das neue Betriebssystem *owl* nicht unterstützt muss diese Komponente zusätzlich geändert werden.

– **Kleine Änderungen an der Benutzerschnittstelle**
Klasse: Indirekt
Erforderliche Änderungen: Die Aufrufe der *win31* API in *main.exe*, *visdiff*, *hook.dll* und eventuell *ctrls.dll* müssen angepasst werden.

– **Zugriffsrechte auf ein Projekt ändern**
Klasse: Direkt
Erforderliche Änderungen: -

– **Einbinden der WRCS Funktionen in weitere Entwicklungsumgebungen**
Klasse: Indirekt
Erforderliche Änderungen: Pro Entwicklungsumgebung muss eine neue kleine spezielle Komponente geschrieben werden. Dies ist aber nur geringer Aufwand, da das generelle Anbinden externer Anwendungen in *hook.dll* gekapselt ist.

6.6 Szenariointeraktion

Von Szenariointeraktion spricht man, wenn eine Komponente aufgrund von mehr als einem unabhängigen Szenario geändert werden muss. Welche Szenarien welche Komponente betreffen kann man bequem aus dem vorherigen Schritt ablesen und ist in Tabelle 1 zusammengefasst. Allerdings sind hier alle 15 Szenarien des Fallbeispiels berücksichtigt und nicht nur die sechs hier näher betrachteten Szenarien.

Tabelle 1. Szenariointeraktion der Module bei WRCS

Modul	**Anzahl Szenarien**
wrcs.dll	7
hook.dll	4
main.exe	4
visdiff.dll	3
ctrls.dll	2
diff.dll	1
diffb.dll	1
nwcalls.dll	1
nwnlm.dll	1
nwspxipx.dll	1
pvcs2rcs.dll	1
rep.dll	1
sccs2rcs.dll	1

Aus Tabelle 1 ist klar ersichtlich, dass die Anwendung nicht sinnvoll modularisiert ist. *wrcs.dll*, *hook.dll*, *main.exe* und *visdiff.dll* enthalten zuviel Funktio-

nalität, die semantisch nicht zusammengehört. Dies sind also die Schwachpunkte an denen man ansetzen muss.

6.7 Ergebnisse

Rückblickend konnten unter anderem folgende konkrete Schwachstellen identifiziert werden:

- Eingeschränkte Portabilität der Benutzerschnittstelle, da die Funktionalität nicht an einer Stelle gekapselt ist, sondern über viele Komponenten verteilt ist.
- Eingeschränkte Modifizierbarkeit der Benutzerschnittstelle, da sie sehr eng mit der win31 API verknüpft ist. Hier wäre eine Zwischenschicht nötig, die beides entkoppelt.

Man kann aber auch eine Stärke der Architektur feststellen:

- Einfache Erweiterbarkeit. Danke der guten Schnittstelle kann die Integration in externe Software sehr einfach erfolgen.

Das Hauptaugenmerk des Kunden bei dieser Untersuchung lag in der Portierbarkeit seiner Anwendung. Da dies bei der Benutzerschnittstelle schwerlich möglich ist, ist die Empfehlung des Evaluierungsteams, das System einer großen Überarbeitung zu unterziehen. Ansatzpunkte hierzu liefert die Szenariointeraktion. Die dort genannten Module sollten in kleinere Module aufgespalten werden. Zusammengehörige Funktionalität sollte in zentralen Komponenten zusammengezogen werden, dies gilt vor allem für die grafische Benutzeroberfläche. Die Überarbeitung würde die Qualität der Software deutlich erhöhen. Das Redesign ist ein relativ aufwändiger Prozess.

7 Fazit

Die Softwarearchitektur eines Anwendungssystems beeinflusst entscheidend die Qualitätsmerkmale eines Anwendungssystems. Die Bewertung der gewählten Architektur bzw. der getroffenen Entwurfsentscheidungen erlaubt deshalb bereits vor der eigentlichen Implementierung, eine Einschätzung der grundlegenden Qualitätsmerkmale des zu entwickelnden Systems [Eick07]. Durch die Verwendung von Szenarien und die Einbeziehung der Interessenvertreter wird sichergestellt, dass die Anwendung aus den Blickwinkeln untersucht wird, die am wichtigsten für die Software sind bzw. sein werden. Dadurch kann die Qualität der Software gesteigert werden und die Kosten für später Korrekturen oder Anpassungen gesenkt werden. Durch die frühe Anwendbarkeit kann man mit der SAAM Fehler oder Probleme vermeiden, bevor sie entstehen und Kosten verursachen. Die Fallstudie im letzten Kapitel ist ein deutlicher Beleg dafür. Die Hürden um die SAAM anzuwenden sind dagegen sehr niedrig. Verglichen mit ihrem potentiellen Nutzen ist sie zu sehr niedrigen Kosten durchführbar. Sie ist

einfach anzuwenden, flexibel und hat sich auch in der Praxis bewährt. Schon alleine durch die implizite Forderung nach einer guten Architekturbeschreibung und einem besseren Verständnis derselben bei den Interessenvertretern erzielt man einen Nutzen durch die SAAM. Wenn man die „Total Cost of Ownership“ einer Software, also den gesamten Kosten, die während des Lebenszykluses der Software anfallen, senken will, sollte die SAAM fester Bestandteil im Softwareentwicklungsprozess werden.

Literatur

BaCK98. Len Bass, Paul Clements und Rick Kazman. *Software architecture in practice*. Addison-Wesley. 1998.

BaZJ04. Muhammad Ali Babar, Liming Zhu und Ross Jeffery. A Framework for Classifying and Comparing Software Architecture Evaluation Methods. *aswec*, Band 00, 2004, S. 309.

ClKK05. Paul Clements, Rick Kazman und Mark Klein. *Evaluating software architectures*. Addison-Wesley. 4. print.. Auflage, 2005.

Dobr02. E. Dobrica, L.; Niemela. A survey on software architecture analysis methods. *Transactions on Software Engineering*, 28(7), Jul 2002, S. 638–653.

Eick07. Christian; Malich Stefan Eicker, Stefan; Hegmanns. Auswahl von Bewertungsmethoden für Softwarearchitekturen. *ICB-Research Report*, Band 14, Mar 2007.

Kazm96. G.; Bass L.; Clements P. Kazman, R.; Abowd. Scenario-based analysis of software architecture. *Software, IEEE*, 13(6), Nov 1996, S. 47–55.

Lang06. M.R.V.; Muskens J. Lange, C.F.J.; Chaudron. In practice: UML software architecture and design description. *Software, IEEE*, 23(2), March-April 2006, S. 40–46.

Mill01. S.K. Miller. Aspect-oriented programming takes aim at software complexity. *Computer*, 34(4), Apr 2001, S. 18–21.

SiKa95. Mauricio De Simone und Rick Kazman. Software architectural analysis: an experience report. In *CASCON '95: Proceedings of the 1995 conference of the Centre for Advanced Studies on Collaborative research*. IBM Press, 1995, S. 18.

Teki04. B. Tekinerdogan. ASAAM: aspectual software architecture analysis method. *Software Architecture, 2004. WICSA 2004. Proceedings. Fourth Working IEEE/IFIP Conference on*, 12-15 June 2004, S. 5–14.

Cost Benefit Analysis Method (CBAM)

Fabian Brosig

Betreuer: Thomas Goldschmidt

Zusammenfassung Der Nutzen eines Software-Systems wird durch die gebotene Funktionalität in Verbindung mit der Ausprägung der Qualitätsmerkmale (Leistung, Wartbarkeit, Sicherheit etc.) diktiert. Die Qualitätsmerkmale werden im Wesentlichen von der Software-Architektur bestimmt. Die Entscheidungen für oder gegen die Umsetzung von Architekturvorschlägen erfordern also eine sorgfältige Analyse der zu erwartenden Auswirkungen. Die in dieser Arbeit vorgestellte Cost Benefit Analysis Method (CBAM) beschreibt einen strukturierten Entscheidungsprozess auf Architekturebene, der auf betriebswirtschaftlichen Überlegungen basiert: Bewertet wird das Verhältnis zwischen Nutzen und Kosten eines Vorschlags, unter Einbeziehung der Risiken. Nach der Vorstellung dieser Methode anhand einer Fallstudie folgt eine Diskussion über die Erfahrungen und Schwierigkeiten, die sich bei der Anwendung der CBAM ergeben.

1 Einführung

Bei der Entwicklung und Wartung eines Software-Systems hat die zugrundeliegende Software-Architektur eine wichtige Rolle. Auf einer abstrakten Ebene definiert sie das Zusammenspiel der einzelnen Software-Komponenten. Entscheidungen für oder gegen die Realisierung von verschiedenen Architekturvorschlägen haben somit eine große Tragweite für das gesamte Produkt.

Um die Entscheidungsfindung im Rahmen einer ingenieurmäßigen Software-Entwicklung zu operationalisieren, hat das Software Engineering Institute (SEI) an der Carnegie Mellon University verschiedene Methoden entwickelt. Die Architecture Tradeoff Analysis Method (ATAM) und die Cost Benefit Analysis Method (CBAM) haben das Ziel, einen rationalen, strukturierten und wiederholbaren Entscheidungsprozess zu etablieren. Die ATAM und die CBAM sind keine konträren Methoden, die CBAM baut auf der ATAM auf. Während die ATAM die Frage nach der "richtigen" Architektur aus einer rein technischen Perspektive zu beantworten versucht, ist die CBAM durch ökonomische Überlegungen motiviert. Die CBAM betrachtet die Auswahl von Architekturen im Hinblick auf Gewinnmaximierung. Eine "richtige" Architektur im Sinne der CBAM ist somit diejenige Architektur, die die beste Investition darstellt. Zur Bewertung einer Architektur werden also eine Kosten- und Nutzen-, sowie eine Risikoanalyse erstellt.

Laut [KAK01, S.297] sind insbesondere bei großen, komplexen Software-Systemen nicht nur technische Überlegungen beim Abwägen zwischen mehreren Architekturvorschlägen von Bedeutung. Die wichtigste Fragestellung sei, wie

man eine bestimmte Menge von Ressourcen möglichst gewinnbringend und möglichst risikoarm einsetzt. Diese Fragestellung möchte die CBAM auf der Ebene der Software-Architektur beantworten. Bisherige Forschungen der Software-Technik konzentrieren sich meist auf den Kostenaspekt, wobei jedoch oft nur die Erstellungskosten betrachtet werden [KAK01, S.297]. Langfristige Kosten, beispielsweise verursacht durch Architekturmaßnahmen, die einen aufwändigeren Betrieb oder eine aufwändigere Wartung der Software zur Folge haben, bleiben oft unberücksichtigt. Analysen zum Gewinnpotenzial der Maßnahmen finden ebenfalls nicht statt.

Die CBAM darf nicht als "Black Box" angesehen werden, die eine fixe, optimale Entscheidung liefert. Die Methode bietet ein Rahmenwerk, das den Prozess zum Treffen von Architekturentscheidungen strukturiert [KAK02, S.21]. Das Team aus Interessenvertretern, etwa bestehend aus Projektmanager, Architekt und Benutzern/Kunden erhält eine Anleitung, wie zum Zwecke der Entscheidungsfindung vorgegangen werden soll. Es wird angenommen, dass diese Interessenvertreter Expertenwissen hinsichtlich des zu analysierenden Systems haben.

Nach einer generellen Einführung in das Thema "Treffen von Architekturentscheidungen" wird die ATAM kurz beschrieben. Danach wird in Teil 2 dieser Arbeit die CBAM anhand einer Fallstudie dargestellt. Eine Diskussion über die CBAM, die Darstellung der Erfahrungen, die aus der Fallstudie gewonnen werden konnten und weitere Einschätzungen bilden den Inhalt von Teil 3. Abgeschlossen wird die Arbeit mit einer Zusammenfassung und einem Ausblick.

1.1 Treffen von Architekturentscheidungen

"Die Software-Architektur ist ein grundlegender Bestandteil eines komplexen Software-Systems" [AKK01, S.1]. Mit steigender Komplexität bekommt die Architektur eine bedeutendere Rolle als die Auswahl von speziellen Algorithmen und Datenstrukturen [SG96]. Mit anderen Worten: Je größer das Software-System, desto wichtiger ist die Struktur. "Traditionelle Software-Entwurfsmethoden behandeln die technischen Aspekte von Software-Systemen, indem die zu liefernde Funktionalität in der Anforderungsanalyse und Spezifikationsphase der Produktentwicklung identifiziert wird" [AKK01, S.25]. Im Folgenden wird angenommen, dass die Anforderungen und die daraus abgeleitete Funktionalität feststehen. Der Wert einer Software wird jedoch nicht allein durch ihre Funktionalität bestimmt, sondern auch durch die Umsetzung von nicht funktionalen Anforderungen. Übliche Qualitätsmerkmale wie Leistung, Wartbarkeit, Sicherheit, Zuverlässigkeit oder Benutzbarkeit werden laut den Autoren der CBAM von "architektonischen Entwurfsentscheidungen diktiert" [KAK01, S.297]. Diese Einschätzung begründet die Meinung, dass Entwurfsentscheidungen, die die Software-Architektur betreffen, sorgfältig abgewogen werden müssen.

Diese Entscheidungen werden von einem Team bestehend aus Interessenvertretern getroffen. "Unstrukturierte Diskussionen, bei denen Anforderungen, Architekturstrategien und eigene Meinungen vermischt werden" [MKKA03, S.6], sollen durch definierte Entscheidungsprozesse vermieden werden.

Das SEI formulierte in den 90er Jahren Methoden, die gegebene Architekturen analysieren. Die Software Architecture Analysis Method (SAAM) dient der Bewertung von Architekturen in Bezug auf die Erfüllung der (üblichen) Qualitätsmerkmale. Die ATAM ist ein Nachfolger der SAAM. Sie hat das Ziel, Konsequenzen von Architekturentscheidungen im Hinblick auf geforderte Qualitätsmerkmale einzuschätzen [KKC00, S.2]. Die im Jahr 2001 vorgestellte CBAM bietet ein Rahmenwerk, um Architekturvorschläge bewerten und auswählen zu können. Die Vorschläge, im Folgenden auch Strategien genannt, werden hierbei ebenfalls auf ihre Auswirkungen auf die Qualitätsmerkmale des Gesamtprodukts untersucht. Die CBAM basiert auf der ATAM, zieht zur Bewertung der Architekturstrategien aber betriebswirtschaftliche Überlegungen wie Kosten, Nutzen, und Risiko hinzu. In der Praxis sollen die genannten Methoden strukturierte, klare Entscheidungsprozesse formen.

1.2 Architecture Tradeoff Analysis Method (ATAM)

Da die CBAM auf der ATAM, beziehungsweise auf deren Ergebnissen aufbaut, folgt an dieser Stelle eine Übersicht über die ATAM. Bei der ATAM geht es darum, herauszufinden, inwieweit eine Architektur den Anforderungen an Leistung, Wartbarkeit, Bedienbarkeit etc. gerecht wird. Am Ende der Methode steht eine die Analyseergebnisse beschreibende und erklärende Architekturdokumentation.

Die Methode hat den Anspruch, wiederholbar und bereits im frühen Entwicklungsstadium der Software anwendbar zu sein, um eventuell auftretende Probleme früh zu erkennen. Die ATAM hat nicht den Anspruch, den Grad der Erfüllung einzelner Qualitätsmerkmale vorherzusagen [KKC00, S.3]. In einer frühen Entwicklungsphase wäre so eine Vorhersage ohnehin nicht möglich, da noch nicht genügend Information existieren würde.

Risiken, sensitive Punkte und Konfliktpunkte Mit der Anwendung der Methode soll bezweckt werden, kritische Stellen der Architektur zu erkennen. Stellen, die einige Qualitätsmerkmale wie zum Beispiel Leistung besonders beeinflussen können. Hat man solche “neuralgischen Punkte” erkannt, kann man nachfolgende Analysen auf diese fokussieren. Die Stellen werden in drei Kategorien, die wie folgt charakterisiert werden, eingeteilt [KKC00, S.3]:

- Risiken: Architekturentscheidungen, die noch nicht getroffen oder deren Auswirkungen noch nicht vollständig verstanden wurden.
- sensitive Punkte: Architekturparameter, die ein Qualitätsmerkmal stark beeinflussen.
- Konfliktpunkte: Architekturparameter, die sensitive Punkte für mehrere Qualitätsmerkmale sind und diese teils negativ, teils positiv beeinflussen.

Zur Verdeutlichung folgen einige Beispiele: Angenommen, eine Architektur legt nicht fest, ob eine relationale oder objektorientierte Datenbank zur Persistierung von Daten verwendet werden soll, so entspricht das nach obiger Unterteilung einem Risiko. Ein Kommunikationskanal, der einen Flaschenhals darstellt, ist ein

sensitiver Punkt. Es wäre außerdem ein Konfliktpunkt, falls eine den Durchsatz beschleunigende Maßnahme die Zuverlässigkeit sinken lässt [KKC00, S.3].

Neben Risiken sind auch Nicht-Risiken von Interesse. Letztere sind Architekturentscheidungen, die als geeignet eingeordnet werden, also keine Probleme verursachen. Die Auswirkungen so einer Architekturentscheidung müssen demnach genau bekannt und gutartig sein.

Die Einstufung eines Architekturbestandteils als sensitiven Punkt dient als Kennzeichnung, dass Änderungen an diesem Bestandteil entscheidende Auswirkungen haben können und deswegen besonders sorgfältig überdacht werden müssen. Die kritischsten Architekturentscheidungen betreffen Konfliktpunkte, da hier zwangsläufig Kompromisse in Bezug auf die Erfüllung von Qualitätskriterien gemacht werden müssen.

Charakterisierung von Qualitätsmerkmalen Die ATAM fokussiert sich auf die Erfüllung der Anforderungen an die Qualitätsmerkmale. Dazu müssen die Anforderungen jedoch erst verstanden werden. Formulierungen wie "Das System soll eine hohe Leistung haben." sind offenbar zu ungenau und abstrakt, um Entscheidungen daran auszurichten. Die Anforderungen können erst dann genauer beschrieben werden, wenn die Geschäftsziele des Software-Produkts verstanden sind.

Um Qualitätsanforderungen zu konkretisieren, werden von den Interessenvertretern Szenarien formuliert. Damit gilt die ATAM als szenarienbasiertes Architekturbewertungsverfahren. Die Szenarien beschreiben die Anforderungen auf eindeutige Art in Form von Reiz und Reizantwort [KKC00, S.5]. Mit anderen Worten: Der Erfüllungsgrad der Anforderungen an die Qualitätsmerkmale wird erst durch die Szenarien messbar und überprüfbar. Ein anschauliches Szenario in Bezug auf die Leistung beispielsweise eines Bibliotheksystems wäre: "Die Suche nach Stichwörtern muss innerhalb von einer Sekunde die Suchergebnisse bereitstellen."

Sämtliche Szenarien werden in Form von sogenannten Qualitätsattributbäumen dargestellt. Ausgehend von den Qualitätsmerkmalen werden die Szenarien in Unterfaktoren gruppiert. Unterfaktoren für die Leistungsfähigkeit eines Systems können zum Beispiel Verzögerungszeit und Durchsatz sein. Abbildung 1 zeigt einen Ausschnitt eines beispielhaften Qualitätsattributbaums [KKC00, S.17]. Die Blätter des Baumes entsprechen den Szenarien. Diese sind auf der Ordinalskala (Hoch, Mittel, Niedrig) nach zwei Kriterien gewichtet. Die Priorität entspricht dem ersten, der geschätzte Aufwand dem zweiten Kriterium.

Schritte der ATAM Die ATAM ist eine strukturierte Methode und besteht aus mehreren Schritten. Am Anfang der Methode steht eine informelle Vorbereitungsphase, in der festgelegt wird, wer die Evaluation leitet, welche Personen als teilnehmende Interessenvertreter in Frage kommen und wie die Bewertung organisiert wird. Anschließend werden folgende Schritte durchgeführt [KKC00, S.7f]:

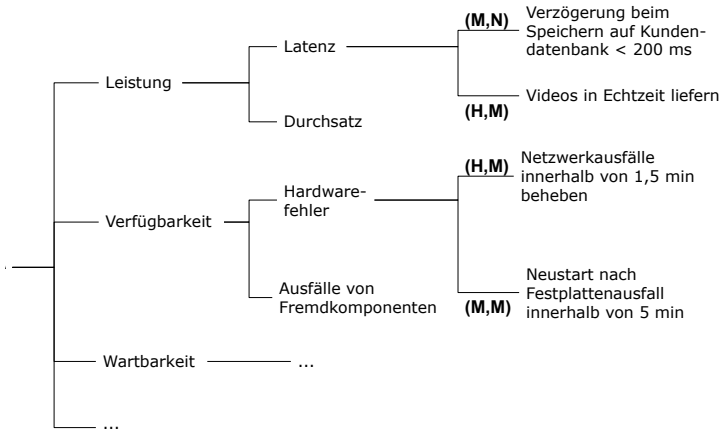


Abbildung 1. Exemplarischer Qualitätsattributbaum [KKC00, S.17]

1. Die ATAM vorstellen: Die Methode wird den an der Bewertung teilnehmenden Interessenvertretern vorgestellt. Das Team besteht meist aus Kundenvertretern, Architekt(en), Benutzern, Administratoren, Managern, Testern, etc.
2. Die Geschäftsziele vorstellen: Der Projektmanager beschreibt die Geschäftsziele und motiviert die Hauptanforderungen an die Architektur und das gesamte Software-Produkt.
3. Die Architektur vorstellen: Der Architekt beschreibt die vorgeschlagene (oder bereits existierende) Architektur und legt insbesondere die Ausrichtung an den Anforderungen dar, die im vorherigen Schritt genannt wurden.
4. Die Architekturansätze identifizieren: Die einzelnen Architekturansätze werden vom Architekten identifiziert, nicht analysiert. Unter einem Architekturansatz wird an dieser Stelle ein Rahmen für detailliertere Architekturentscheidungen verstanden, der auf die Erfüllung bestimmter Qualitätsmerkmale abzielt. Beispiel: Das Dienstnehmer/Dienstgeber-Prinzip ist ein Architekturansatz und gilt als skalierbar [KKC00, S.5].
5. Generieren des Qualitätsattributbaums: Zur Beschreibung der abstrakten Qualitätsmerkmale werden Faktoren und Szenarien gesucht, priorisiert und mittels eines Qualitätsattributbaums zusammengefasst.
6. Die Architekturansätze analysieren: In diesem Schritt werden die Risiken, die Nicht-Risiken, die sensitiven Punkte und Konfliktpunkte identifiziert. Das geschieht auf Grundlage der Ergebnisse der vorherigen Schritte unter Berücksichtigung der Prioritäten der Szenarien.
7. Weitere Szenarien finden und priorisieren: Unter Beteiligung aller werden weitere Szenarien erstellt und priorisiert. Während in den vorherigen Schritten primär aus der Perspektive des Architekten agiert wird, ist es in diesem Schritt wichtig, dass alle Interessenvertreter ihre Sichtweisen darlegen [BCK03]. Das soll durch einen Brainstorming Prozess unterstützt werden.

8. Die Architekturansätze analysieren: Dieser Schritt ist eine Wiederholung von Schritt 6. Allerdings werden hier die hoch priorisierten Szenarien des vorherigen Schrittes als Tests auf die Architektur angewendet. Diese Tests können weitere Schwachstellen der Architektur finden, die dem Architekten allein nicht auffallen würden.
9. Ergebnis präsentieren: Aus den gesammelten Informationen wird ein Abschlussbericht erstellt, der die Ergebnisse des Verfahrens detailliert darstellt.

Resultate der Anwendung der ATAM Die Ergebnisse einer ATAM bestehen aus der Menge der identifizierten Risiken (und Nicht-Risiken), der Menge der sensitiven Punkte und der Konfliktpunkte. Die Ergebnisse werden in Form eines ausführlichen Dokuments abgelegt. Es ist von Bedeutung, dass das Dokument die Resultate nicht nur benennt, sondern auch sorgfältig dokumentiert, wie man zu den Resultaten gelangt ist. Es muss später nachvollziehbar sein, mit welchen Argumenten welche Bereiche der Architektur wie eingeschätzt wurden. Dazu gehört eine präzise Darstellung der Geschäftsziele, der Qualitätsanforderungen in Form von Qualitätsattributbäumen, der evaluierten Architektur und die Bewertung, ob die Architektur den Anforderungen genügt. Das Abschlussdokument fungiert also als umfassende Architekturdokumentation.

2 Cost Benefit Analysis Method anhand einer Fallstudie

Für den Hersteller von Software-Systemen misst sich der Wert einer Software durch den Gewinn, der mir ihr erzielt werden kann. Stark vereinfacht gesprochen, setzt sich der Gewinn aus dem Software-Erlös minus den Produktions- und Wartungskosten zusammen. Insofern scheint es konsequent, Entscheidungen im Produktionsprozess so zu treffen, dass der Gewinn maximal ausfällt. Produziert werden sollte möglichst günstig, aber ausgerichtet an den Marktanforderungen.

Die Software-Architektur wird als entscheidendes Gebiet angesehen, um Entwurfsentscheidungen anhand von Kosten-Nutzen-Analysen zu treffen [KAK01, S.1]. Die hier vorgestellte Cost Benefit Analysis Method (CBAM) hat das Ziel, eine Menge von Architekturvorschlägen im Hinblick auf ihr Gewinnpotenzial zu untersuchen und eine Untermenge der Vorschläge, begrenzt durch limitierte finanzielle Ressourcen, als “beste” Investition zu empfehlen. Die “beste” Investition wird nicht allein durch ihren Gewinn bestimmt. Eine gute Investition ist ebenso gewinnbringend wie risikoarm [KAK01, S.1]. Das begründet den hohen Stellenwert der Risikobewertung in der CBAM. Es ist nicht möglich, den Nutzen einer Architekturänderung exakt zu ermitteln, also wird ein besonderes Augenmerk auf die Bewertung der Unsicherheit der Kalkulation gelegt.

Im nächsten Abschnitt wird der Kontext beschrieben, in den die CBAM eingebettet ist. Danach wird das Projekt vorgestellt, auf das die Autoren die Methode im Rahmen einer Fallstudie angewandt haben. Diese Fallstudie wird in dieser Arbeit dazu benutzt, das Vorgehen der CBAM anschaulich darzustellen. Danach werden die einzelnen Schritte der CBAM beschrieben: Beginnend

mit der Nutzen-Analyse, über die Kosten-Analyse, über die Berechnung des Ertrags bis hin zur Risikobewertung und Entscheidung wird die Vorgehensweise erläutert. Die einzelnen Schritte des Verfahrens werden also auf die jeweiligen Unterabschnitte aufgeteilt. Wie die einzelnen Schritte zu einem Prozess zusammengefügt werden, wird anschließend anhand Abbildung 3 erläutert.

2.1 Kontext

"Die CBAM beginnt mit der Annahme, dass die Geschäftsziele die Architekturentscheidungen beeinflussen" [MKKA03, S.1]. Das wird damit begründet, dass die Architekturvorschläge, nachfolgend auch Architekturstrategien genannt, sowohl technische als auch ökonomische Implikationen haben. Die technischen Implikationen beziehen sich auf die Eigenschaften des Systems, also die Qualitätsmerkmale. Die ökonomischen Implikationen einer Strategie beruhen nicht nur auf den Kosten der Implementierung, sondern auch auf dem Nutzen, der vom Erfüllungsgrad der Qualitätsanforderungen im Verhältnis zu den Geschäftszielen ausgeht [KAK02, S.2].

Zum Beispiel gilt für medizinische Überwachungssoftware, dass die Zuverlässigkeit eine bedeutende Rolle spielt. So wird angenommen, dass sich eine zuverlässigere Software besser verkaufen lässt (beziehungsweise der Software-Hersteller im Falle eines Ausfalls verklagt wird). Die Umsetzung von Architekturstrategien zur Erreichung von Zuverlässigkeit wiederum kostet Geld. Es könnte zum Beispiel eine Strategie sein, wichtige Systemfunktionen redundant auszulegen. Bei einem begrenzten Budget sollte abgeschätzt werden, welche Strategie den größten Gewinn verspricht [KAK02, S.2].

"Die CBAM beginnt, wo die ATAM endet" [AKK01, S.3]. Die obigen Überlegungen zeigen, dass die Ergebnisse der ATAM hier wiederverwendet werden können. Dazu gehören die Beschreibung der wichtigsten Geschäftsziele, eine Dokumentation der (existierenden oder vorgeschlagenen) Architektur, ein Qualitätsattributbaum zur Konkretisierung der Qualitätsanforderungen und die Mengen der Risiken, der sensitiven Punkte und der Konfliktpunkte.

2.2 NASA ECS

Zu Testzwecken und um Erfahrungen zu gewinnen haben die Autoren der CBAM das Verfahren im Rahmen einer Fallstudie angewandt. Bei dem zu analysierenden System handelte es sich um die Software des NASA ECS Projekts. Es ist ein verteiltes Informationssystem, das von Satelliten gesammelte Klimadaten zu jeder Zeit in aufbereiteter Form weltweit zur Verfügung stellt. Das Datenvolumen ist sehr groß und verdoppelt sich Jahr für Jahr: Pro Tag werden über 100 Gigabyte Daten erfasst, insgesamt verwaltet das System einige hundert Terabyte. Die Software besteht aus etwa 1,1 Millionen Zeilen Quelltext und befindet sich in der Wartungsphase [KAK01, S.302]. "Das ECS hat hohe Anforderungen an Leistung und Verfügbarkeit" [KAK02, S.9], auch die Wartbarkeit stellt eine wichtige Anforderung dar.

Die Aufgabe bestand darin, mit einem beschränkten Etat das bestehende System zu warten und zu ändern. Eine vorausgehende Analyse mit der ATAM lieferte Änderungswünsche und zugehörige Architekturstrategien. Die CBAM wurde angewandt, um eine Untermenge der umzusetzenden Strategien unter Beachtung des Budgets und des Gewinns auszuwählen [MKKA03, S.1].

2.3 Strategien sammeln und Nutzen quantifizieren

Die Bestimmung des Nutzen einer Architekturstrategie ist ein Hauptbestandteil der CBAM. Der Nutzen einer Strategie wird im Wesentlichen durch Einschätzung der Auswirkungen auf die Szenarien bewertet. In diesem Teil werden die einzelnen Schritte der CBAM vorgestellt, die zur Nutzenbewertung der Architekturstrategien führen [KAK02, S.5-7]. Zugleich wird beschrieben, wie die Schritte auf das NASA ECS Projekt angewandt wurden [KAK02, 9-15]. Dabei werden nur Auszüge der Daten beschrieben, die ausführliche Bewertung kann im Originaldokument [KAK02] nachgeschlagen werden.

1. Szenarien sammeln: In diesem Schritt werden sowohl die Szenarien, die bei der Anwendung der ATAM erstellt wurden, als auch neue Szenarien der Interessenvertreter aufgesammelt. Die Szenarien werden unter Beachtung der Geschäftsziele priorisiert, nur das wichtigste Drittel wird für die anschließende genauere Analyse verwendet.

In der NASA ECS Fallstudie existierten durch die vorherige Anwendung der ATAM bereits Szenarienbeschreibungen. Zusätzlich nannte das Bewertungsteam noch neue Szenarien. Die folgende Tabelle 1 zeigt zwei Beispielszenarien.

Szenario	**Beschreibung**
3	Anzahl der im Prozess der Auftragsübergabe fehlschlagenden Aufträge verringern.
5	Reduzieren der fehlschlagenden Aufträge, die zu verlorenen Aufträgen führen.
10	Das System sollte in der Lage sein, eine Benutzeranfrage der Größe 50 Gigabyte in einem Tag und eine Benutzeranfrage der Größe 1 Terabyte in einer Woche zu verarbeiten.
...	...

Tabelle 1. Exemplarische Szenarien

2. Szenarien verfeinern: Die Szenarien werden verfeinert. Genauer gesagt wird für jedes Szenario das Verhältnis zwischen Reiz und Reizantwort sichtbar gemacht. Dazu werden für jedes Szenario die schlechteste, die derzeitige, die erwünschte und die beste Reizantwort bestimmt.

Szenario	Semantik	Reizantwort			
		schlechteste	derzeitige	erwünschte	beste
3	Fehlschlag in %	10	5	1	0
5	Verlust in %	10	<1	0	0
10	Vorgabe erreicht in %	<50	60	80	>90
...	...	...	...	...	...

Tabelle 2. Verfeinerte Szenarien mit kategorisierten Reizantworten

Die Datensätze der Fallstudie dienen als Beispiel, um das Vorgehen zu veranschaulichen (siehe Tabelle 2). Für Szenario 3 gilt also, dass derzeit 5% der Aufträge im Prozess der Auftragsübergabe fehlschlagen, wünschenswert wäre jedoch eine Verlustrate von einem Prozent. Für Szenario 10 wird erwünscht, dass mindestens 80% der Benutzeranfragen in der genannten Zeitvorgabe bearbeitet werden.

3. Szenarien priorisieren: Hier werden die Szenarien priorisiert. Die CBAM liefert einen genauen Vorschlag, wie man bei der Gewichtung vorgehen kann [KAK02, S.5]. Nach welchem Verfahren tatsächlich priorisiert wird, ob im Rahmen einer Gruppendiskussion oder ob jedes Team-Mitglied einzeln wählt und anschließend ausgezählt wird, ist zweitrangig [KAK02, S.11]. Von Bedeutung ist, dass sich die Bewertung an der erwünschten Reizantwort, die im vorigen Abschnitt für jedes Szenario bestimmt wurde, orientiert.

Das Evaluierungsteam des NASA Projekts hat die Priorisierung in einer Gruppendiskussion vorgenommen. Dazu wurden insgesamt 100 Stimmen auf die einzelnen Szenarien verteilt. Die Szenarien wurden nur mit 5, 10, oder 15 Stim-

Szenario	Stimmen	Semantik	erwünschte Reizantwort
3	15	Fehlschlag in %	1
5	15	Verlust in %	0
10	5	Vorgabe erreicht in %	80
...	...	...	...

Tabelle 3. Verfeinerte Szenarien mit Priorisierung

men bewertet, eine höhere Genauigkeit wurde von den Interessenvertretern als "unnötig und [..] nicht gerechtfertigt" angesehen[MKKA03, S.4]. Wie dem Datenauszug zu entnehmen ist, wird es also als dreimal so wichtig erachtet, für Szenario 3 die erwünschte Reizantwort zu erreichen als für Szenario 10.

4. Nützlichkeit zuweisen: An dieser Stelle werden den kategorisierten Reizantworten der gewichteten Szenarien Nützlichkeiten zugewiesen. Die Nützlichkeit wird auf einer Skala von 0 bis 100 angegeben. Der Wert 0 bedeutet keine Nützlichkeit, der Wert 100 die höchstmögliche Nützlichkeit.

Dieser Schritt der CBAM lieferte bei der NASA ECS Studie [KAK02, S.11f] die in der folgenden Tabelle 4 aufgelisteten Bewertungen. Die Erreichung der

Szenario	**Stimmen**	**Nützlichkeit**			
		schlechteste	**derzeitige**	**erwünschte**	**beste**
3	15	25	70	100	100
5	15	0	80	95	100
10	5	0	70	90	100
...	...	...	...	...	...

Tabelle 4. Szenarien mit Nützlichkeiten der kategorisierten Reizantworten

schlechtesten Reizantwort von Szenario 10 hat noch eine Nützlichkeit von 25, während der schlechtesten Reizantwort von Szenario 5 gar keine Nützlichkeit zugeordnet wird. Abgebildet auf die Kategorisierung der Reizantworten in Schritt 2 bedeutet dies, dass für Szenario 3 eine Fehlerrate von 10% noch eine geringe Nützlichkeit aufweist, während für Szenario 5 eine Verlustrate von 10% gar keine Nützlichkeit mehr hat.

5. Architekturstrategien entwickeln und erwartete Auswirkung auf Qualitätsmerkmale bestimmen: Nun werden Architekturstrategien (ASn) für die Szenarien entwickelt beziehungsweise schon vorhandene Strategien gesammelt. Dann werden für diese Strategien die bei einer Implementierung erwarteten Reizantworten der Szenarien geschätzt. Die Zahl der Strategien ist unabhängig von der Zahl der Szenarien. Pro Strategie muss aber für jedes Szenario die erwartete Reizantwort ermittelt werden. Falls eine Strategie ein Szenario nicht berührt, entspricht die erwartete Reizantwort natürlich der derzeitigen Reizantwort.

Tabelle 5 stellt zwei Architekturstrategien vor, die im Rahmen der Studie erstellt wurden. Tabelle 6 zeigt die Auswirkungen der Strategien auf die Qualitätsmerkmale. Die Auswirkung wird durch erwartete Reizantworten von Szenarien beschrieben. Aus den Tabellen lässt sich entnehmen, dass erwartet wird,

AS	**AS Name**	**AS Beschreibung**
1	Aufträge persistieren	Speichern des Auftrags sobald er im System ankommt.
3	Aufträge bündeln	Mehrere kleine Aufträge zu einem großen Auftrag kombinieren.
...	...	...

Tabelle 5. entwickelte Architekturstrategien (ASn)

dass die Implementierung von Strategie 1 eine Verbesserung der Reizantworten der Szenarien 3 und 5 mit sich bringt. Für Strategie 3 hingegen wird bei

AS	beeinflusste Szenarien	Semantik	Reizantwort	
			derzeitige	erwartete
1	3	Fehlschlag in %	5	2
	5	Verlust in %	<1	0
	6	...	...	...
3	9	...	...	...
	10	Vorgabe erreicht in %	60	55
...	...	...	...	...

Tabelle 6. Von den Architekturstrategien (ASn) beeinflusste Szenarien

Implementierung eine leichte Verschlechterung der Reizantwort von Szenario 10 angenommen.

6. Nützlichkeit der erwarteten Reizantworten durch Interpolation bestimmen: Über die in Schritt 4 festgelegten Nützlichkeiten werden die Nützlichkeiten der erwarteten Reizantworten bestimmt. Die einzelnen Kategorien der Reizantworten (schlechteste, derzeitige, erwünschte, beste Reizantwort) stellen Stützpunkte einer Funktion dar, die für eine Reizantwort die Nützlichkeit bestimmt. Die Funktion wird durch Interpolation erzeugt. Anschließend kann die Nützlichkeit der erwarteten Reizantwort einfach durch Anwenden der Funktion bestimmt werden. Die folgende Zeichnung in Abbildung 2 stellt diese Vorgehensweise schematisch dar: Offensichtlich ist die interpolierte Funktion nicht

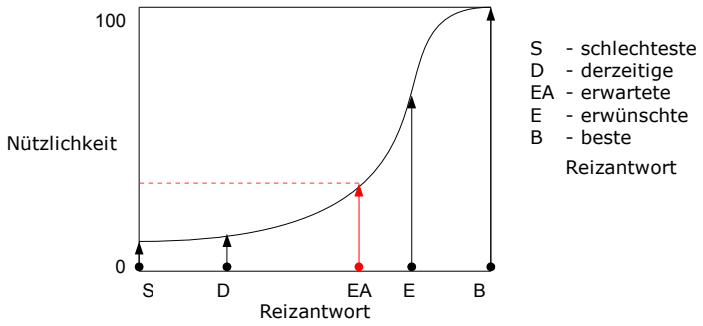


Abbildung 2. schematische Interpolation

eindeutig. So hätte man im obigen Beispiel auch eine Treppenfunktion definieren können. Wie die Funktion letztendlich bestimmt wird, hängt vom Kontext des zugehörigen Szenarios ab. Existieren für die Reizantworten beispielsweise Schwellwerte, so sollte an diesen Stellen der Anstieg der Kurve besonders steil sein. Schwellwerte sind Stellen, bei denen bereits kleine Abweichungen der Reizantwort einen großen Einfluss auf die Nützlichkeiten haben.

Die Kurven für Szenarien 1 (fehlerhafte Aufträge) und 3 (verloren gegangene Aufträge) könnte man beide als S-Kurven einordnen, wobei die Kurve für das Szenario 1 “stumpfer” sein wird als die Kurve für das Szenario 3 (die Bewertung der Nützlichkeit in Schritt 4 liefert die Begründung). So eine Kurve kann man auch als Zugehörigkeitsfunktion einer unscharfen Menge über dem Universum der Reizantworten interpretieren. Dann modelliert diese Menge die Zugehörigkeit der Reizantworten zur Nützlichkeit.

In der Fallstudie wurden den erwarteten Reizantworten die Nützlichkeiten aus Tabelle 7 zugeordnet.

AS	**beeinflusste Szenarien**	**Nützlichkeit**	
		derzeitige	**erwartete**
1	3	70	90
	5	70	100
	6	80	100
3	9	50	80
	10	70	65
...	...	...	...

Tabelle 7. Architekturstrategien (ASn) und deren erwartete Nützlichkeit

7. Nutzen einer Architekturstragie berechnen: Um den Nutzen einer Architekturstrategie zu berechnen, werden für die beeinflussten Szenarien zuerst die Differenzen zwischen den erwarteten und derzeitigen Nützlichkeiten berechnet. Diese Differenzen werden mit der Gewichtung aus Tabelle 3 normalisiert und anschließend zu einem Gesamtnutzen aufsummiert.

Die Tabelle 8 zeigt den Nutzen der Beispielstrategien der NASA ECS Studie: Die Szenariengewichtung entspricht den Stimmen aus Schritt 3. Der Gesamtnut-

AS	**beeinflusste Szenarien**	**Szenarien-gewichtung**	**Differenz der Nützlichkeiten**	**normalisierte Differenz**	**Nutzen der AS**
1	3	15	20	300	950
	5	15	30	450	
	6	10	20	200	
3	9	10	30	300	275
	10	5	-5	-25	
...	...	...	...	...	...

Tabelle 8. Gesamtnutzen der Architekturstrategien (ASn)

zen der Architekturstrategie 3 beispielsweise ergibt sich mit den Nützlichkeiten aus Tabelle 7 zu $(80-50)*10+(65-70)*5=275$. Im Vergleich dazu fällt der Gesamtnutzen der Strategie 1 mehr als dreimal so hoch aus.

2.4 Kosten quantifizieren

Die CBAM stellt keine Methodik zur Quantifizierung der Kosten einer Architekturstrategie bereit. Es wird angenommen, dass in dem betreffenden Unternehmen eine Kostenschätzungsmethode existiert. Solche Schätzmethoden existieren in der Software-Technik bereits länger, die meisten größeren Organisationen haben sogar angepasste Schätzverfahren entwickelt [AKK01, S.11]. Es muss jedoch beachtet werden, dass an dieser Stelle nicht nur die reinen Implementierungskosten zur Lösungsumsetzung berechnet werden. In die Kostenrechnung sollen auch mittel- und langfristige Kosten mit einfließen. An dieser Stelle spielen die Erfahrung und das Expertenwissen der teilnehmenden Interessenvertreter eine besonders große Rolle, weil keine Vorgaben gemacht werden.

Bezugnehmend auf die Beispieldaten des vorherigen Abschnittes 2.3 enthält Tabelle 9 die Kosten der Architekturstrategien, wie sie in der Fallstudie am NASA ECS Projekt geschätzt wurden. Das Team der Interessenvertreter hat die Kosten aus Erfahrungswerten geschätzt. Für alle Architekturstrategien befinden sich die geschätzten Kosten in dem Bereich von 100 bis 1200. Es wurde also keine Strategie teurer als Strategie 1 eingeschätzt.

2.5 Ertrag berechnen

Der Ertrag einer Architekturstrategie berechnet sich aus dem Verhältnis zwischen Nutzen und Kosten [KAK01, S.301]. In formaler Schreibweise ergibt sich für eine Strategie AS_i der Ertrag aus

$$\text{Ertrag}(AS_i) = \frac{\text{Nutzen}(AS_i)}{\text{Kosten}(AS_i)} \ .$$

Je größer der Nutzen, desto größer ist der Ertrag. Je größer die Kosten, desto kleiner ist der Ertrag. Man beachte, dass der Ertrag im Normalfall nicht in einer Währung vorliegt. Von Bedeutung ist jedoch, dass die Erträge verhältnisskaliert sind, also Abstände zwischen Erträgen interpretierbar sind und verschiedene Erträge in ein Verhältnis gesetzt werden können. Beispiel: “Der Ertrag von x ist doppelt so groß wie der Ertrag von y”.

In der NASA ECS Studie wurden die in der nachfolgenden Tabelle 9 gezeigten Erträge ermittelt [KAK02, S.15]. Unter allen untersuchten Architekturstrategien sind die Strategien 1 und 3 mit dem höchsten Ertrag, mit dem höchsten Return On Investment (ROI), bewertet worden. Der Bereich der Erträge der einzelnen

Architekturstrategie	**Kostenschätzung**	**Nutzen der AS**	**Ertrag der AS**
1	1200	950	0.79
3	400	275	0.69
...	...	...	...

Tabelle 9. ermittelte Erträge bei Umsetzung der Architekturstrategien (ASn)

Architekturstrategien erstreckt sich über das Intervall von 0.22 bis 0.79. Während Szenario 1 eine sehr teure Strategie ist, dafür aber auch einen sehr hohen Nutzen hat, gilt für Szenario 3, dass sowohl Kosten als auch Nutzen deutlich geringer sind.

2.6 Entscheidungen treffen

Die Quantifizierung des Nutzen und die Berechnung des Ertrags aus den vorangehenden Abschnitten 2.4 und 2.5 werden in der CBAM zu einem Schritt zusammengefasst [KAK02, S.6]. Die in Abschnitt 2.3 eingeführte Nummerierung wird fortgesetzt:

8. Architekturen unter Beachtung von Abhängigkeiten nach Ertrag auswählen: Schätze für jeden Architekturstrategie die Kosten der Umsetzung, und berechne den Ertrag als Verhältnis von Nutzen zu Kosten [KAK02, S.6]. Beachte dabei die Verflechtungen der Ablaufpläne, die sich bei der Umsetzung durch Abhängigkeiten der Strategien ergeben können. Wähle unter Beachtung des limitierten Budgets diejenigen Strategien mit den höchsten Erträgen.

9. Ergebnisse mit Intuition bestätigen: Die im vorherigen Schritt erstellte Auswahl sollte von den Interessenvertretern noch einmal dahingehend beurteilt werden, ob sie tatsächlich an den Geschäftszielen ausgerichtet ist. [KAK02, S.6]. Falls das nicht der Fall ist, sollte überprüft werden, ob es wichtige Sachverhalte gibt, die während der Analyse nicht beachtet wurden. Wenn solche Punkte gefunden werden, sollten die betreffenden Schritte iteriert werden. Für die Iteration und für eine Einbeziehung von Risiken in die Berechnung siehe den nächsten Abschnitt.

2.7 CBAM unter Berücksichtigung von Unsicherheiten anwenden

Die einzelnen Schritte der CBAM wurden in den vorherigen Abschnitten erklärt. An dieser Stelle wird erläutert, wie die Schritte zu einem Entscheidungsprozess zusammengefügt werden und Unsicherheiten behandelt werden können. Die CBAM ist eine iterative Methode, die einzelnen Schritte können also mehrmals durchlaufen werden [KAK02, S.5]. Abbildung 3 stellt die Vorgänge anschaulich dar. In der ersten Iteration wird eine "initiale Rangfolge" [KAK02, S.5] der Architekturstrategien (ASn) erstellt. Diese Rangfolge kann durch weitere Iterationen verfeinert werden. Dieses Vorgehen dient dazu, frühzeitig den Entscheidungsraum einzuschränken und dadurch Zeit zu sparen. In dem Diagramm werden im ersten Schritt N Szenarien gesammelt, wobei ab dem vierten Schritt nur noch die wichtigsten 1/6 N Szenarien weiter analysiert werden.

Eine weitere Iteration eignet sich, um detailliertere Analysen durchzuführen. "Informationen [..] über Risikoschätzung und Unsicherheit und die Zuteilung

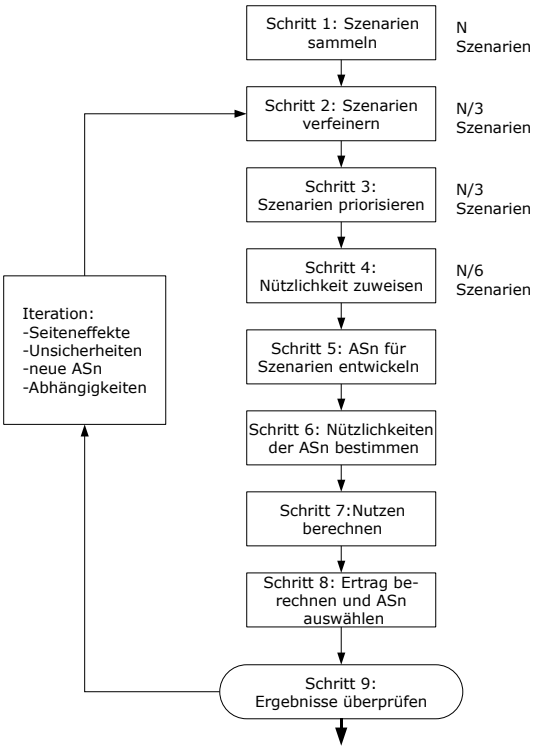


Abbildung 3. Flussdiagramm der CBAM [KAK02, S.8]

von Ressourcen" [KAK02, S.6] können hinzugefügt werden. Eine Berücksichtigung dieser Informationen können die Ergebnisse der ersten Iteration entscheidend verändern. Im Folgenden wird insbesondere die Behandlung der Risiken beschrieben.

Die CBAM basiert in der Kosten- und Nutzenanalyse auf einer Vielzahl von Variablen, die von den Interessenvertretern beurteilt werden. Das Einzige was als sicher angesehen werden kann, ist die Unsicherheit der ermittelten Variablenwerte [KAK02, S.28]. Ein Weg, die Unsicherheiten mit in die Bewertung einzubeziehen, ist eine Risikoanalyse. Die Autoren der CBAM teilen die Risiken in 4 Gruppen ein [KAK02, S.6]:

1. Risiken bei der Kostenschätzung.
2. Risiken bei der Ermittlung der Reiz/Reizantwort-Charakteristik der Szenarien und der damit verbundenen Schätzung der Nützlichkeit einer Strategie.
3. Risiken bei der Schätzung des Nutzen einer Strategie, die durch nicht erwogene Szenarien oder nicht erwogene Qualitätsmerkmale entstehen. Diese Risiken können Seiteneffekte bei der Umsetzung einer Architekturstrategie hervorrufen.
4. Risiken beim Projektmanagement und der Zeitplanung.

Diese Risiken können nicht vollständig ausgeschlossen werden. Daher kann die Annahme, dass die Kosten, die Kategorisierung der Reizantworten, die Nützlichkeiten etc. durch das Evaluierungsteam zuverlässig bestimmt werden, nicht aufrecht erhalten werden. Abhilfe wird in der CBAM dadurch geschafft, indem man keine einzelnen Werte, sondern Intervalle schätzt. Damit diese Schätzbereiche bestimmt werden können, wird jede Architekturstrategie auf Risiken untersucht. Die Risiken werden dokumentiert, kategorisiert, mit einer Wahrscheinlichkeit und einer Beschreibung der Auswirkungen versehen. Am Ende einer Iteration mit Risikobetrachtung steht keine sortierte Liste von Strategien, sondern ein Schätzer der Strategienrangfolge.

Die Studie, die am NASA ECS Projekt durchgeführt wurde, umfasst die Durchführung einer Risikoanalyse [KAK02, S.17-20]. Wie in den vorherigen Abschnitten folgen einige exemplarische Datensätze, die die obige Methode veranschaulichen. Tabelle 10 zeigt die Risiken der Architekturstrategien auf. Unter der Überschrift "Wahrscheinlichkeit" wird die Auftretenswahrscheinlichkeit verstanden. Die Spalte mit dem Titel "Auswirkung" enthält die erwarteten Auswirkungen im Falle des auftretenden Problems. Strategie 1 werden mehrere Risiken zugeordnet, während Strategie 3 als risikolos eingeschätzt wird. Sowohl Risiko T4 als auch S1 wurden der ersten Risikokategorie zugeordnet und beeinflussen die Kostenschätzung. Andere (hier nicht aufgeführte) Risiken beeinflussen die Einschätzung des Nutzen oder die Zeitplanung. Basierend auf diesen Wertungen wurden Nutzen und Kosten neu geschätzt, wobei dieses Mal Intervalle bestimmt wurden, die aus einer pessimistischen Schätzung resultieren. Es wird erwartet, dass die tatsächlichen Werte in dem geschätzten Intervall liegen, auch wenn Probleme auftreten. Die in der obigen Tabelle beschriebenen Auswirkungen im Falle eines Problems wurden also mit eingerechnet.

ID	Risiko	Wahrscheinlichkeit	Auswirkung	Auflösungsstrategie
AS 1: Aufträge bei Übergabe persistieren				
T4	Gewähltes DBMS könnte nicht mehr zur Verfügung stehen.	0.4	Hoch, Kostenerhöhung um 50%	DBMS ersetzen, Kostenerhöhung entspricht Änderung von 50% der SLOC (source lines of code)
S1	Neue Objektschnittstelle des DBMS könnte in der Entwicklung mehr Code als erwartet erfordern.	0.6	Hoch, Kostenerhöhung um bis zu 70%	keine
...	...	...	...	...
AS 3: Aufträge bündeln				
keine				
...				

Tabelle 10. Risiken für die Architekturstrategien

Daraufhin wurden die Erträge der Strategien neu berechnet. Die Erträge unterliegen nun einer Wahrscheinlichkeitsverteilung, in Tabelle 11 ist jedoch nur der jeweilige Erwartungswert E angegeben. Unter Annahme einer Normalverteilung

AS	Kosten		Nutzen		Ertrag		E(Ertrag)
	Min	Max	Min	Max	Min	Max	
1	1200	7152.75	850	950	0.119	0.79	0.454
3	360	440	247.5	302.5	0.563	0.84	0.701
...	...	...	...	...	...	...	...

Tabelle 11. Ertrag pro Architekturstrategie (AS)

der Erträge im ermittelten Ertragsintervall wurde anschließend ein Schätzer für die Strategierangfolge berechnet [KAK02, S.20]. Laut Tabelle 12 ist die Wahrscheinlichkeit, dass Strategie 3 einen höheren Ertrag als Strategie 1 liefert, gleich 0.77. Das bedeutetet umgekehrt, dass Strategie 1 nur zu 23% einen besseren Ertrag liefert als Strategie 3. Die Durchschnittswahrscheinlichkeit von 0.97, dass Strategie 3 besser ist als eine andere, zeigt deutlich den Einfluss der Einbeziehung der Unsicherheiten. Nach Tabelle 9 wurde Strategie 1 als besser eingestuft, nach der zweiten Iteration ist es Strategie 3. Anhand dieses Schätzers kann nun eine Auswahl der umzusetzenden Architekturstrategien vorgenommen werden. Falls den Entscheidern die Rangfolge-Wahrscheinlichkeiten nicht signifikant genug wären, könnte ebenso noch weiter iteriert und genauer analysiert werden, um zum Beispiel die Kostenschätzungen weiter zu präzisieren.

	AS1	AS3	...	Durchschnitt
AS1	0.5	0.23	...	0.68
AS3	0.77	0.5	...	0.97
...	...	...	...	...

Tabelle 12. $\mathbb{P}\{$ "AS(Zeile) liefert höheren Ertrag als AS(Spalte)" $\}$

3 Diskussion

In diesem Teil werden Fragen zum Nutzen der Methode gestellt. Welche an die Methode erhobenen Ansprüche werden tatsächlich erfüllt, welche nicht? Wie wendet man die CBAM konkret an? Die Diskussion findet in zwei Abschnitten statt. Im ersten Abschnitt werden die Erkenntnisse dargelegt, die die CBAM Autoren durch die Erfahrungen aus der Fallstudie am NASA ECS Projekt gewonnen haben. Im zweiten Abschnitt wird die Methode in einem größeren Kontext betrachtet, werden Erweiterungsvorschläge beleuchtet und Einschätzungen aus der Forschungsliteratur dargestellt.

3.1 Erfahrungen aus dem NASA Projekt ECS

Die im Teil 2 vorgestellte CBAM wird auch Version 2 der CBAM genannt. Die Vorversion, die initiale Version der CBAM, wurde in [AKK01] und [KAK01] veröffentlicht. Schon auf Grundlage dieser ersten Version wurde eine Fallstudie durchgeführt. Dabei handelte es sich ebenfalls um das NASA ECS Projekt. Die Erkenntnisse aus der ersten Anwendung dienten als Antrieb zur Erstellung der zweiten Version der CBAM. Anschließend wurde mit der zweiten Fallstudie überprüft, ob die Änderungen Wirkung zeigen. Zusammengefasst lässt sich also sagen: "CBAM Version 1 + 1.Fallstudie = CBAM Version 2".

Um diesen Entwicklungsprozess nachzuvollziehen, werden an dieser Stelle kurz das Vorgehen der ersten CBAM Version erläutert und die Erfahrungen der Autoren bei der Durchführung der ersten Fallstudie geschildert. Der Frage, ob die zweite CBAM Version eine Verbesserung bringt, wird genauso nachgegangen wie den allgemeinen Schwierigkeiten so einer Evaluierungsmethode.

CBAM Version 1 Während die Vorgehen zur Berechnung der Kosten und des Ertrags in beiden Versionen gleich sind, wurde in der ersten Variante der Nutzen einer Architekturstrategie auf andere Art und Weise quantifiziert: Zu Anfang wurden die Qualitätsmerkmale wie Leistung, Wartbarkeit etc. priorisiert, indem jedes Mitglied des Evaluierungsteams 100 Stimmen für die Wichtigkeit der Qualitätsmerkmale abgegeben hat. Dann schätzte jeder die Beiträge der Strategien zu den Qualitätsmerkmalen.

Seien $\text{Beitrag}_{i,j}$ die Einschätzung der Auswirkung der Strategie AS_i auf Qualitätsmerkmal QM_j auf einer Skala von -1 bis +1 und QMGewichtung_j die Prio-

rität von QM_j. Dann wurde der Gesamtnutzen einer Strategie mit

$$\text{Nutzen}(AS_i) = \sum_j (\text{Beitrag}_{i,j} \times \text{QMGewichtung}_j)$$

als Summe der gewichteten Auswirkungen auf die Qualitätsmerkmale berechnet [KAK01, S.300]. Jeder Interessenvertreter ermittelte also für jede Strategie den Nutzen und die Kosten. Die Differenzen zwischen den Bewertungen der Interessenvertreter wurden dann als Unsicherheiten aufgefasst [KAK01, S.300].

Es wurde angenommen, dass die Bewertungen der Interessenvertreter größtenteils übereinstimmen würden. Größere Abweichungen in einigen Bereichen wurden als Unsicherheit des Bereichs eingeschätzt. Da verschiedene Personen "Erfolgsmöglichkeiten verschieden" [KAK01, S.300] beurteilen würden, wurden diese Differenzen als "Charakterisierung der [..] Unsicherheit" [KAK01, S.300] angesehen und als "Hilfe, um Geschäftsbeschlüsse zu fassen" [KAK01, S.300]. Ob diese Annahmen realistisch sind, wird im folgenden Abschnitt untersucht.

Erkenntnisse aus der ersten Fallstudie Die CBAM hat den Interessenvertretern eine wiederholbare Methode zum Treffen von Architekturentscheidungen gegeben [KAK01, S.304]. Die Methode wurde als erfolgreich eingestuft, weil die Team-Mitglieder angeleitet wurden, Architekturstrategien zu erwägen, die sie ansonsten vielleicht übersehen hätten. Als ein Grund wird angeführt [KAK01, S.304]: Strategien, die keinen hohen Nutzen versprechen, aber dafür sehr wenig kosten, werden dennoch hohe Erträge zugeordnet. Weiterhin glauben die Autoren, dass die CBAM durch ihr rationales Bewertungsprinzip hilft, "die Tendenz der Interessenvertreter, sich auf kurzfristige Ziele zu fokussieren, zu kompensieren" [KAK01, S.305]. Es sei nämlich typisch, dass die Leute eher auf kurzfristige Gewinne schauen als den gesamten Lebenszyklus des Produkts im Auge zu haben [KAK01, S.304].

Da die Bewertungen der einzelnen Interessenvertreter unerwartet stark voneinander abwichen, wurden mögliche Gründe für diesen Sachverhalt gesucht. Laut [MKKA03, S.2] gibt es zwei Aspekte, die unterschiedliche Bewertungen erklären können. Zum Einen können tatsächliche Meinungsverschiedenheiten unter den Team-Mitgliedern existieren. In diesem Fall würden die verschiedenen Bewertungen die Unsicherheit widerspiegeln. Zum Anderen könnten die unterschiedlichen Bewertungen auf unterschiedliche Verständnisse der Architekturstrategien und Qualitätsmerkmale zurückzuführen sein. In diesem Fall würden die Interessenvertreter schlicht verschiedene Dinge bewerten. So wurde es im Nachhinein als schwierig eingestuft, Auswirkungen auf abstrakte Konzepte wie Qualitätsmerkmale zu bestimmen. In der Studie wurden die Qualitätsmerkmale von den Experten "von einem Kontext zum nächsten" [MKKA03, S.2] unterschiedlich verstanden.

Zu Anfang erschien es den Autoren als offensichtlich, dass Interessenvertreter den Effekt einer Systemänderung quantifizieren können. Es wurde entdeckt, dass die unterschiedlichen Wertungen auch aus unterschiedlichem Verständnis des Ist-Zustands des Systems resultierten [MKKA03, S.3]. Bei großen Systemen sei es

typisch, dass ein Interessenvertreter nur in einem Teilgebiet Experte ist. Da er den Ist-Zustand des Gesamtsystems nicht genau kennt, könne er auch die Auswirkungen von Änderungen nicht richtig beurteilen.

Ursprünglich wurde angenommen, dass der Hauptnutzen der CBAM darin besteht, "einen disziplinierten Prozess" [MKKA03, S.3] zu definieren. Es stellte sich jedoch heraus, dass die von der Methode geförderten Dialoge und Diskussionen untereinander ebenso wichtig waren.

Änderungen in der zweiten Version der CBAM Um die oben beschriebenen Verständnisprobleme zu beheben, wurden Szenarien für die Qualitätsmerkmale und Kurven zur Ermittlung der Nützlichkeit der Reizantworten der Szenarien eingeführt. Diese Maßnahmen dienen dazu, die Auswirkungen der Architekturstrategien auf die Qualitätsmerkmale verständlicher zu machen. Das Verständnis des Ist-Zustandes wird verbessert, indem Bewertungen vermehrt im Rahmen von Gruppendiskussionen getroffen werden. Weiterhin wurde der Evaluierungsprozess als iterativer Prozess definiert, damit das Resultat Iteration für Iteration durch Hinzunahme von Informationen verfeinert werden kann. Die Iterationen sollen den Teilnehmern das Verständnis der Methode erleichtern. Durch vorläufige Zwischenresultate soll die Akzeptanz der CBAM gefördert werden.

In der zweiten Fallstudie konnten die Änderungen daraufhin überprüft werden, ob die Methode verbessert wurde. Die Autoren heben hervor, dass die Gruppendiskussionen durch die CBAM strukturierter und zielorientierter abliefen [MKKA03, S.6]. Die Interessenvertreter werden gezwungen, die Szenarien klar zu definieren, die Nützlichkeiten mit dem vorgegebenen Verfahren zu ermitteln, Risiken und deren Auswirkungen zu formulieren. Die Intuitionen der Team-Mitglieder müssen in explizite Bewertungen umgewandelt werden [KAK02, S.21], die Bewertungen werden objektiver.

Auch wenn die exemplarische Anwendung der CBAM gezeigt hat, dass sie bessere Ergebnisse als Ad-hoc-Prozesse liefert, können keine Aussagen zur Qualität der Methode gemacht werden. Eine Optimalität der Resultate kann nicht überprüft werden. Dennoch bewerten die Autoren die CBAM positiv, da sie ein die Entscheidungsfindung strukturierendes Werkzeug ist [MKKA03, S.6]. Für genauere Aussagen müssten weitere Studien durchgeführt werden [KAK02, S.21].

Insgesamt wurde festgestellt, dass bei realen Projekte das Herauslösen von Informationen mit großen Schwierigkeiten verbunden ist [MKKA03, S.6]. So gibt es beispielsweise ein Zeitproblem. Die Interessenvertreter haben typischerweise wenig Zeit zur Verfügung. Die Methode muss von den Teilnehmern also schnell akzeptiert werden und effizient sein. Diese Probleme treten jedoch bei jeder Architekturanalysemethode auf.

Allgemeine Schwierigkeiten einer Architekturanalysemethode Einige Schwierigkeiten, die nicht nur für die CBAM im Speziellen, sondern für alle Architekturanalysemethoden gelten, werden an dieser Stelle aufgelistet. So wird in [KKC00, S.63] jede Architekturanalysemethode als "Müll-rein-Müll-raus-Prozess" beschrieben. Sie hängen von dem Mitwirken der Teilnehmer ab. Die

Qualität der Ergebnisse hängt massiv von der Motivation und der Erfahrung der Team-Mitglieder ab. Es zeigt sich, dass die Lösung so eines Analyseproblems in der Praxis deutlich komplexer ist als in der Theorie [KAK02, S.21].

Ein weiteres Problem ist, dass die Teilnehmer das System zu ihren Gunsten "spielen" können, sobald sie die Analysemethode verstanden haben. Der Grund hierfür könnte sein, eigene Forderungen, in der CBAM zum Beispiel eigene Architekturstrategien, durchzusetzen. Durch Gruppendiskussionen kann das teilweise unterbunden werden. Verfahren, die auf einen Gruppenkonsens zielen, können aber ebenfalls durch dominante Personen beeinflusst werden [MKKA03, S.3].

3.2 Weitere Erfahrungen und Kritik

Konkrete Zahlen über die Verbreitung der CBAM sind nicht bekannt. Es wird jedoch behauptet, dass die Methode einen "gewissen Erfolg" [KBK06, S.1] hat und mittlerweile von vielen Unternehmen in ihren Software-Entwicklungsprozess integriert wird. In [KLKB07, S.646] wird die CBAM als weit verbreitet und etabliert beschrieben.

Der Gesamtaufwand der CBAM ist verkleinerbar, indem man die CBAM in den Software-Entwicklungsprozess einbindet, anstatt jeweils separat die ATAM und anschließend die CBAM durchzuführen. Zu diesem Zweck wurde vom SEI ein Ansatz vorgestellt, die ATAM und die CBAM zu integrieren [NBC03]. Nicht nur die ATAM und die CBAM, sondern auch weitere am SEI entwickelte Architekturanalysemethoden wurden in [KNK03] dahingehend untersucht, wie sie zusammenwirken und sich auf den Lebenszyklus einer Software-Entwicklung zuschneiden lassen. Die Autoren kommen zu dem Schluss, dass die Methoden an unterschiedlichste Entwicklungsprozesse anpassbar sind. Durch Anwendung der Methoden werden die Entwicklungsprozesse zwar aufwändiger, resultieren aber in einer Architektur, die auf disziplinierte Art und Weise "entworfen, dokumentiert, analysiert und entwickelt" [KNK03, S.22] wurde. Wie sich die Architekturanalysemethoden in ein Vorgehensmodell einbinden lassen, wird in [KKNT04] für den Rational Unified Process (RUP) exemplarisch vorgestellt.

Die Unsicherheiten, die durch subjektive Fehler erzeugt werden, werden als Schwäche der Evaluierungstechnik identifiziert. In [KLKB07, S.646] wird der Entscheidungsprozess der CBAM als schwergewichtig und als in der Praxis nur schwierig umsetzbar eingestuft. Der Artikel schlägt einen leichtgewichtigen Prozess namens LiVASAE vor, der es ermöglicht, technische Entscheidungen unter Beachtung von Geschäftszielen, Kosten und Marktbedingungen zu treffen. Anschließend wurde eine Aufwandsanalyse durchgeführt, indem LiVASAE und CBAM auf das gleiche Projekt angewandt wurden. Es wurden vier verschiedene Architekturstrategien betrachtet. Die mit LiVASAE ermittelten Rangfolgen waren mit den Ergebnissen der CBAM "konsistent" [KLKB07, S.648], LiVASAE wurde als geringfügig effizienter als die CBAM eingeschätzt. Da es sich bei der Studie nur um ein einjähriges, kleineres Entwicklungsprojekt handelte, ist die Übertragbarkeit dieser Studie in Frage zu stellen.

4 Fazit und Ausblick

In der Praxis wird man in der Software-Entwicklung mit der Fragestellung konfrontiert, wie limitierte Ressourcen möglichst effektiv eingesetzt werden können. Der Nutzen des Software-Produkts sollte erhöht werden, die Kosten für die (Weiter-)Entwicklung gering bleiben. Die Fragestellung bezieht sich jedoch nicht nur auf die Einführung neuer Funktionalität. Es ist zwar notwendig, die "richtige" Funktionalität im Hinblick auf die funktionalen Anforderungen zu implementieren, aber nicht hinreichend. Es ist entscheidend, inwieweit die Software die von den Geschäftszielen bestimmten Anforderungen an die Qualitätsmerkmale erfüllt. Da angenommen wird, dass die Architektur einen sehr großen Einfluss auf die Software-Qualität hat, erscheint die Ebene der Architektur für eine ökonomische Analyse gut geeignet.

Die CBAM beschreibt einen strukturierten Prozess für Architekturentscheidungen. Sie definiert einen iterativen Prozess in Verbindung mit einem Rahmenwerk für das Bewertungsverfahren. Zur Bewertung werden Nutzen und Kosten von Architekturvorschlägen gegenübergestellt. Unter Betrachtung von Risiken wird eine Rangfolge der Vorschläge etabliert, die auf dem zu erwartenden Ertrag bei der Umsetzung der jeweiligen Architekturstrategie basiert. Von dieser Rangfolge ausgehend werden die Entwurfsentscheidungen getroffen.

Die Autoren bezeichnen die CBAM aufgrund der Erfahrungen, die im Kontext zweier Fallstudien gemacht wurden, als gewinnbringendes Verfahren. So wird der Prozess der Entscheidungsfindung strukturiert und die Bewertung der Architekturstrategien durch Einarbeitung von Szenarien und Kurven zur Ermittlung der Nützlichkeit objektiviert. Insbesondere profitieren die Gruppendiskussionen durch die letztgenannten Maßnahmen, da unscharfe persönliche Einschätzungen nun klar artikuliert werden müssen. Da die Ergebnisse auf den Einschätzungen der teilnehmenden Experten beruhen, können keine Angaben zur Optimalität gemacht werden. Weitere empirische Studien sind notwendig, um weitere Erfahrungen zu machen und stärkere Aussagen über die Effizienz der Methode zu ermöglichen.

Damit das Verfahren weniger Zeit beansprucht, ist eine Integration in bestehende Prozessmodelle wünschenswert. Durch engere Bindung an bestehende Modelle wird eine Effizienzsteigerung erwartet, da der Aufwand für die bewertenden Expertenteams geringer wird.

Eine Methodik zur Kostenschätzung wird in der CBAM nicht vorgeschrieben. Es wird angenommen, dass so eine Schätzmethode im betreffenden Unternehmen oder der betreffenden Organisation bereits existiert. Da viele Verfahren zur Kostenschätzung nur die Kosten der Implementierung bewerten, erscheint es wünschenswert, eine Schätzmethode auf Architekturebene bereitzustellen, die auch die langfristigen Kosten (zum Beispiel durch Wartung) abdeckt [KAK01, S.298].

Literatur

AKK01. Jai Asundi, Rick Kazman, and Mark Klein. Using economic considerations to choose among architecture design alternatives, cmu/sei-2001-tr-035. Technical report, Software Engineering Institute, Carnegie Mellon University, 2001.

BCK03. Len Bass, Paul Clements, and Rick Kazman. *Software architecture in practice, Second Edition*. Addison-Wesley, Boston, 2003.

KAK01. Rick Kazman, Jai Asundi, and Mark Klein. Quantifying the costs and benefits of architectural decisions. In *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, pages 297–306, Washington, DC, USA, 2001. IEEE Computer Society.

KAK02. Rick Kazman, Jai Asundi, and Mark Klein. Making architecture design decisions: An economic approach, cmu/sei-2002-tr-035. Technical report, Software Engineering Institute, Carnegie Mellon University, 2002.

KBK06. Rick Kazman, Len Bass, and Mark Klein. The essential components of software architecture design and analysis. *J. Syst. Softw.*, 79(8):1207–1216, 2006.

KKC00. Rick Kazman, Mark Klein, and Paul Clements. ATAM: A method for architecture evaluation, cmu/sei-2000-tr-004. Technical report, Software Engineering Institute, Carnegie Mellon University, 2000.

KKNT04. Rick Kazman, Philippe Kruchten, Robert L. Nord, and James E. Tomayko. Integrating software-architecture-centric methods into the rational unified process, cmu/sei-2004-tr-011. Technical report, Software Engineering Institute, Carnegie Mellon University, 2004.

KLKB07. Chang-Ki Kim, Dan-Hyung Lee, In-Young Ko, and Jongmoon Baik. A lightweight value-based software architecture evaluation. In *SNPD '07: Proceedings of the Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2007)*, pages 646–649, Washington, DC, USA, 2007. IEEE Computer Society.

KNK03. Rick Kazman, Robert L. Nord, and Mark Klein. A life-cycle view of architecture analysis and design methods, cmu/sei-2003-tn-026. Technical report, Software Engineering Institute, Carnegie Mellon University, 2003.

MKKA03. Mike Moore, Rick Kazman, Mark Klein, and Jai Asundi. Quantifying the value of architecture design decisions: lessons from the field. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 557–562, Washington, DC, USA, 2003. IEEE Computer Society.

NBC03. Robert L. Nord, Mario R. Barbacci, Paul Clements, Rick Kazman, Mark Klein, Liam O'Brien, and James E. Tomayko. Integrating the architecture tradeoff analysis method (ATAM) with the cost benefit analysis method (CBAM), cmu/sei-2003-tn-038. Technical report, Software Engineering Institute, Carnegie Mellon University, 2003.

SG96. Mary Shaw and David Garlan. *Software architecture: perspectives on an emerging discipline*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

Übersicht über Reifegradmodelle

Thomas Wurth

Betreuer: Christoph Rathfelder

Zusammenfassung Diese Arbeit gibt einen Überblick über verschiedene Arten von Reifegradmodellen. Nach einer kurzen Einführung in das Thema werden einige Modelle vorgestellt. Anschließend soll auf die Vorteile eingegangen werden, die Reifegradmodelle sowohl für Softwarehersteller, als auch für deren Kunden bieten. Hierbei werden speziell Modelle aus dem Bereich der Softwareentwicklung betrachtet. Darauf folgt eine Darlegung verschiedener bekannter Kritikpunkte. Abschließend wird die Arbeit zusammengefasst und eine kurze persönliche Einschätzung des Beitrages gegeben, den Reifegradmodelle der Softwareentwicklung bieten.

1 Einführung

In Sachen Qualitätsmanagement gab es in den letzten Jahren einen Wandel. Durch vermehrten Einsatz konstruktiver und prozessbezogener Ansätze wird das aufwändige Testen nach Fertigstellung der Produktentwicklung entlastet [Kne06]. Während bei den konstruktiven Ansätzen (z.B. Programmierrichtlinien) versucht wird, Produktfehler schon von Anfang an während der Entwicklung zu vermeiden [Kne06], bauen die prozessbezogenen Ansätze auf die Annahme, dass sich die Qualität des Entwicklungsprozesses unmittelbar auf die Produktqualität auswirkt und diese maßgeblich bestimmt. Dieses prozessbezogene Qualitätsmanagement hat daher die Bewertung des Entwicklungsprozesses in Bezug auf seine Qualität und eine darauf basierende stetige Prozessverbesserung als Ziel.

Das zentrale Werkzeug für die Bewertung von Entwicklungsprozessen nach deren Qualität sind diverse Reifegradmodelle. Diesen liegen zunächst einmal sog. „Best Practices“ zugrunde [Hin04]. Das sind Praktiken, die sich nach langen Jahren der Anwendung bewährt haben und von denen man weiß, dass ihre Implementierung zu einer deutlichen Verbesserung von Faktoren wie z.B. der Qualität oder Termin- und Budgeteinhaltung führt. Zu Erwähnen ist allerdings, dass die Praktiken meist nur gefordert werden und nicht erklärt wird, wie sie zu implementieren sind.

So gesehen stellen Reifegradmodelle eine Sammlung von Verbesserungspunkte dar, welche Unternehmen ihren Bedürfnissen entsprechend anpassen, um ihren Entwicklungsprozess zu verbessern. Eine entsprechende individuelle Anpassung der Prozesse ist aufgrund der sehr allgemein gehaltenen Beschreibung jedoch unerlässlich, z.T. ist nicht einmal eine Beschreibung vorhanden. Wie jedoch schon angesprochen steckt noch mehr hinter dem Konzept Reifegradmodell. Der zweite Zweck den es erfüllt, ist die Beurteilung des Entwicklungsprozesses eines Unternehmens und dessen Einstufung in eine - je nach Modell unterschiedlich -

normierte Skala. Dazu ist das Modell in verschiedene Reifegradstufen gegliedert, welchen jeweils bestimmte „Best Practices“ zugeordnet sind. Die Stufen repräsentieren das jeweilige evolutionäre Stadium, in dem sich ein Entwicklungsprozess befindet. Entsprechend diesen Stufen bauen auch die ihnen jeweils zugeordneten Gruppen von Praktiken aufeinander auf. Auf diese Art wird dem Management klar ersichtlich, wo das Unternehmen derzeit steht und welche nächsten Schritte erforderlich sind, um die Prozessverbesserung möglichst effektiv voran zu treiben.

Die Bewertung und Einstufung erfolgt durch sog. Assessments. Ein Assessment untersucht die Abläufe in einem Unternehmen und vergleicht sie mit den Anforderungen des entsprechenden Reifegradmodells. So wird die Reifegradstufe für das Unternehmen oder ggf. für einzelne Prozesse ermittelt. In manchen Assessments werden sogar Stärken und Schwächen der Prozesse ausgearbeitet, wobei deutlich wird, welche einzelnen Verbesserungsschritte in der Zukunft noch durchzuführen sind.

2 Arten von Reifegradmodellen

Reifegradmodelle lassen sich in den verschiedensten Bereichen einsetzen. Ein großes Gebiet, in dem sie Verwendung finden, ist die Softwareentwicklung, worauf sich auch diese Arbeit vorwiegend bezieht. Neben bekannten Modellen für Softwareprozesse wie CMM (Capability Maturity Model) und SPICE existieren aber auch zahlreiche Abwandlungen. So gibt es beispielsweise auf Architekturen maßgeschneiderte Reifegradmodelle wie ACMM (IT Architecture Capability Maturity Model) [Gro06,oC03], sowie speziell für Service-Orientierte Architekturen entwickelte wie NSOAMM (New SOA Maturity Model) [Son07] oder ESOMM (Enterprise Service Oriented Maturity Model) [Ser]. Diese Modelle werden im folgenden Kapitel kurz vorgestellt. Außerdem existieren noch weitere, wie z.B. auf Mitarbeiter ausgelegte Modelle, die deren Produktivität in Stufen einordnen. Hierauf soll aber in diesem Paper nicht genauer eingegangen werden.

2.1 CMM

Nach der Software Krise der 70er und 80er Jahre, in der sich katastrophale Misserfolge in Softwareprojekten immer mehr häuften, war der Wunsch nach einer Idee oder einem Modell groß, um derartige Fehlschläge in Grenzen zu halten. Watts Humphrey machte 1987 schließlich den Vorschlag, bewährte Praktiken der Industrie in einem Modell zu bündeln. Dies war ein Schritt in Richtung einheitlicherer Software-Entwicklung, der schlecht organisiertem Projektmanagement entgegenwirken sollte. Das Modell nannte sich Capability Maturity Model (CMM) [CMU95] und wurde am Software Engineering Institute (SEI) der Carnegie Mellon University entwickelt. Nachdem die ursprüngliche Version 1.0 im August 1991 erschien, folgte im Februar 1993 die überarbeitete Version 1.1 [CMU93], die noch bis heute gültig ist.

Die Motivation zur Entwicklung von CMM geht z.B. aus dem Vorwort des technischen Berichts zu CMM Version 1.1 vor. Hier heißt es: *„Anhaltende Verbesserung kann nur durch [...] Bau einer Prozessinfrastruktur oder effektiver Software-Engineering- und Management-Praktiken geschehen“* [CMU93]. Ein weiteres Zitat besagt: *„Das SEI [...] begann mit der Entwicklung eines Prozess-Reife-Frameworks [...] als Antwort auf eine Anfrage, der bundesstaatlichen Regierung eine Methode zu bieten, um die Reife ihrer Software-Lieferanten zu beurteilen“* [CMU93]. Hier wird z.B. die Ausrichtung von CMM auf das Management deutlich, sowie auf Software-Prozessmodelle und die Bewertung von Software-Lieferanten [Pad07].

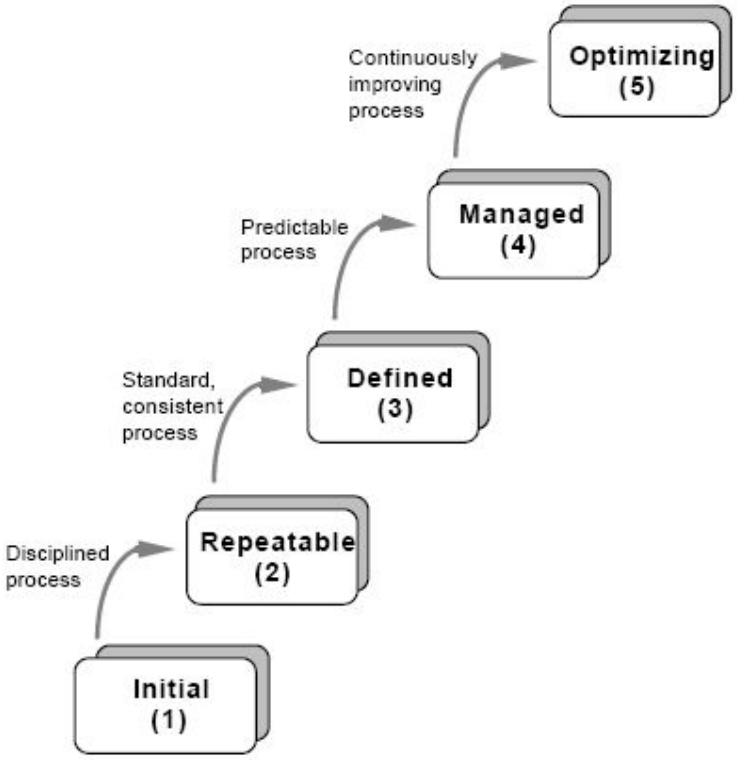


Abbildung 1. Stufen von CMM [CMU93]

Stufe 1 (initial): Unternehmen dieser Stufe haben keinen festgelegten Entwicklungsprozess, es gibt keine Kontrolle über den Projektfortschritt und entstehende Kosten. Der Projektablauf ist chaotisch.

Stufe 2 (repeatable): Der Entwicklungsprozess ist in Arbeitspakete unterteilt, einzelne Aktivitäten innerhalb dieser Arbeitspakete sind jedoch nicht festgelegt

oder überschaubar. Der Fortschritt kann somit nur beim Erreichen von Meilensteinen genau festgestellt werden. Es gibt ab dieser Stufe Schlüsselprozessbereiche, die in allen Projekten des Unternehmens implementiert sind. Schlüsselprozessbereiche (Key Process Areas/KPAs) sind die Bereiche, auf die sich das Management für eine Prozessverbesserung konzentrieren muss. Sie beinhalten die für die nächste Stufe zu erfüllenden Punkte.

Stufe 3 (defined): Die einzelnen Aktivitäten sind detailliert festgelegt, was den Projektfortschritt sehr transparent macht. Das Management hat also einen genauen Überblick über die technischen Fortschritte aller Projekte. Alle Prozesse sind für das gesamte Unternehmen standardisiert.

Stufe 4 (managed): Die Aktivitäten werden gemessen. Dadurch lässt sich der Prozess gezielt steuern, die Variation nimmt ab.

Stufe 5 (optimizing): Prozessverbesserungen werden systematisch durchgeführt. Kosten, Dauer und Schwankungen der Softwareprozesse werden laufend besser.

2.2 CMMI

Aus dem ursprünglichen CMM für Software entwickelten sich mit der Zeit Reifegradmodelle für andere Bereiche (z.B. integrierte Produktentwicklung), weshalb man zur Differenzierung bei dem Modell aus 2.1 auch von Software CMM spricht. Im Jahre 1997 wurde das Nachfolgeprojekt CMMI (Capability Maturity Model Integrated) [LL07] ins Leben gerufen. Die Bezeichnung kommt daher, dass es die verschiedenen Arten von CMM in sich integriert. CMMI ist also nicht mehr ausschließlich auf Software spezialisiert wie CMM und damit auch auf allgemeinere Bereiche anwendbar[Hin04, S.186].

Der erste Entwurf wurde 1998 als Version v0.2 veröffentlicht, die momentan aktuellste Version ist v1.2. Laut SEI handelt es sich bei CMMI um eine „Product Suite“[CMU08], die aus Modellen, Assessmentmethoden und Trainingsprodukten besteht [Hin04, S.186]. Die Modelle haben zwei verschiedene Dimensionen:

1. Dimension: Disziplin
- Software Engineering
- Systems Engineering
- Integrated Product and Process Development (IPPD)
- Supplier Sourcing (SS)

2. Dimension: Repräsentationsform
- Staged Representation
- Continuous Representation

Die vier Disziplinen der ersten Dimension sind sich gegenseitig bis auf einige kleinere Anpassungen sehr ähnlich. Ein Modell ist in beiden Repräsentationsformen der zweiten Dimension erhältlich. Während die Staged Representation wie

Software CMM in Stufen aufgebaut ist, denen die Prozessbereiche fest zugeordnet sind, ist die Continuous Representation flexibler strukturiert. Die Prozessbereiche sind hier nach Funktionalität in Gruppen unterteilt, und Prozesse können in einem Assessment beliebig gewählt und jeweils einer eigenen Reifegradstufe zugeteilt werden - unabhängig von den anderen Prozessen. Diese Idee wurde von dem im folgenden Abschnitt vorgestellten Reifegradmodell SPICE übernommen, um Kompatibilität zu diesem zu schaffen.

Da CMMI das ältere Modell CMM ablöst, wird dieses nicht mehr weiterentwickelt und das SEI stellte seine Schulungen Ende 2003 ein. Schulungen und Assessments können aber dennoch weiterhin durchgeführt werden, jedoch nur noch von Partnern des SEI.

2.3 SPICE

1993 wurde von den Organisationen ISO und IEC das Projekt ISO/IEC 15504 [Hin04,LL07] gestartet, auch bekannt unter dem Namen SPICE. Man nahm sich dabei CMM als Vorbild, jedoch wurde wie schon angedeutet einiges geändert.

Vergleichen lässt sich SPICE mit CMM in Bezug auf seine Reifegrade. Hier gibt es allerdings 6 Reifegradstufen, die zusätzliche Stufe wurde zwischen Stufe 1 und 2 von CMM eingefügt [Hin04] und verlangt, dass die „Base Practices“ (die grundlegenden Praktiken) gelebt werden, während die neue Stufe 2 zusätzliche Management Praktiken fordert. Ab hier blieb man weitgehend bei dem CMM-Stufenmodell.

Nach [LL07] zeichnen sich die Stufen durch folgende Eigenschaften aus:

Stufe 0 (incomplete): Es gibt keinen festgelegten Prozess, oder dieser wird nicht eingehalten.

Stufe 1 (performed): Ein Prozess existiert, wird angewendet und liefert die geforderten Ergebnisse.

Stufe 2 (managed): Der Prozess wird systematisch umgesetzt: Er wird geplant, überwacht und angepasst.

Stufe 3 (established): Ein einheitlicher, definierter und dokumentierter Prozess wird im gesamten Unternehmen verwendet, der aber noch an spezielle Eigenschaften der jeweiligen Projekte angepasst werden kann.

Stufe 4 (predictable): Die Qualität von Prozess und Produkten wird durchgehend überwacht.

Stufe 5 (optimizing): Der Prozess wird laufend optimiert. Bei Anzeichen für Qualitätseinbußen werden Gegenmaßnahmen ergriffen.

Der Unterschied zu CMM ist die Idee, die später von CMMI aufgegriffen wurde, nämlich das Lösen der Kopplung der Prozesse an bestimmte Reifegrade. So kann ein beliebiger Prozess ausgewählt und für ihn ein individueller Reifegrad ermittelt werden. Auch die Verbesserungsreihenfolge ist nicht mehr an Stufen gebunden, so dass sich das Unternehmen selbst aussuchen kann, in welcher Reihenfolge die Prozesse verbessert werden sollen.

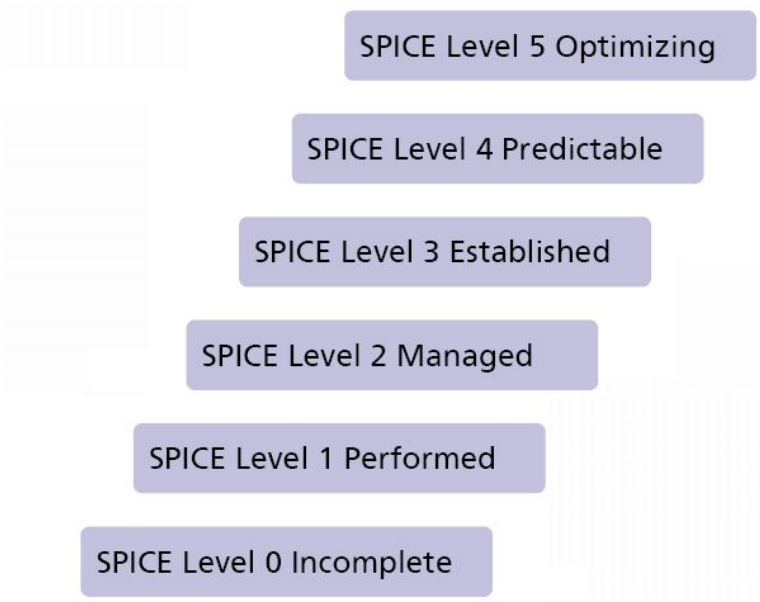


Abbildung 2. Stufen von SPICE [Spr07]

2.4 ACMM

Mit der zunehmenden Bekanntheit von Reifegradmodellen entstand unweigerlich der Wunsch, diese kontrollierte Prozessverbesserung auch auf IT Architekturen zu übertragen. So entstanden Schablonen-Tools, die eine IT Architektur, den Status des Architektur-Prozesses und die Investitionen einer Organisation in die Architektur erfassen. Das Department of Commerce (DoC) entwickelte schließlich das IT Architecture Capability Maturity Model (ACMM) [Gro06,oC03], das auf CMM zurückgreift und ein Framework bietet, das die wichtigen Komponenten eines gewinnbringenden IT Architektur-Prozesses enthält. Schwachpunkte in Architekturen sollen identifiziert und gezielt ausgemerzt werden. Das ACMM besteht aus drei Bereichen, die in den folgenden Unterabschnitten vorgestellt werden.

Das IT Architecture Maturity Model Dieser erste Bereich listet die sechs Stufen des DoC ACMM auf.

Stufe 0: None
Stufe 1: Initial
Stufe 2: Under Development
Stufe 3: Definded
Stufe 4: Managed
Stufe 6: Measured

IT Architektur Charakteristiken Der zweite Bereich beschreibt neun verschiedene Charakteristiken einer IT Architektur, welchen in einem Assesment jeweils eine separate Stufe zugeordnet wird.

1. Architecture Process
2. Architecture Development
3. Business Linkage
4. Senior Management Involvement
5. Operating Unit Participation
6. Architecture Communication
7. IT Security
8. Governance
9. IT Investment and Acquisition Strategy

Die ACMM Scorecard Die Scorecard schließlich dient der formellen Erfassung des Reifegrades. Sie verwendet zwei sich ergänzende Vorgehensweisen, um den Reifegrad zu errechnen. Die erste bestimmt das gewichtete Mittel des Reifegrades, wohingegen bei der zweiten festgehalten wird, zu welchem Prozentanteil ein Reifegrad für jede der neun Charakteristiken erreicht ist. Somit können Defizite aufgedeckt werden und es wird deutlich, welche Schritte zur Verbesserung der einzelnen Charakteristiken noch notwendig sind.

2.5 NSOAMM

Service-Orientierte Architekturen (SOAs) als Art und Weise, Geschäftsanwendungen zu designen, zu entwickeln und zu implementieren sind als eine sehr wichtige Entwicklung der letzten zehn Jahre anzusehen [Son07]. Ein Hauptargument hierfür ist die dadurch verbesserte Unterstützung von Geschäftsprozessen [CR07]. Eine Vorhersage von Gartner, Inc. besagt, dass SOAs im Jahre 2008 die Basis für 80% aller neuen Entwicklungen darstellen werden [Hay05].

Während die Vorteile einer SOA an sich für Manager in der IT Branche eher leicht zu sehen sind ist es für sie doch ungleich schwerer zu erkennen, für welche Technologie sich eine Investition am Meisten lohnt. Implementierte SOA-Lösungen sind schwer einzustufen, und ihr tatsächlicher Geschäftswert ist sehr undurchsichtig.

Aus diesem Grund entwickelten Sonic Software und dessen Partner das wohl bekannteste Reifegradmodell für SOAs, das New SOA Maturity Model (NSOAMM) [Son07]. Es zeigt den mit steigender Reifegradstufe zunehmenden Vorteil einer SOA für ein Unternehmen. Außerdem stellt es zum Einen ein Framework zur Verfügung, mit dem der strategische Wert einer SOA-Lösung für ein Unternehmen bestimmt werden kann. Zum Anderen bietet es ein Modell, mit Hilfe dessen sich das Unternehmen für die Zukunft darüber klar werden kann, wie die SOA optimal verbessert muss.

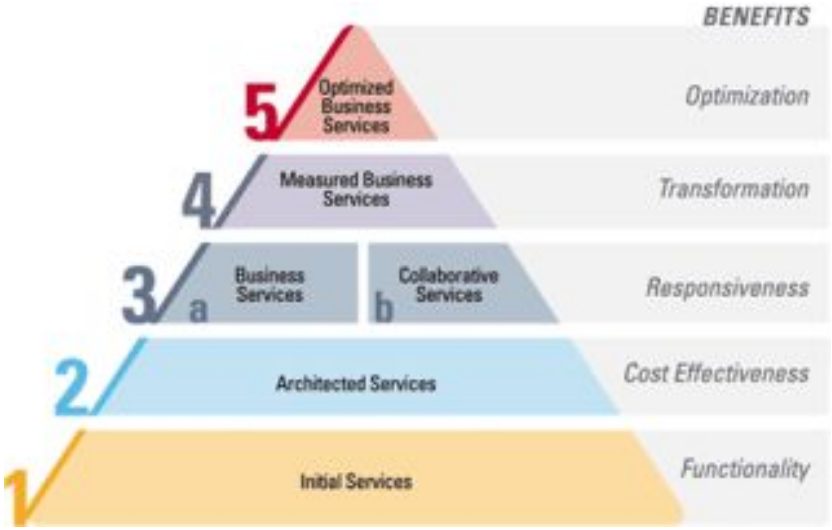


Abbildung 3. Reifegradstufen von NSOAMM und ihr Nutzen [Son07]

Die Stufen haben unterschiedlichen geschäftlichen Nutzen: Stufe 1 bringt dem Unternehmen neue Funktionalität, Stufe 2 reduziert die Kosten und bringt

sie unter Kontrolle. Während sich in Stufe 3a die Prozesse schnell und effektiv ändern lassen, kommt Stufe 3b der Zusammenarbeit mit Geschäfts- und Handelspartnern zugute. Stufe 4 erhöht die Anpassungsfähigkeit des Unternehmens an sich verändernde Anforderungen, und in Stufe 5 wird eine Geschäftsoptimierung durch selbstkorrigierende Geschäftsprozesse erreicht.

2.6 ESOMM

Ein weiteres Reifegradmodell für Service-Orientierte Architekturen ist das Enterprise Service Oriented Maturity Model (ESOMM) [Ser] von Microsoft. SOAs sind in ihrer Gesamtheit schwer zu erfassen, da es bis dato noch nicht einmal eine eindeutige Definition für sie gibt, was Vergleiche zwischen verschiedenen SOAs sehr erschwert. ESOMM soll diese Komplexität derart vereinfachen, dass die Anwendung einer SOA praktikabel wird. Es greift dabei wie viele andere Reifegradmodelle der verschiedensten Bereiche auch auf CMM zurück, da sich durch die Capabilities leicht Vergleiche anstellen lassen. Auch macht sich ESOMM zunutze, dass sich die Reifegradstufen von CMMI unabhängig der jeweiligen Capabilities bestimmen lassen, was der Adaptierbarkeit zugute kommt.

ESOMM reduziert die fünf Reifegradstufen von CMM/CMMI allerdings auf folgende vier:

Stufe 1 (usable): Das Unternehmen kann in großem Umfang standardkonforme Services schreiben und einsetzen.

Stufe 2 (repeatable): Das Unternehmen implementiert Services, nimmt sie auf und verwendet sie effizient und konsistent wieder.

Stufe 3 (supportable): Das Unternehmen kann eine steigende Anzahl an Services zu zugesicherten Service Level Agreements wirkungsvoll managen.

Stufe 4 (extensible): Das Unternehmen kann Services verbinden und ihre Verwendung über die eigenen Grenzen hinweg erweitern.

Außerdem berücksichtigt ESOMM drei Dimensionen einer SOA:

Dimension 1 (Implementation): Sie bezieht sich hauptsächlich auf Definition, Modellierung, Umsetzung und Publizieren von Services.

Dimension 2 (Consumption): Sie betrifft die Nutzung von Services.

Dimension 3 (Administration): Sie betrachtet Aspekte in Bezug auf das Management und die Richtlinientreue.

Die Reife wird anhand der Dimensionen ermittelt. Dies ergibt folgende Capabilities:

3 Nutzen von Reifegradmodellen am Beispiel CMM

3.1 Interner Nutzen

Reifegradmodelle werden mit einer bestimmten Erwartung eingesetzt: Unternehmen wollen so ihren Softwareprozess verbessern, um eine bessere Qualität ihrer

	Implementation	Consumption	Administration
Extensible	Service collaboration Service orchestration	Service SDKs External policy	Business analytics Automated policy management
Supportable	Versioning Executable policy Schema bank	Explicit SLAs Service portal Execution visibility	Auditing Monitoring Provisioning model
Repeatable	Common schema Service blocks	Self provisioning Service discoverability	Deployment management Enterprise policies
Usable	Design patterns Development processes	Testability Explicit contracts	Security model Basic monitoring

Abbildung 4. Capabilities von ESOMM [Ser]

Produkte zu erlangen. Um den internen Nutzen von Reifegradmodellen für ein Unternehmen darzulegen, wird nachfolgend basierend auf Aussagen aus [Pad07] der spezifische Nutzen der jeweiligen Stufen von CMM als wohl bekannteste und am Meisten verbreitete Metrik in Sachen Softwareentwicklung genauer erläutert.

Stufe 1 (initial) wird Unternehmen zugeteilt, die keinen festgelegten Entwicklungsprozess haben. Da der Prozess undurchsichtig ist, geraten Zeit und Kosten meist aus dem Ruder.

Stufe 2 (repeatable) macht durch die Dokumentation von Prozessen, was diese wiederholbar macht, ein effektives Projektmanagement möglich. Durch den Einsatz grundlegender Praktiken, die Kontrolle über Anforderungen, Zwischenprodukte und Ressourcen und die Möglichkeit des Managements, auf Probleme zu reagieren, können geplante Zeit und Kosten akzeptabel und vorhersehbar eingehalten werden.

Stufe 3 (defined) zielt auf den Softwareprozess und die gesamte Organisation ab. Alle Aktivitäten des Softwareprozesses sind detailliert festgelegt, können aber bei Bedarf z.T. angepasst werden. Außerdem sind sich Entwickler und Manager über ihre Verantwortlichkeiten bewusster, und das Management kann Risiken schon im Voraus einplanen. Insgesamt kommt dies der Transparenz des Projektstatus in hohem Maße zugute, Zeit und Kosten werden verringert.

Stufe 4 (managed) konzentriert sich auf Produkt- und Prozessqualität. Die Softwareprozessaktivitäten werden mit Hilfe von Metriken gemessen und kontrolliert. Mit diesen Messungen als Basis für Entscheidungen und der hohen Transparenz können Manager den Projektfortschritt präzise beeinflussen. Die Variation im Softwareprozess nimmt ab und Schätzungen werden genauer. Resultate sind zuverlässigere Vorhersagen, ein frühes Erkennen von Fehlern und eine weitere Senkung von Zeit und Kosten.

Stufe 5 (optimizing) fokussiert eine stetige Prozessverbesserung. Durch kontrolliertes Ausprobieren der Verbesserungen und Identifizieren und Austauschen von ineffizienten oder fehlerhaften Methoden können Manager Prozessverbesserungen klar abschätzen und systematisch durchführen. Auf diese Weise werden Zeit, Kosten und Varianz des Softwareprozesses fortwährend besser.

Hier lässt sich gut erkennen, auf welche Art die einzelnen Stufen einem Unternehmen Vorteile bringen, wenn sie diese erreichen. Einerseits werden natürlich Aspekte betrachtet, die sich auf den Softwareentwicklungsprozess beziehen. So werden z.B. grundlegende Softwaretechniken verlangt, wie etwa Peer Reviews, Software-Inspektionen und eine Konfigurationskontrolle [CMU93, Seite 48]. Schwachstellen im Softwareprozess werden so aufgedeckt und können behoben werden. Stufe 2 beispielsweise beschreibt die Minimalanforderungen an einen annehmbaren Softwareprozess, die ein Unternehmen erfüllen muss um diese Stufe zu erreichen. Die nächste Stufe fordert hingegen schon einen fest integrierten Softwareprozess [Kni04] und liefert auch Hinweise darauf, wie ein solcher zu erreichen ist. Größere Firmen erfüllen meistens zumindest die dritte Stufe.

Da für den Erfolg eines Projektes aber nicht nur die Softwaretechnik maßgeblich ist berücksichtigt CMM andererseits neben den technischen Aspekten des Softwareprozesses auch Management-Anforderungen, die bei CMM sogar die stärkere Aufmerksamkeit erfahren. Beispiele solcher Anforderungen sind etwa die benötigte Zeit zur Fertigstellung des Produktes oder die bei der Entwicklung entstehenden Kosten. Schon Stufe 2 setzt an diesem Punkt an und stellt sicher, dass Projekte überschaubarer und vorhersehbarer werden und sich so die Einhaltung geplanter Ziele verbessert. Mit jeder neu erreichten Stufe wird das Projekt noch transparenter, und **Zeit und Kosten** können immer besser kontrolliert und optimiert werden.

Michael Diaz und Joseph Sligo bestätigen dies in ihrer Fallstudie über den erfolgreichen Einsatz von CMM bei Motorola [DS97]. Folgende Grafik zeigt die Verbesserung der Zyklus-Dauer je CMM-Stufe. Der dargestellte x-Faktor errechnet sich aus der Zeit, die in einem vergleichbaren Projekt in der Vergangenheit für die Entwicklung eines Produkts gebraucht wurde, geteilt durch die Zeit, die in einem neuen Projekt bis zur Fertigstellung verging. Ein x-Faktor von 2 bedeutet also z.B. eine Halbierung der Entwicklungszeit.

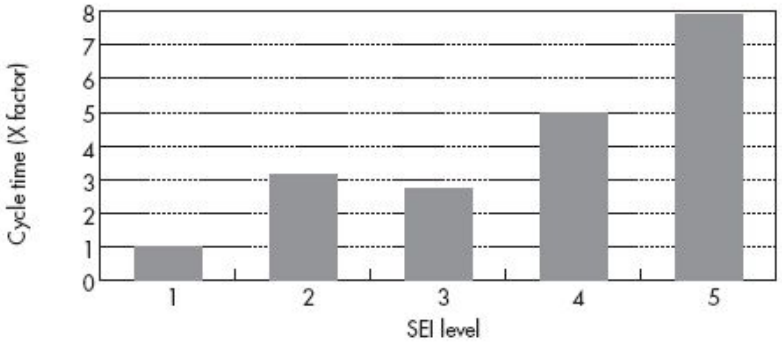


Abbildung 5. Verbesserungsfaktor der Zyklusdauer je Reifestufe [DS97]

Abgesehen von solch reinen „Management-Faktoren“ hat dies auch auf einen anderen, nicht minder wichtigen Faktor Auswirkungen: Die **Software-Qualität**. Ein verbesserter Softwareprozess (oder gar die bloße Einführung eines Prozesses überhaupt) mit neuen Qualitätssicherungstechniken (siehe z.B. die sehr effektiven Software-Inspektionen) führt ganz offensichtlich zu weniger Fehlern. Dies gilt nicht nur für den Code, sondern beginnt schon bei den Anforderungen an das Projekt, über den Entwurf bis hin zur Dokumentation. Explizite Software-Qualitäten die sich so aller Wahrscheinlichkeit nach verbessern lassen sind Korrektheit, Zuverlässigkeit, Robustheit, Effizienz, Benutzbarkeit, Wart-

barkeit, Wiederverwendbarkeit, Portierbarkeit, Verständlichkeit und Interoperabilität [Kni04]. Die so reduzierten Fehlerraten resultieren wiederum in einer kürzeren Entwicklungszeit (weniger Aufwand für Testen und Fehlerkorrektur vor der Freigabe des Produktes), geringeren Kosten sowie zufriedeneren Kunden durch kürzere Time-to-Market und weniger Fehlern in der freigegebenen Version.

Auch folgende Grafik entstammt der Diaz und Sligo Studie und illustriert das Sinken der Defektdichte mit zunehmender Reifegradstufe. MAELOC steht hier für „Million Assembly-Equivalent Lines of Code“. MAELOC berechnen sich aus den Anforderungen an das Produkt in Funktionspunkten multipliziert mit dem Expansionsfaktor von Capers Jones. Dieser von der Programmiersprache abhängige Faktor berechnet aus Funktionspunkten die erwarteten Code-Zeilen, die das Produkt bei Implementierung aller Funktionen umfasst. Bei der Grafik ist zu beachten, dass die y-Skala invertiert wurde:

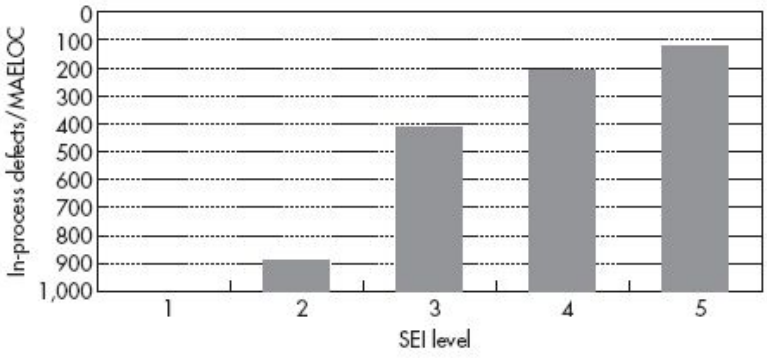


Abbildung 6. Verbesserung der Defektdichte je Reifestufe [DS97]

Die Optimierung von Zeit und Kosten und die nachhaltige Verbesserung der Produktqualität sind schon wesentliche Gründe für ein Unternehmen, auf CMM umzusteigen. Außerdem wurden die Stufen mit einer gewissen inneren Logik so entworfen, dass sie aufeinander aufbauen. Der Softwareprozess wird somit erst stabilisiert und dann Stück für Stück systematisch verbessert. Dies soll Unternehmen auf einem sinnvollen Weg zur Verbesserung führen. Schwachstellen können so leicht erkannt und behoben werden. Außerdem kann sich das Unternehmen bezüglich seines Softwareprozesses weitaus besser selbst einschätzen. Weiterhin ist zu vermuten, dass sich der reduzierte Stress durch fehlerfreiere Ergebnisse und weniger Zeitdruck positiv auf das Arbeitsklima auswirkt. Dies wurde bis jetzt jedoch noch in keiner dem Autor bekannten Studie untersucht. Eine hohe

Einstufung fördert zudem das Vertrauen bei Kunden und kann das Personal mit Stolz erfüllen und so den Teamgeist stärken.

3.2 Externer Nutzen

Lassen sich Unternehmen nach Reifegradmodellen einstufen, so hat dies auch viele Vorteile für den Kunden. Einer der wichtigsten Vorteile ist der damit gegebene offizielle **Qualitäts-Nachweis**. Der Kunde fühlt sich mit dem Unternehmen sicherer und kauft nicht die sprichwörtliche Katze im Sack. Dies geht sogar soweit, dass manche Kunden nicht-zertifizierte Lieferanten schon von vornherein ablehnen und nur noch auf offizielle Einstufungen in Reifegrade vertrauen. Ein weiterer erwähnenswerter Vorteil für Kunden ist außerdem, dass zertifizierte Firmen auf diese Art miteinander vergleichbarer werden.

Doch nicht nur die Einstufungen an sich stellen einen Nutzen für den Markt dar. Folgend werden zwei der genannten internen Vorteile nochmals aufgegriffen, um daraus externe abzuleiten.

Time-to-Market: Eine kürzere Dauer bis zur Freigabe des Produktes ist offensichtlich im Sinne des Kunden. Abgesehen davon, dass er sich nicht so lange gedulden muss, kann er durch einen früheren Einsatz des Produktes unter Umständen Aufwand und bares Geld sparen. Von der verkürzten Entwicklungsdauer abgesehen kann durch höhere Kontrolle des Softwareprozesses und geringere Varianz das Releasedatum präziser angegeben werden, was die Zuverlässigkeit des Unternehmens steigert.

Software-Qualität: Ein ganz klarer Vorteil für Kunden ist eine höhere Software-Qualität. Die Wahrscheinlichkeit für Fehler in der Freigabeversion wird mit steigender Stufe geringer. Weniger Fehler ersparen dem Benutzer Zeit und Frust. Die Benutzerfreundlichkeit und der Komfort können höher sein und es ist wahrscheinlich, dass das Programm korrekter und zuverlässiger arbeitet. All dies sind Punkte, die eine höhere Kundenzufriedenheit zur Folge haben.

4 Kritikpunkte

Trotz all der Hochschätzung und den Vorteilen die Reifegradmodelle bieten gibt es einige nicht zu verleugnende Punkte, die zu kritisieren sind [Pad07].

Ein zu beanstandender Punkt sind die durch sie verursachten Kosten. Dies beginnt schon bei der Einführung eines solchen Modelles in einem Unternehmen, welches meist seinen kompletten Entwicklungsprozess umstellen muss. Mitarbeiter müssen sich auf das neue System einstellen und auch (oder gerade) das Management hat viele neue Aufgaben zu erledigen. Diese Umstellung kostet außer Geld zunächst mal auch viel Zeit, die für aktuelle Entwicklungen verlorengeht, was wiederum Geld kostet. Der Nutzen dagegen wird meist erst nach einiger Zeit sichtbar. Ein weiterer Aufstieg zur nächsten Stufe ist meist langwierig und verursacht weitere Kosten für das Unternehmen, da die Assessments zur Feststellung einer neuen Stufe recht teuer sind, ob sie positiv ausfallen oder nicht. Aus diesem Grund haben viele Firmen lediglich eine inoffizielle Einstufung die

z.B. selbst durchgeführt sein kann. Außerdem versprechen Reifegradmodelle eine hohe Kontrolle über den Softwareprozess, wodurch er an feine Änderungen genau angepasst werden kann. Diese Kontrolle hat jedoch ebenfalls ihren Preis. Gerade in den höheren Stufen 4 und 5 sind hierfür intensive Messungen mit Hilfe von Metriken notwendig, deren Aufwand sich für die meisten Firmen gar nicht erst lohnt.

Auch was das Grundkonzept von Reifegradmodellen angeht gibt es einiges zu beanstanden. Ist die Umstellung in einem Unternehmen erstmal erfolgt, so schränkt dies dessen Flexibilität ein. Die Anforderungen an den Prozess werden früh und genau festgelegt. Innerhalb dieser Anforderungen ist freie Bewegung zwar erlaubt, wird jedoch eine spätere Abweichung nötig, führt dies unweigerlich zum Verlust der erreichten Stufe. So muss zum Erhalt der Stufe der Prozess beibehalten werden, auch wenn Mitarbeiter oder Manager für eine Änderung stimmen würden. Dieser Nachteil kommt oft auch schon bei der Umstellung selbst zum Tragen, wenn eine Organisation bewährte Methoden aufgeben muss, mit denen die Mitarbeiter schon viel Erfahrung gemacht haben. Leider ist gerade in den USA der Druck sich zertifizieren zu lassen recht hoch. Daher ist es vorstellbar, dass viele Unternehmen den Schritt trotz gemischter Gefühle wagen.

Außerdem wird oft nicht angegeben, wie eine Vorgabe genau durchzusetzen ist. Dies steht einer einfachen Prozessverbesserung im Wege. Gerade was softwaretechnische Belange angeht gibt es keine klare Prozessbeschreibung der Forderungen. Insgesamt ist CMM sehr auf Management-Aspekte ausgelegt und betont technische Aspekte dafür umso weniger.

Ein anderer Kritikpunkt ist die Anwendung der statistischen Prozesskontrolle, einem aus der industriellen Fertigung stammenden Konzept, auf die Softwareentwicklung. Wie der Name schon sagt wird Software aber nicht gefertigt, sondern entwickelt. Der entscheidende Unterschied liegt darin, dass in der Softwareentwicklung ein starkes Gewicht auf dem Entwurf liegt. Treten hier Fehler auf die nicht gleich beseitigt werden, ziehen sie sich durch das gesamte Projekt und werden desto teurer, je später sie entdeckt und behoben werden. In der Fertigung dagegen muss hierauf nicht geachtet werden. Noch heute wird darüber diskutiert, inwieweit sich also dieses Konzept auf dem die Reifegradmodelle beruhen sich auf die Softwareentwicklung übertragen lässt.

Was die Bewertung des Softwareprozesses angeht erfasst diese nur teilweise die Einflussfaktoren, die sich tatsächlich auf die Produktqualität auswirken. So werden beispielsweise für die Entwicklung eingesetzte Techniken wie Programmiersprachen oder benutzte Werkzeuge gar nicht erst berücksichtigt, genauso wenig die Erfahrung und Kompetenz der Entwickler. Der Fragenkatalog ist zwar sehr groß, um solch komplexe Vorgänge korrekt zu bewerten reicht er jedoch bei Weitem nicht aus. Weiterhin ist es nicht möglich Kriterien nur teilweise zu erfüllen, hier gibt es nur entweder ja oder nein. Sind nicht alle Kriterien erfüllt, so kann dies einen Unterschied von mehreren Stufen ausmachen, auch wenn es am Ende nur an Kleinigkeiten liegt. Insgesamt gibt es viele Firmen auf Stufe 1, die dennoch einen guten bis sehr guten Softwareprozess haben. Im Gegensatz

dazu gibt auch eine Stufe 5 keine Garantie, dass die fertigen Produkte von hoher Qualität sein werden.

Dem Autor ist keine Studie bekannt die belegt, dass eine hohe Prozessreife zwangsweise zu qualitativ hochwertigen Produkten führt. Wie die Studie von Diaz und Sligo [DS97] über Motorola zeigt, ist jedoch bei hoher Prozessreife die Wahrscheinlichkeit für bessere Produkte höher. Es gibt noch einige andere Fallstudien die über Erfolge mit CMM berichten, wie z.B. [HSW91] und [Dio93]. Doch ist dabei zu beachten, dass meist nur über positive Erfahrungen ausführlich berichtet wird.

Was die Akzeptanz von CMM bei Entwicklern angeht spricht dies auch nicht gerade für das Reifegradmodell. Es hat eher wenig Bezug zur eigentlichen Softwaretechnik und konzentriert sich dagegen vielmehr auf die Bürokratie. So werden häufige Probleme, mit denen Entwickler immer wieder konfrontiert werden, durch CMM nicht gelöst. Assessments lenken von der eigentlichen Arbeit ab und werden als störend empfunden. Deshalb kommt es häufig vor, dass Entwickler von dem Modell nicht gerade überzeugt sind. Dies ist auch eine Ursache dafür, dass vorgegebene Prozesse oft nicht eingehalten und sinngemäß umgesetzt werden.

5 Zusammenfassung

Reifegradmodelle stellen eine Methode dar, die Qualität der internen Prozesse und Strukturen eines Unternehmens und somit dessen Chance, dass Projekte in qualitativ hochwertigen Produkten resultieren, nach außen glaubwürdig sichtbar zu machen. Dies ist ein großer Vorteil sowohl für Kunden, als auch für die zertifizierten Unternehmen selbst. Außerdem bekommen die Unternehmen mehr Überblick und Kontrolle über ihre Prozesse, was eine effektive Prozessverbesserung ermöglicht. Allerdings gibt es noch immer zahlreiche Schwachstellen und Kritikpunkte, die trotz der mittlerweile schon fortgeschrittenen Erfahrung mit Reifegradmodellen noch nicht verbessert wurden. Speziell bei Modellen für Softwareprozesse werden vor allem gegen CMMI immer wieder Kritiken laut, welches in diesem Sektor aktuell das wohl am Meisten verbreitete Modell ist und zusammen mit CMM mittlerweile schon auf 16 Jahre Erfahrung in der Anwendung zurückgreifen kann. Ein Hauptkritikpunkt ist die starke Konzentration auf das Management und die damit einhergehende Vernachlässigung technischer Aspekte. Außerdem wird oft bemängelt, dass keine Vorgaben gemacht werden, wie geforderte Punkte für die nächste Stufe erreicht werden können. Die selben Kritikpunkte treffen z.B. auch auf SPICE zu.

Dennoch handelt es sich bei Reifegradmodellen um ein starkes Konzept, das viele Vorteile bietet. Neben den genannten Vorteilen der Prozessmodelle haben z.B. gerade Reifegradmodelle für Service-Orientierte Architekturen einen wesentlichen Beitrag dazu geleistet, das nicht eindeutig definierte und schwer zu durchschauende Konzept von SOAs besser zu verstehen und weiterzuentwickeln. Werden aktuelle Kritiken bei der Entwicklung zukünftiger Modelle berücksichtigt, so wird dies nach Ansicht des Autors dazu führen, dass Reifegradmodelle

aufgrund ihres wichtigen Beitrages zum Markt in Zukunft nicht mehr wegzudenken sein werden. Schon aktuelle Entwicklungen deuten dies an, da alleine die Anzahl der Bewertungen nach CMMI von Jahr zu Jahr zunimmt.

Literatur

CMU93. CMU/SEI. 93-TR-24: Capability Maturity Model® for Software (Version 1.1). Technical report, 1993.

CMU95. Software Engineering Institute Carnegie Mellon University. *The Capability Maturity Model: Guidelines for Improving the Software Process*. Addison-Wesley, 1995.

CMU08. CMU/SEI. Product Suite Basics. http://www.sei.cmu.edu/cmmi/faq/ps-faq.html, 2008.

CR07. Henning Groenda Christoph Rathfelder, editor. *Geschäftsprozessorientierte Kategorisierung von SOA*. FZI Karlsruhe, 2007.

Dio93. Raymond Dion. Process Improvement and the Corporate Balance Sheet. *IEEE Software*, 10(4):28–35, 1993.

DS97. Michael Diaz and Joseph Sligo. How Software Process Improvement Helped Motorola. *IEEE Software*, 14(5):75–81, 1997.

Gro06. The Open Group. Architecture Maturity Models. http://www.opengroup.org/architecture/togaf8-doc/arch/chap27.html, 2006.

Hay05. S. Hayward. Positions 2005: Service-Oriented Architecture Adds Flexibility to Business Processes. Gartner, Inc., Feb. 2005.

Hin04. Bernd Hindel. *Basiswissen Software-Projektmanagement*. dpunkt-Verl., 1. aufl. edition, 2004.

HSW91. Watts S. Humphrey, Terry R. Snyder, and Ronald R. Willis. Software Process Improvement at Hughes Aircraft. *IEEE Software*, 08(4):11–23, 1991.

Kne06. Ralf Kneuper. Links: Qualitätsmanagement, Qualitätssicherung und Vorgehensmodelle zur Softwareentwicklung. www.kneuper.de/Links/, 05 2006.

Kni04. Jan Knies. Grundlagen von Softwareprozessen und das Capability Maturity Model, 2004.

LL07. Jochen Ludewig and Horst Lichter. *Software-Engineering*. dpunkt-Verl., 1. aufl. edition, 2007.

oC03. Department of Commerce. Introduction - IT Architecture Capability Maturity Model, 05 2003.

Pad07. Frank Padberg. Vorlesung Software-Qualitätssicherung SS07, 2007.

Ser. Servicearchitecture. Bestimmung und Bedeutung des Reifegrads einer Service Orientierten Architektur. http://www.servicearchitecture.de/.

Son07. Progress Sonic. SOA Maturity Model. http://www.sonicsoftware.com/soamm, 2007.

Spr07. Markus Sprunck. ISO/IEC 15504 SPICE. SE 2007, 03 2007. http://www.se2007.de/.

ISO/IEC 15504 (SPICE)

Thomas Weinstein

Betreuer: Christoph Rathfelder

Zusammenfassung Dieses Paper wird einen Überblick über den Prozessbewertungsstandard ISO 15504, insbesondere die Prozessassessmentmethode bekannt als SPICE, geben und diesen insbesondere gegen die konkurrierende Methode CMMI abgrenzen. Daneben soll kurz auf die Entstehungsgeschichte und die Beziehungen zu den Standards ISO 9000 und ISO 12207 eingegangen werden. Die praktische Relevanz dieser Qualitätssicherungsmethode soll anhand von einigen Beispielen erörtert werden, wobei auch auf eventuelle Problematiken eingegangen wird. Abschließend wird versucht die Erkenntnisse kurz zusammenzufassen, den Standard insbesondere im Hinblick auf die Vielzahl konkurrierender Methoden zu reflektieren und einen kurzen Ausblick in die weitere Entwicklungen in diesem Gebiet zu geben.

1 Motivation

Man stelle sich vor, man muss bei einem Arztbesuch ein Körperteil röntgen lassen und erhält stattdessen eine Krebstherapie-Bestrahlung. Oder man erwartet eine solche und erhält stattdessen eine Röntgendosis in 100fach erhöhter Dosis und trägt schwere Strahlenverbrennungen davon, teilweise mit Todesfolge. Geschehen in den USA und Kanada Mitte der 80er Jahre, bei dem Multifunktionsgerät THERAC-25 ([Lev95]). Grund dieser Ausfälle, die teilweise Menschleben kosteten, war fehlerhafte Software, wobei ganz klar das Qualitätsmanagement der produzierenden Firma versagt hatte. Weiter Beispiele für kostspielige Software-Schäden sind die verantwortungslose Software-Verwendung bei der Ariane 5 Rakete, die den Steuerzahler 500 Millionen US-Dollar kostete ([LL96]), oder die mangelhafte Anforderungsanalyse beim Airbus A-320, die die Schubumkehr nur zuließ, wenn beide Rädersensoren Bodenkontakt meldeten ([Ste00]).

Man sieht, dass es sich durchaus lohnt, sich über Qualität Gedanken zu machen, da Software immer weiter in tägliche Dinge unseres Lebens eindringt und so die Verantwortung der Hersteller steigt. In der IT-Branche herrscht darüberhinaus eine sehr starke Konkurrenz, so dass es von höchster Wichtigkeit ist, Budget- und Zeitvorgaben möglichst genau einhalten zu können - oftmals leidet gerade deswegen die Qualität. Verzögerte Auslieferungen von Produkten und erhöhte Time-to-Market-Werte ziehen oftmals verringerte Marktanteile und damit verlorene Gewinne nach sich. Wenn man sich aber nun geeinigt hat, dass qualitativ hochwertige Software wichtig ist, werfen sich sofort weitere Fragen auf: Wie definiert man Qualität bei Software? Wenige Defekte, möglichst gute Wartbarkeit und Testbarkeit oder einfache Benutzbarkeit sind sicher gute Ansätze und jeweils verschiedene Seiten derselben Medaille.

Aber wie erreicht man eine konstant hohe Qualität seiner Produkte? Eine mögliche Ansicht ist die, dass ein definierter, stabiler Entwicklungsprozess dabei hilft konstant gute Produkte abzuliefern. Ansatzpunkt ist hierbei, dass die teilweise extremen Schwankungen in Zeit- bzw. Budgetverbrauch durch ein wiederholbares, definiertes Vorgehen verringert werden sollen. Ob dies empirisch belegt werden kann, oder ob Prozessdefinition und -bewertung nur zur Folge haben, dass das Unternehmen standardisiert und wiederholbar unbrauchbare Ergebnisse produziert, wird in den Praxisbeispielen näher erläutert. Doch alleine schon die Definition eines gelebten Entwicklungsprozesses ist keine triviale Aufgabe, geschweige denn die Bewertung des Status Quo und eventuelle Verbesserungsmaßnahmen. Genau dafür existieren die verschiedensten Reifegradmodelle, die versuchen diese Bewertung konsistent, vergleichbar und wiederholbar durchzuführen. Doch zunächst ein kleiner Überblick über die verschiedenen Modelle, die aktuell für den IT-Markt relevant sind.

2 Einordnung in Qualitätsmodelle

2.1 Überblick

Ein Reifegradmodell versucht die Reife eines Unternehmens oder einzelner Prozesse festzustellen, wobei diese je nach Modell auf unterschiedlichen Skalen gemessen wird. Wenn man die relevanten Reifegradmodelle aufzählen möchte, sind die zwei wichtigsten CMMI - das Capability Maturity Model Integration - und SPICE - Software Process Improvement and Capability Determination. Es gibt noch eine Reihe weiterer, auszugsweise aufgeführt etwa in [Wal07], doch sind diese entweder nur branchenspezifisch (Automotive SPICE), bereits veraltet oder durch Nachfolgemodelle ersetzt (CMM, Bootstrap, Trillium) oder nur regional von Bedeutung (MITO). Durchaus gebräuchlich waren auch lange Zeit Zertifizierungen nach dem ISO 9000 Standard, doch in wiefern ISO 9000, 12207 und 15504 aufeinander aufbauen soll genauer in einem eigenen Kapitel behandelt werden.

MITO - Maturity Model for IT Operations - ist ein Standard der von der SAQ - Swiss Association for Quality Promotion - in den Jahren 2001/2002 verabschiedet wurde, und zunächst eine Prozessbewertung vornimmt und dann daraus nach festgelegten Regeln einen Reifegrad für das Unternehmen bestimmt, wie in [Sch00] beschrieben. Da sich dieser Standard aber gegen den wachsenden Erfolg von CMM und ISO 15504 konformen Assessments (engl.: Bewertungen) nicht durchsetzen konnte hat er heutzutage kaum bis keine Praxisrelevanz. Bootstrap und Trillium waren Abwandlungen von CMM, die von der EU bzw. von Bell Canada entwickelt worden waren, die jedoch heute auch kaum noch von Bedeutung sind. Automotive SPICE ist ein alternatives Prozessassessment-Modell zum originären SPICE, das speziell auf die Automobilindustrie zugeschnitten ist, und dort sehr verbreitet ist, wie die Mitgliederliste der Automotive SIG - Special Interest Group - [Gro05] zeigt (Audi, BMW, Daimler AG, Fiat, Ford u.a.). Dieses Modell ist konform zur ISO 15504 Norm, wobei auf den Unterschied zwischen SPICE und ISO 15504 in der gesonderten Betrachtung noch näher eingegangen

wird. Für die marktrelevanten Standards SPICE und CMMI soll nun ein kurzer Abriss über die historische Entwicklung folgen.

2.2 Entwicklung und Verabschiedung der relevanten Standards

CMM, das Capability Maturity Model, wurde vom SEI (Software Engineering Institute) der Carnegie Mellon University 1991 veröffentlicht. Dies geschah auf Drängen des US-Verteidigungsministeriums, da laut einer Studie von 1989 nur 24% der Software ohne größere Einschränkung nutzbar war. (Daten aus [Hör06]) Das Veröffentlichungsjahr von CMM ist nur deswegen interessant, da es weit vor dem des ISO 15504 Standards liegt und daher ein Grund für den äußerst hohen Marktanteil und die weitgehende Akzeptanz in allen Branchen darstellt. Die erste Version der weiterentwickelten Version CMMI wurde im Jahre 2000 veröffentlicht. Folgende Entwicklungsgeschichte (Daten auszugsweise aus [Ema98] und [Loo07]) beschriebt den Werdegang des ISO 15504 Standards. Er fand seinen Ursprung auch in einer Studie, nämlich "Improve-IT", eine Studie im Auftrag des britischen Verteidigungsministeriums im Jahre 1991, die dazu dienen sollte einen Überblick über die gebräuchlichen Assessmentmodelle zu erhalten. Sie umfasste schließlich CMM, Bootstrap, Trillium und viele weitere Verfahren die von einzelnen Firmen für den internen Gebrauch entwickelt worden waren. Die Ergebnisse dieser Studie gingen an die ISO, die daraus einen Standard entwickeln sollte. Die zuständige Unterabteilung war das Joint Technical Committee 1, zuständig für Information Technolgy, mit seinem Subcommittee 7 für Software Engineering. Speziell für Softwareprozessassessments gab es darin wiederum die Unterabteilung Working Group 10, die eigentlich die Entwicklung beginnen sollte. Diese startete jedoch zunächst ein Projekt mit dem Namen "SPICE", um einerseits die Länder die nicht in der ISO vertreten waren nicht von vornherein bei der Konkretisierung des Standards auszuschließen (nur offizielle Vertreter der ISO-Mitgliedsländer wären sonst zur Mitarbeit berechtigt gewesen) und andererseits sehr früh praxisrelevantes Feedback zurückzubekommen, da die langsame Bürokratie der Veröffentlichung eines offiziellen Technical Reports der ISO umgangen wurde. In dem von dieser Projektgruppe entwickelten Standard war noch ein vollkommen eigenständiges Prozessreferenzmodell enthalten, später wurde dieses durch ein angepasstes Modell nach dem ISO 12207 Standard ausgetauscht, doch mehr dazu in den entsprechenden Kapiteln. Die erste Version erschien 1998, die aktuellen Teile 1 bis 5 des ISO 15504 Standards sukzessive von 2003 bis 2006. Dieser Standard ist wie CMMI ständig in der Entwicklung und weitere Teile sind bereits in Arbeit bzw. werden sehr bald veröffentlicht. Doch nun zur detaillierten Betrachtung der entsprechenden ISO Normen.

3 Detaillierte Beschreibung

3.1 Verknüpfungen

Dieser Abschnitt soll kurz zur Verdeutlichung der Zusammenhänge zwischen den drei ISO Standards dienen. ISO 9000:2000 und 9001:2000 stellen so etwas wie

eine Basis, einen Sockel dar, das absolute Minimum, das Qualitätsmanagementsysteme, egal welcher Art und Ausprägung, egal in welcher Branche, erfüllen müssen. Laut [Sti99] kann mit einer Erfüllung dieser Forderung im Allgemeinen in etwa bei SPICE-Stufe 2 gerechnet werden, spätestens wenn die relevanten Prozesse den Reifegrad 3 erreicht haben. Dennoch muss dies explizit in getrennten Assessments festgestellt werden, da die Normen zwar aufeinander aufbauen, aber eine direkte Zuordnung dennoch nicht möglich ist. ISO 12207 stellt die Basis für das Prozessreferenzmodell von SPICE dar, wurde erweitert und verbessert und fand so in leicht abgewandelter Form Eingang in ISO 15504 - Teil 5. Somit verwendet SPICE grundsätzlich gesehen die selbe Prozessstruktur. ISO 12207 selbst verwendet teils Vokabular das in ISO 9000:2000 definiert wurde. ISO 15504 besteht aus fünf Teilen von denen zwei besondere Bedeutung haben: Teil 2 ist der normative Teil des Standards, Teil 5 beinhaltet ein beispielhaftes PRM und PAM. Die entsprechenden Verbindungen zu den obigen Normen sind bereits ausführlich aufgezählt worden. Visualisiert wird dies in Abbildung 1 dargestellt.

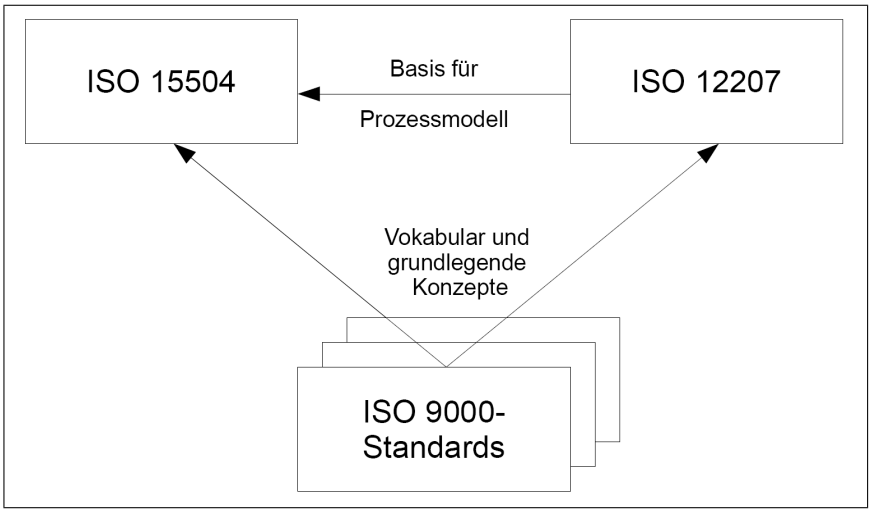


Abbildung 1. Verknüpfungen zwischen den Standards

3.2 ISO 9000

Die ISO 9000 Normen sind eine Familie von Standards die allgemeine Regeln und Bewertungsmaßstäbe für Qualitätsmanagementsysteme vorgeben. Diese beziehen sich jedoch nicht alleine auf Software oder IT, sondern sind allgemeine Regeln für beliebige Branchen. ISO 9000:2005 (also die aktuelle Fassung der Norm, verabschiedet im Jahre 2005) beinhaltet elementare Grundlagen wie Begriffsdefinitionen, die für alle anderen Normen der 9000-Familie aber auch für

12207 und 15504 verwendet werden, hat aber selbst kaum bis keinen normativen Charakter, da keine Forderungen zu seiner Erfüllung gestellt werden. ISO 9001:2000, nach der Zertifikate ausgestellt werden können, fordert sechs wichtige Eigenschaften für ein Qualitätsmanagement-System, genannt etwa in [Sch94]. Um diese Norm zu erfüllen und die entsprechenden Zertifikate zu erhalten müssen dokumentierte Verfahren nachgewiesen werden für folgende Maßnahmen:

1. Kontrolle von Dokumenten
2. Kontrolle von Qualitätsaufzeichnungen
3. Interne Audits
4. Kontrolle fehlerhafter Produkte
5. Korrekturmaßnahmen
6. Vorbeugemaßnahmen

Die aktuelle Version dieses Standards führt die ehemaligen Normen DIN EN ISO 9001 bis 9003 in sehr kompakter Form zusammen, so dass im Prinzip diese eine Norm als einzige wirkliche normative Relevanz besitzt. ISO 9004:2000 beinhaltet ein Leitfaden zur Leistungs- und Kundenzufriedenheitssteigerung, von dem aber durch Angabe guter Gründe abgewichen werden kann. Diese Norm ist laut [Sch94] in der Praxis nur als zusätzlicher Kommentar zu verstehen, der bei ISO 9000 Zertifzierungen allerdings kaum bis keine praktische Relevanz zuzuschreiben ist. Zu beachten ist noch, dass durch die Umstrukturierung der Standards zwar umgangssprachlich Firmen ISO 9000 zertifiziert sind, jedoch dies im weiteren Sinne für die ISO 9000 Familie steht, da nur ISO 9001 normative Wirkung hat. Für weitere Details wird auf [fS05], [fS00a] und [fS00b] verwiesen.

3.3 ISO 12207

ISO 12207 ([fS02], oder beschrieben in [Sin96] und [Wal07]) definiert ein Prozessreferenzmodell und beschreibt die Eigenschaften der einzelnen spezifischen Prozesse, die bei der Softwareentwicklung verwendet werden. Diese werden zunächst in drei Prozesskategorien eingeteilt:

- Primäre Prozesse (Sie stellen die notwendigen Voraussetzungen für eine funktionierende Softwareentwicklung dar)
- Unterstützungsprozesse (Sie stellen nützliche Funktionalitäten dar oder verbessern die Produktivität, sind aber prinzipiell entbehrlich)
- Organisatorische Prozesse (Sie dienen der Verwaltung und dem allgemeinen Prozessmanagement)

Die primären Prozesse nach ISO 12207 sind:

- Akquisition
- Lieferung
- Entwicklung
- Betrieb
- Wartung

Die Unterstützungsprozesse beinhalten:

- Dokumentation
- Konfigurationsmanagement
- Qualitätssicherung
- Verifikation
- Validation
- Gemeinsames Review
- Audit
- Problemlösung

Die Organisatorischen Prozesse sind:

- Management
- Infrastruktur
- Verbesserung
- Training

Eine festgelegte Entwicklungsmethodik, Dokumente oder exakt definierte Abläufe der Prozesse werden in dieser Norm allerdings nicht gefordert. Für eine genauere Betrachtung muss auf [Sin96] oder gar die Lektüre der Norm verwiesen werden, hier soll nun nur noch kurz der Aufbau eines einzelnen Prozesses beschrieben werden. Für jeden Prozess werden Aktivitäten und Aufgaben definiert, die ihn eindeutig bestimmen. Dabei sind diese wie in folgender Grafik (entnommen aus [Sin96]) angedeutet untereinander verknüpft, also einem Prozess sind direkt die Aktivitäten zugeordnet, welchen dann die notwendigen Aufgaben zugeordnet sind.

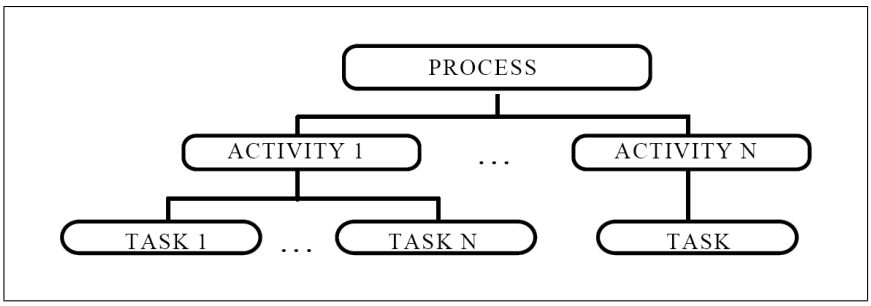


Abbildung 2. ISO 12207 - Struktur

3.4 ISO 15504

ISO 15504 ([fS04], beschrieben in [Wal07] und [Loo07])besteht im Moment aus fünf separaten Teilen, die folgenden Inhalt haben:

- Teil 1: Allgemeine Konzepte und Begriffsdefinitionen
- Teil 2: Forderungen an die Durchführung eines Assessments
- Teil 3: Leitfaden für die Durchführung eines Assessments
- Teil 4: Leitfaden zur Nutzung bei Prozessverbesserung und Prozessbewertung
- Teil 5: ein beispielhaftes Prozessassementmodell (basierend auf den ISO 12207 Prozesssdefinitionen)

Abschnitt 2 ist der wichtige normative Teil. Ein Assessment, das ISO 15504 konform ablaufen soll, muss folgende Teilaspekte mindestens erfüllen:

Der Assessment Prozess Dieser muss folgende fünf Vorgehensweisen spezifizieren:

- Planung: Der zeitliche Ablauf muss fix sein, benötigte Inputs müssen von vornherein klar sein, ebenso benötigte Ressourcen und welche Rollen zwingend vergeben werden müssen.
- Sammlung: Für die Datensammlung müssen nachvollziehbare, rationale, wiederholbare Strategien angegeben werden und die Prozesse müssen bereits grob den Entsprechungen im Prozessreferenzmodell zugeordnet werden. Vorgehensweisen für die Sichtung und korrekte Archivierung der gesammelten Daten müssen ebenfalls angegeben werden.
- Validierung: Es muss eine Möglichkeit angegeben werden, die Daten konsistent, objektiv und fair auf ihre Korrektheit und Wahrheit hin zu überprüfen
- Bewertung: Die gesammelten Nachweise müssen zur Profilbildung den entsprechenden Indikatoren für die im Bewertungsrahmen fixierten Prozessattribute zugewiesen werden.
- Reports: Es muss klar sein, in welchem Maße und wie die Nachweise und das erreichte Profil als Ergebnis dokumentiert werden.

Rollen Folgende Rollen müssen mindestens definiert und Personen zugewiesen werden:

- Sponsor: Dieser muss Zugang zu den benötigten Ressourcen gewähren und die benötigten Mittel an Geld und Manpower zur Verfügung stellen.
- Assessment-Leiter: Dieser ist verantwortlich für die Qualität des Assessments, das hinreichende Fachwissen seiner Assessoren und die standardkonforme Durchführung der Bewertung.
- Assessoren: Bis auf eine hinreichende Kenntnis des Assessments sind keine besonderen Anforderungen zu stellen.

Inputs / Outputs Als Mindestangaben sind wiederum vorgegeben:

- Zweck und Umfang der Bewertung müssen als Input klar sein, gegebenenfalls vorhandene Restriktionen bezüglich Dauer, Ressourcenzugang und Geheimhaltung, sowie spezielle Kompetenz-Anforderungen an die Assessoren oder benötigte Zusatzinformationen sind hier aufzuführen.

– Als gewünschte Outputs sind mindestens das Datum, sämtliche Inputs und gefundene Nachweise anzuführen. Darüber hinaus noch die verwendeten Modelle, das Ergebnis des Assessments, also der erreichte Reifegrad aller bewerteten Prozesse und ggf. Zusatzinformationen.

Bewertungsrahmen Dieser ist fix vorgegeben und enthält drei Teile:

– **Reifegradstufen**: Die sechs definierten Reifegrade sind:
  • 0 *Incomplete*: Der Prozess ist nicht implementiert oder erreicht nicht systematisch seinen Zweck.
  • 1 *Performed*: Er erreicht zumindest seinen Zweck.
  • 2 *Managed*: Der Prozess ist verwaltet, es werden also Planung, Aufzeichnung und Anpassung vollzogen und die Zwischenergebnisse sind etabliert und kontrolliert.
  • 3 *Established*: Es wird ein standardisierter Prozess verwendet, der nur noch flexibel angepasst wird.
  • 4 *Predictable*: Der Prozess verläuft innerhalb definierter Schranken, quantitative Metriken werden zur Kontrolle erhoben.
  • 5 *Optimizing*: Kontinuierliche Prozessverbesserung durch Innovation und Optimierung.

– **Prozessattribute**: Jede Reifegradstufe hat definierte Prozessattribute, die zu einem gewissen Grad erfüllt sein müssen, um den jeweiligen Reifegrad zu erreichen. Diese sind jedoch nur namentlich definiert und werden erst in den entsprechenden Implementierungen des Standards genauer spezifiziert.

– **Bewertungsskala**: Als Skala ist eine vierwertige Ordinalskala vorgegeben, wobei Prozentwerte angegeben sind, die die einzelnen Stufen definieren.
  • *N[ot]*: kleiner 15%
  • *P[artially]*: kleiner 50%
  • *L[argely]*: kleiner 85%
  • *F[ully]*: kleiner 100%
  • um Reifegrad X zu erreichen müssen alle Prozessattribute der Grade 1 bis X-1 voll (F) und die des Grades X größtenteils (L) erfüllt sein.

Die folgende Teilaspekte sind variabel und können selbst definiert oder aus einem vorgegebenen Prozessassessmentmodel (wie in Teil 5) übernommen werden:

Prozessreferenzmodell - PRM An diesen Teil werden nur die Anforderungen gestellt, den Anwendungsbereich und den Umfang der einzelnen Prozesse, der Zweck und die erwarteten Outputs zu spezifizieren. Mögliche Alternativen gibt es viele, hier seien nur die wichtigsten nach Ansicht von [Loo07] aufgeführt, abgesehen von dem später noch genauer beschriebenen:

– direkte Anwendung des ISO 12207: Information Technology - Software Lifecycle Processes

- direkte Anwendung des ISO 15288: Systems Engineering - Systems Life Cycle Processes
- ein Prozessmodell speziell für Objektorientierte Entwicklung aus OOSpice
- das in der Automobilindustrie weit verbreitete PRM von Automotive Spice

Prozessassessmentmodell - PAM Dieser Teil ist dafür zuständig, die Indikatoren des vorhandenen, gelebten Prozesses den Prozessattributen zuzuordnen und stellt damit das Bindglied zwischen PRM und Bewertungsrahmen dar. Die Bewertung ist bisher nur sehr generisch vorgegeben und wird erst durch das verwendete PAM spezieller. Dies ermöglicht es, den ISO 15504 auf eine Vielzahl von Anwendungsdomänen anzupassen, und macht ihn dadurch sehr flexibel. Zwar werden PAMs weitestgehend zusammen mit passenden PRMs verabschiedet, jedoch sind beide theoretisch getrennt entwickelbar, austauschbar und aufeinander abstimmbar. Eine beispielhafte Aufzählung verschiedener ISO 15504 konformen Modelle frei nach [Loo07], wiederum exklusive SPICE:

- OOSpice - erweitert SPICE insbesondere in den Bereichen Modellierung, Spezifikationen und Modulzusammenführung.
- Automotive Spice - entwickelt von der Herstellerinitiative Software (HIS), ein Zusammenschluss der großen deutschen Automobilhersteller Audi, BMW, Porsche, Daimler AG und VW
- CMMI - das Capability Maturity Model Integration des SEI ist weitestgehend ISO 15504 konform

Die Abschnitte 3 und 4 stellen nur Leitfäden dar, von denen abgewichen werden kann, so dass nun direkt auf Abschnitt 5 eingegangen werden soll. Dieser Teil des Standards definiert beispielhaft ein ISO 15504 konformes Prozessreferenzmodell und ein Prozessassessmentmodell. Das PRM basiert auf ISO 12207 und hat zahlreiche Prozesse übernommen, einige aufgesplittet und wieder andere hinzugefügt. Das PAM (zusammen mit dem PRM) ist bekannt unter dem Namen SPICE, der von dem Projekt übernommen wurde, das den Standard in seiner ursprünglichen Fassung verabschiedet hatte. Um also noch einmal die Zusammenhänge klarzustellen, ISO 15504 ist der generische Standard, der obige Forderungen an eine Bewertung oder Zertifizierung stellt, und SPICE ist eine beispielhafte Ausprägung, nach der Assessments durchgeführt werden können.

Das SPICE-PRM Hier werden unter analoger Anwendung des ISO 12207 drei Prozesskategorien definiert die jeweils bis zu vier Prozessgruppen enthalten, in denen wiederum bis zu zwölf Einzelprozesse angegeben sind. Die Einzelprozesse würden den Rahmen sprengen, aber die Prozessgruppen sollen hier aufgeführt werden, um einen Überblick über die Prozessdimension von SPICE zu erhalten:

Primäre Prozesse

- ACQ - Acquisition

- SPL - Supply
- ENG - Engineering
- OPE - Operation

Unterstützende Prozesse

- SUP - Support

Organisatorische Prozesse

- MAN - Management
- PIM - Process Improvement
- RIN - Ressource and Infrastructure
- REU - Reuse

Jeder Einzelprozess definiert den Zweck, gewünschte Outputs, geforderte Basispraktiken, die sich als notwendig erweisen, um sowohl Zweck als auch Output zu erreichen. Beispielhaft sei dies an einem Einzelprozess dargestellt, an ENG.1 Anforderungserhebung (wörtliche Übersetzung nach Hörmann, Dittmann)

- Zweck: Zweck des Prozesses ist es, entstehende Kundenbedürfnisse und -anforderungen während der gesamten Lebensdauer eines Produkts bzw. einer Dienstleistung zu erfassen, zu verarbeiten und zu verfolgen, um dadurch eine Anforderungsbaseline zu bilden, die als Basis für die Definition der benötigten Arbeitsprodukte dient. Anforderungsanalyse kann durch den Käufer oder den Entwickler des Systems durchgeführt werden.
- Basispraktik 1: Kundenforderungen und -wünsche einholen
- Basispraktik 2: Kundenerwartungen verstehen
- Basispraktik 3: Übereinkunft bezüglich der Anforderungen erzielen
- Basispraktik 4: Baseline der Kundenanforderungen aufstellen
- Basispraktik 5: Änderungen der Kundenanforderungen managen
- Basispraktik 6: Kundenanfrageverfahren aufbauen
- Output 1: Dokument zur Änderungsaufzeichnung
- Output 2: Dokument für Kundenanforderung

Das SPICE-PAM Die Verbindung zwischen Bewertungsrahmen und Prozessdimension findet in diesem Modell über die so genannten Prozessattribute aus Abschnitt 2 statt. Jeder Reifegradstufe sind bis zu zwei solcher Attribute zugeordnet und jedem Attribut sind eine Vielzahl von Generischen Praktiken, Generischen Ressourcen und Generischen Outputs zugeordnet, die schließlich vom Assessor nachgewiesenermaßen in der realen Ausprägung gefunden werden müssen. Zunächst ein Überblick über die Prozessatrribute:

- Stufe 0: keine
- Stufe 1: Prozessdurchführung
- Stufe 2: Management der Prozessdurchführung, Management der Arbeitsprodukte
- Stufe 3: Prozessdefinition, Prozessanwendung

- Stufe 4: Prozessmessung, Prozesssteuerung
- Stufe 5: Prozessinnovation, Prozessoptimierung

Das erste Prozessattribut, PA 1.1 Prozessdurchführung, nimmt hierbei eine Sonderrolle ein. Dieses Attribut verweist durch nur eine Generische Praktik direkt auf das verwendete Prozessreferenzmodell und fordert nur, dass die dort definierten Eigenschaften erfüllt sind, also die definierten Outputs und Basispraktiken nachweisbar sind. Alle anderen Attribute erfordern die Implementierung verschiedenster Generischer Elemente, die aufgrund ihrer Flexibilität aber weder auf konkrete Technologien noch auf bestimmte Prozessausprägungen Bezug nehmen. Beipsiel für eine Generische Praktik ist z.B. GP 2.1.2 "Planung und Überwachung der Performance des Prozesses um die identifizierten Ziele zu erreichen", in der Meilensteine, einfache Schätzungen und Planungen gefordert werden. Sie ist ein Teilaspekt des Prozessattributs PA 2.1 "Management der Prozessdurchführung". Für eine erschöpfende Auflistung wird auf [Hör06] oder [Loo07] verwiesen.

4 Praktische Bedeutung

4.1 Nutzen oder Nichtnutzen von Prozessbewertung

Es ist allgemein anerkannter Standard, dass Prozessverbesserung im Normalfall einhergeht mit Einsparungen in den Entwicklungskosten, einer Steigerung der Qualität und einer verbesserten Planungssicherheit. Als Beleg hierfür kann die gesamte zitierte Literatur herangezogen werdem, die nicht in einem Abschnitt dieser Arbeit mit eher kritischer Sichtweise verwendet wurde. Alle Autoren gehen von dem kausalen Zusammenhang "Aus Prozessverbesserung folgt Produktverbesserung" aus. Der Versuch das Risiko zu kontrollieren um so Budget-, Zeit- und Qualitätslimits einhalten zu können ist grundsätzlich auch nicht verkehrt, jedoch müssen sich die immensen Kosten einer Prozessverbesserung erst einmal bezahlt machen oder die verbesserten Aktivitäten vom Personal überhaupt eingesetzt werden. Dass dies nicht immer so ist, soll folgendes Beispiel verdeutlichen: In [DKW99] wird einer der wenigen veröffentlichten Fälle beschrieben in denen eine gut gemeinte Prozessverbesserung nicht zum Erfolg führte.

Bei ACME Stores, einem Homeshopping-Unternehmen, das eine IT-Abteilung von mehr als 60 Mitarbeitern besaß, traten in den frühen 90ern vermehrt Probleme mit zu langen Zykluszeiten, zu viel Arbeitsaufwand für Instandhaltung und Support und das allgemeine Fehlen eines Standardprozesses auf. Eine externe Beratungsfirma führte daher eine Methodik mit formaler Definition der Prozesse ein, die allgemein als hilfreich anerkannt war. Die Folge war, dass die Mitarbeiter jedoch mehr und mehr ihren Fokus auf die korrekte Notation der geforderten Dokumente statt auf den Inhalt der Prozessergebnisse legten. Der Zweck der einzelnen Prozessschritte trat in den Hintergrund und wich einer mechanischen Abarbeitung um die eigene Produktivität zu zeigen. Den Kunden wurde nach der Anforderungsanalyse komplizierte Diagramme zur Validierung vorgelegt, die sie kaum verstanden, was diese nur verärgerte. Nach Beendigung des Pilotprojekts

wurden noch zwei ähnliche Projekte mit abgeschwächter Methodik versucht, die aber genauso scheiterten und die Änderungen komplett zurückgezogen wurden. Man sieht, dass ein definierter Prozess und Fixierung auf die Prozessverbesserung kein Allheilmittel darstellen, und wenn sie zum Selbstzweck verfolgt werden sogar schädlich sein können. Dazu passt ein Zitat vom Technical Director von Columbine, einem 200 Mitarbeiter großen Softwarehaus: "Quality is much more than having a certificate on the wall" ([DKW99]).

4.2 Validität der Bewertung

El Emam ([Ema98]) versuchte anhand einer empirischen Untersuchung durch die Befragung der Firmen, welche am ersten SPICE Trial teilgenommen hatten, die Aussagekraft der Spicebewertung zu validieren. Diese Trials wurden während des SPICE Projekts mit der geplanten Version 1.00 durchgeführt, um möglichst schnell Praxiserfahrung mit dem neuen Standard sammeln zu können. Zwar unterscheidet sich diese Version von der heutig gebräuchlichen, jedoch sind die Ansätze noch die selben, weshalb diese Daten nichts von ihrer Aussagekraft verloren haben. Auf die Frage ob das Assessment seine Kosten wert war, antworteten 40% der Firmenvertreter "more than worth it", wenn man alle positiven und neutralen Antworten der Skala dazu nimmt, waren es 97%. Laut 80% der Antworten war das Assessment sehr gut dazu geeignet, um das höhere Management auf das Thema Softwareprozessverbesserung zu sensibilisieren und deren Engagement in diese Richtung zu verbessern. Die selbe Frage für die Technischen Mitarbeiter wurde dagegen nur von 65% positiv beantwortet. Ob die SPICE-Bewertung die Stärken und Schwächen gut erkannt hatte und wertvolle Hinweise für die Verbesserung gegeben hatte wurde durchweg mit über 90% Zustimmung beantwortet. Die Fragebögen wurden auch von den Assessoren ausgefüllt, um zu bewerten wie sie aus Sicht der Firmen antworten würden, wobei hier zu 80-90% sehr positives Feedback zurückkam, was lediglich eine starke Überzeugung der Befragten von der Richtigkeit ihrer Arbeit zum Ausdruck bringt. An sich unterstützen diese Daten also sehr gut die Aussagekraft von SPICE-Assessments, jedoch müssen sowohl interne als auch externe Validität der Umfrage in Frage gestellt werden.

Einerseits müssen die (eigentlich aus der Theorie der kontrollierten Experimente bekannten) Hawthorne-, Neuigkeits- und Subjekteffekte beachtet werden, nach denen Personen die wissen, dass sie an einer wissenschaftlichen Arbeit beteiligt sind, sich anders als im Alltag verhalten, neue Tools, Methoden und Theorien bei den ersten Tests allgemein besser bewertet werden als in nachfolgenden und Personen sich oftmals so verhalten wie sie denken, dass es von ihnen erwartet wird. Darüber hinaus muss man in Betracht ziehen, dass an den ersten SPICE Trials nur Firmen teilnehmen durften, die bereits stark in das Projekt involviert waren, also in dieser Umfrage Firmenvertreter im weitesten Sinne die Arbeit ihrer eigenen Kollegen bewerten mussten, was letzten Endes doch sehr an der Aussagekraft dieser Befragungen zweifeln lässt. Jedoch konnten die Ergebnisse in einem zweiten Durchgang wiederholt werden, dieses mal in den zweiten SPICE Trials, bei denen auch unabhängige Firmen teilnehmen durften. Diese Daten

aus [EE98] bestärken die These, dass die interne Validität gegeben ist und sich insbesondere von Version 1.00 zu Version 2.00 nicht verschlechtert hat.

4.3 Konsistenz der Bewertungen

Ein klein angelegtes Experiment ([EEBS96]), wiederum unter der Leitung von Khaled El Emam, sollte nachweisen, dass die SPICE-Bewertungen konsistent und objektiv seien, also zufällige Schwankungen der Reifegrade aufgrund anderer Assessoren zu vernachlässigen wären. Als Indikator hierfür sollte das Maß der Übereinstimmung zwischen verschiedenen Assessoren bei Konstanz der Prozesse dienen, wobei hierfür eine einfache Korrelationsanalyse verwendet wurde. Drei Teams wurden in die selbe Firma geschickt um die beiden Prozesse ENG.3: Development of Software Design und PRO.7: Management of Ressources and Schedules zu bewerten. Besonderes Augenmerk wurde darauf gelegt, dass die Assessoren eine vergleichbare Kompetenz besaßen, die Ressourcen und Indizien zwischen den Assessments nicht verändert wurden und keine Kommunikation zwischen den Teams bestand. Für die beiden Prozessbewertungen ergaben sich Korrelationen mit den Werten 0.5 bis 0.6 zum Signifikanzniveau 0.05, was eine durchaus respektable Korrelation bestätigt. Allerdings muss auch in diesem Fall wieder die interne Validität des Experiments in Frage gestellt werden, da als Datensatz nur drei Datenpunkte auf einer Skala von 0-3 (die Reifegrade 4 und 5 standen von vornherein nie zur Debatte) zur Verfügung standen. Jedoch stellt dieses Experiment einer der wenigen Versuche dar, von den vielen Fallstudien wegzukommen, hin zu kontrollierten Experimenten, die für empirische Aussagen, insbesondere Verallgemeinerung und Vorhersagen, extrem wichtig sind. Jedoch sind groß angelegte Feldexperimente aufgrund der immensen Kosten kaum zu erwarten.

4.4 Praxisbeispiel A

Ein weiteres Paper ([AvWSS04]) beschäftigt sich mit der Durchführung von SPICE-Assessments in vier kleinen Firmen in Brasilien, die alle keinen definierten Entwicklungsprozess besaßen und laut eigenen Angaben sowohl Probleme hatten die vom Kunden geforderte Qualität zu erreichen als auch gegebene Budget- und Zeitrestriktionen einzuhalten. Sie befanden sich in einem wirtschaftlichen Umfeld in dem nur rund 7% aller Softwarefirmen eine ISO 9000/CMM Zertifikation haben durchführen lassen (Von den erreichten Reifegraden ganz zu schweigen). Über die teilnehmenden Firmen lässt sich folgendes sagen:

- Firma 1 entwickelte Anwendersoftware nach Kundenauftrag und kleine Management -Systeme für den internen Gebrauch. Sie bestand zweieinhalb Jahre und hatte fünf Mitarbeiter von denen drei an dieser Studie beteiligt waren.
- Firma 2 entwickelte sowohl Standardprodukte für den freien Markt als auch auf Kundenauftrag, vorrangig Anwendersoftware für Handel und Industrie. Sie war erst ein Jahr tätig und bestand aus zwei Mitarbeitern, von denen beide den Assessoren zur Verfügung standen.

- Firma 3 entwickelte Standard-Informationssysteme für die Elektro- und Metallbranche und verbrauchte sehr viel Manpower für Support- und Maintenanceaufgaben. Sie bestand sechs Jahre und umfasste elf Mitarbeiter, davon sechs am Assessment beteiligt.
- Firma 4 erstellte Datenkommunikationssysteme für B2B und B2C, war bereits seit vier Jahren am Markt und umfasste mittlerweile 56 Mitarbeiter, von denen aber viele Neueinstellungen waren, so dass sie noch für die Zielgruppe "Small Companies" mit unter 50 Mitarbeitern akzeptiert wurde.

Es wurden in jeder Firma einzelne Prozesse bewertet, wobei erst während der Assessments letztlich feststand, welche dies im Einzelfall waren. Die Ergebnisse wurden nicht veröffentlicht, was - zusammen mit der teils minimalen Mitarbeiteranzahl - auf einen sehr geringen Reifegrad schließen lässt. Die Firmen gaben allerdings allesamt ein sehr positives Feedback bezüglich der Prozessprofile, die alle sehr gut die Stärken und Schwächen widerspiegelten, und vor allem wertvolle Hilfestellungen für Verbesserungen anbieten konnten. Die Kosten pro Assessment betrugen etwa 80 Mannstunden, was für ein kleines Unternehmen ein nicht unerheblicher Aufwand darstellt, wenn er nicht durch akademische Experimente finanziert wird. Als Ergebnisse lieferte die Studie auch einige kritische Anmerkungen:

- Die erforderlichen Formalismen bezüglich Planung und Dokumentation erforderten zu viel Aufmerksamkeit und Zeit.
- Das Budget für ein Assessment reicht maximal für ein bis zwei Schlüsselprozesse des Unternehmens, jedoch steht gerade bei solch kleinen Firmen im Vornherein nur selten fest welche dies sein könnten.
- Informelle Interviews stellten sich als effektiver heraus als formale Checklisten, da auf diese ohne erhebliches Prozessverständnis und Softwaretechnik-Erfahrung kaum eine Antwort möglich war.
- Die Level 4 und 5 sind von vornherein auszuschließen, da diese allgemein für kleine Firmen kaum zu erreichen sind.

Zusammenfassend ergab sich also ein recht positives Ergebnis. Wenn eine vereinfachte Methodik für kleine Softwarefirmen mit der Einschränkung auf die Levels 0-3 verfügbar wäre, könnten viele kleine Firmen die Vorteile für sich nutzen.

Ähnlich Ergebnisse lieferte eine Studie in Australien ([AvWSS04]), an der ebenfalls vier Unternehmen der Größe 6 bis 60 (Median: 10) teilnahmen. Jedoch wurde hier noch untersucht, wie viele der wertvollen Verbesserungsansätze tatsächlich verfolgt wurden, wobei sich zeigte, dass aufgrund von Budgetschwierigkeiten quasi nichts verändert wurde. Für die meisten Firmen dieser Kategorie stellen solche Assessments also eine nette intellektuelle Spielerei dar, wirtschaftlich jedoch oft nur Kosten ohne Nutzen.

4.5 Praxisbeispiel B

Eine letzte Studie ([EEB00]) soll nun angeführt werden, die sich mit dem Erfolg von SPICE-Assessments gemäß ISO 15505 - Teil 5 auch in größeren Firmen

beschäftigt. Wieder von El Emam durchgeführt, wurden Daten von 56 Firmen weltweit gesammelt, hauptsächlich ansässig in Europa und Südostasien, da in den USA CMMI sehr stark dominiert. Die Unternehmen stammen aus den verschiedensten Branchen, von Banken, Telekommunikation, Logistik, über militärische Einrichtungen zu Softwarehäusern und Systemherstellern. Negativ kommentiert wurde vom Autor selbst die Tatsache, dass nur tendenziell positive Bewertungen veröffentlicht wurden, da keine Firma ein "Versagen" öffentlich zugeben möchte. Weiterhin konnte nur eine einfache Korrelationsanalyse durchgeführt werden, es gab weder formulierte Hypothesen, Kontrollgruppen, noch kann durch solch eine Studie ein Ursache-Wirkungs-Zusammenhang aufgestellt werden. Die Ergebnisse waren jedoch, dass bei großen Unternehmen die Prozessbewertung stark negativ mit den Produktionskosten korreliert war, was bei kleineren nicht der Fall war. Ein Assessment mit nachfolgenden Prozessverbesserungsmaßnahmen lohnt sich also erst ab einer gewissen Größe. Sehr wertvoll kann wohl der Sensibilisierungsprozess im Top-Management sein, was Qualität angeht, und dass diese ein eigenes Budget, eigene Institutionen und eigene Methodiken erfordern. Sobald sich diese Ansicht in der Unternehmenskultur verbreitet hat, sollte sukzessive ein akzeptabler Reifegrad erreicht werden können und der volle Nutzen aus einer Reifegradermittlung gezogen werden können.

5 Vergleich mit CMMI

Vielfach werden ISO 15504 und CMMI als Konkurrenz zueinander gesehen und angeblich auf Gemeinsamkeiten und Unterschiede hin verglichen. Dies ist aber an sich bereits im Ansatz sinnlos, da CMMI laut [FR03] weitestgehend die Anforderungen erfüllt, die ISO 15504 - Abschnitt 2 an ein Prozessassessment stellt. Somit sind diese beiden nicht miteinander in Konkurrenz zu sehen, sondern CMMI ist eine Ausprägung, die ISO 15504 erfüllt. Stattdessen kann man nur das Beispielmodell SPICE aus dem Abschnitt 5 mit CMMI vergleichen und dies soll hier nun auch geschehen.

Laut [Hör06] sind beide Modelle zunächst insofern gleichwertig, als dass sie beide Best Practice Modelle sind, also versuchen, Praktiken und Methodiken die sich in der Praxis als nützlich erwiesen haben, zusammenzufassen und den Firmen als einheitlicher Rahmen zur Verfügung zu stellen. Ein offensichtlicher Unterschied besteht darin, dass das eine Modell vom SEI der Carnegie Mellon University stammt, einer Institution, die in engem Kontakt mit dem US-Verteidigungsministerium steht. CMMI sollte nicht die wirtschaftliche Lage verbessern oder den Firmen helfen ihre Prozesse in den Griff zu bekommen, sondern diente einzig dem Zweck, dem Militär das Problem abzunehmen, dass zu viel seiner von Dritten entwickelten Software unbrauchbar war. Auf der anderen Seite der ISO 15504 Standard, der von der International Organisation for Standardisation stammt, die sich zwar die Vereinheitlichung der Märkte auf die Fahnen geschrieben hat, durch den massiven Einsatz ihrer Standards aber auch gut verdient. Zum einen durch den kostenpflichtigen Vertrieb der Standards und sicherlich auch durch Vorträge, Beratungen und Schulungen durch die persönlich

involvierten Personen, die natürlich ein tief greifendes Verständnis der Normen haben.

Oftmals werden die beiden Modelle auch dahingehend unterschieden ob sie kontinuierlich oder stufenweise seien (etwa in [Wal07]), da letzten Endes aber beide eine Stufenweise Reifegraddimension haben, sind solche Unterscheidungen wohl wenig praxisrelevant oder logisch begründbar. Auch enthalten beide Modelle laut [VM98] im Grunde die selben Basispraktiken und die selben Prozesse, nur werden teilweise Unterschiede in der Abstraktionsebene oder dem Detaillierungsgrad ersichtlich, die eine direkte Zuordnung von Elementen der beiden Standards verhindern. Wäre allerdings eine solche gegenseitige Zuordnung möglich würde dies wirtschaftlich große Konsequenzen haben, da dann ja beide Assessments 100% kompatibel wären und man sich eines der beiden sparen könnte, wogegen sicherlich mindestens eine Seite der Standardisierungsgremien große Einwände hätte. Die vorhandenen Ähnlichkeiten überwiegen jedoch deutlich die Unterschiede. Ob nun ein Prozess wie Configuration Management bewertet wird, oder eine Key Process Area namens Software Configuration Management den Reifegrad mitbestimmt, ist im weitesten Sinne äquivalent. Die Ergebnisse der Assessments, also insbesondere die Korrelation der ermittelten Reifegrade und die vorgeschlagenen Verbesserungsmaßnahmen sind in beiden Modellen laut der Studie von Varkoi ([VM98]) sehr ähnlich. Mark C. Paulk vom SEI führt in seinem Paper [Pau99] einen Unterschied auf, der sich aktuell nicht verneinen lässt, jedoch mit der Weiterentwicklung von SPICE bald der Vergangenheit angehören wird: CMMI erlaubt es den Reifegrad des Gesamtunternehmens zu messen, SPICE nur für Einzelprozesse. In naher Zukunft ist eine entsprechende Erweiterung des ISO 15504 Standards bereits zu erwarten. Er führt weiterhin einige kleiner Unterschiede bezüglich der Erweiterbarkeit des Prozessmodells oder der Genauigkeit der Hilfestellungen für die Prozessverbesserung an, jedoch bleibt als Quintessenz auch in diesem Paper die sehr enge Verwandtschaft und die grundsätzliche Äquivalenz der Bewertungen zurück.

6 Resumé und Ausblick

Die Teile 6 und 7 des ISO 15504 Standards sind bereits in Arbeit bzw. kurz vor der Veröffentlichung und werden folgenden Inhalt haben: Teil 6: Exemplar Systems Life Cycle Process Assessment Model: Dieser Teil wird eine Verbindung zum ISO 15288 Standard besitzen. Teil 7: Assessment of Organizational Maturity: Dieser Teil wird die letzte große Lücke zu CMMI schließen und eine Bewertung des unternehmensweiten Reifegrades erlauben. Weiterhin muss daran gearbeitet werden, eine Ausweitung der Akzeptanz gegen das vom SEI gepushte CMMI zu erreichen, was aufgrund der Strukturähnlichkeit eigentlich durchaus möglich sein sollte, jedoch durch die oftmalige Verwechslung einer ISO 15504 konformen Bewertung und einer SPICE-Bewertung verhindert wird. Die Entscheidung den generischen Standard und die konkrete Implementierung in einer einzigen Norm zu veröffentlichen war allerdings auch nicht die beste, denn fachfremde Manager können sich sehr leicht in dem bereits komplexen Dschungel

aus ISO Normen verirren, so dass die CMMI-Variante nach außen einfacher und konsistenter wirkt. Jedoch wäre eine allgemeingültige Akzeptanz eines weiterentwickelten ISO 15504 als Einstiegspunkt der Vergleichbarkeit sicherlich dienlich, und die gewählte Methodik würde in den Hintergrund treten.

Allgemein bleibt noch festzuhalten, dass Prozessassessments für kleine Firmen aufgrund der hohen Kosten und der mangelnden Budgets für die Durchführung von Verbesserungsprogramme nicht wirklich zu empfehlen sind, die Gefahr den Prozess zu zementieren und zum Selbstzweck entarten zu lassen ist zu groß. Für größere Firmen kann die Zertifizierung durchaus von Nutzen sein, und sei es nur der recht profane Grund, dass der bevorzugte Kunde nur Lieferanten mit einem zertifzierten Reifegrad zulässt. Wie bereits von El Emam gezeigt, lassen sich durchaus Einsparungen realisieren, allerdings sind die höchsten Reifegrade eher als intellektuelle Spielerei zu betrachten, da sich kaum ein Unternehmen diesen Anforderungen gewachsen sieht. Als Abschließendes Zitat hierzu passt eine Einschätzung von Roche Diagnostics, eine Firma in der komplexeste Steuerungssoftware für Medizinischen Gerätschaften entsteht: "Roche Diagnostics strebt die SPICE Level 4 und 5 nicht an" (aus [Har07]) da die Kosten den ökonomischen Nutzen selbst für einen Unternehmenszweig mit über 10.000 Mitarbeitern deutlich übersteigen. Letztlich werden sich wohl die Stufen 2 und 3 der SPICE-Assessments weiterhin als Quasi-Standard für die größeren Unternehmen durchsetzen (bzw. äquivalente Bewertungen nach anderen Verfahren). Und sobald es ein auf kleine Firmen ausgerichtetes Prozessassessmentmodell gibt, wird eine formale Betrachtung der Stärken und Schwächen in den Unternehmen für diese durchaus einen ökonomischen Vorteil darstellen können.

Literatur

AvWSS04. A. Anacleto, C.G. von Wangenheim, CF Salviano, and R. Savi. Experiences gained from applying iso/iec 15504 to small software companies in brazil. *4th International SPICE Conference on Process Assessment and Improvement, Lisbon, Portugal*, pages 33–37, 2004.

DKW99. J.C. Derniame, B.A. Kaba, and D. Wastell. *Software Process: Principles, Methodology, and Technology*. Springer, 1999.

EE98. K. El Emam. The internal consistency of the iso/iec 15504 software process capability scale. *Software Metrics Symposium, 1998. Metrics 1998. Proceedings. Fifth International*, pages 72–81, 20-21 Nov 1998.

EEB00. K. El Emam and A. Birk. Validating the iso/iec 15504 measure of software requirements analysis process capability. *IEEE Transactions on Software Engineering*, 26(6):541–566, 2000.

EEBS96. K. El Emam, L. Briand, and R. Smith. Assessor agreement in rating spice processes. *Software Process Improvement and Practice Journal*, 2(4):291–306, 1996.

Ema98. Khaled el Emam, editor. *SPICE*. IEEE Computer Soc. Press, 1998.

FR03. Malte Foegen and Jürgen Richter. Cmm, cmmi and iso 15504 (spice), October 2003.

fS00a. International Organization for Standardization. Iso/iec 9001:2000, quality management systems - requirements. Technical report, International Organization for Standardization, 2000.

fS00b. International Organization for Standardization. Iso/iec 9004:2000, quality management systems - guidelines for performance improvements. Technical report, International Organization for Standardization, 2000.

fS02. International Organization for Standardization. Iso/iec 12207:1995 information technology. Technical report, International Organization for Standardization, 2002.

fS04. International Organization for Standardization. Iso/iec 15504:2004 information technology. Technical report, International Organization for Standardization, 2004.

fS05. International Organization for Standardization. Iso/iec 9000:2005, quality management systems - fundamentals and vocabulary. Technical report, International Organization for Standardization, 2005.

Gro05. The SPICE User Group. Automotive spice - release statement, 2005.

Har07. Johann Harer. Spice - ein schlüssel zur system- und softwarequalität in der diagnostik, 2007.

Hör06. Klaus Hörmann. *SPICE in der Praxis*. dpunkt, 1. aufl. edition, 2006.

Lev95. N.G. Leveson. *Safeware: system safety and computers*. ACM Press New York, NY, USA, 1995.

LL96. G. Le Lann. The ariane 5 flight 501 failure-a case study in system engineering for computing systems. *INRIA Research Report*, 3079, 1996.

Loo07. Han van Loon. *Process assessment and ISO IEC 15504*. Springer, 2. ed. edition, 2007.

Pau99. M.C. Paulk. Analyzing the conceptual relationship between iso/iec 15504 (software process assessment) and the capability maturity model for software. *Proceedings, Ninth International Conference on Software Quality*, pages 4–6, 1999.

Sch94. Charles H. Schmauch. *ISO 9000 for software developers*. ASQC Quality Press, 2. print. edition, 1994.

Sch00. Arnold Q. Scheuing. Maturity model for it operations - mito, 2000.

Sin96. R. Singh. International standard iso/iec 12207 software life cycle processes. *Software Process Improvement and Practice*, 2(1):35–50, 1996.

Ste00. D. Stelzer. Qualitätsmanagement in der softwareentwicklung. *Computer Reseller News Nr*, 13(2000):13, 2000.

Sti99. H. Stienen. Nach cmm und bootstrap: Spice–die neue norm für prozessbewertungen. *Industrie Management*, 15(4), 1999.

VM98. T.K. Varkoi and T.K. Makinen. Case study of cmm and spice comparison in software process assessment. In *Proc. International Conference on Pioneering New Technologies: Management Issues and Challenges in the Third Millennium Engineering and Technology Management IEMC '98*, pages 477–482, 1998.

Wal07. Ernest Wallmüller. *SPI - Software Process Improvement mit CMMI, PSP/TSP und ISO 15504*. Hanser, 1. aufl. edition, 2007.

Nutzen von Zertifikaten

Michael Stübs

Betreuer: Johannes Stammel

Zusammenfassung Diese Seminararbeit befasst sich mit Zertifikaten und dem Nutzen und den Nachteilen die dadurch für Hersteller von Produkten, ihren Konsumenten sowie den Erstellern der Zertifikate enstehen. Hierzu wird eine kurze Einführung in Zertifikate gegeben und abschließend Vor- und Nachteile gegenüber gestellt. Spezielles Augenmerk wird hierbei auf Software-Zertifikate und Sicherheit in der IT-Branche, aber auch auf Qualitätssicherung gelegt.

Schlüsselwörter: Zertifikate, Nutzen, Common Criteria, ITsec, Reifegradmodelle, Software-Zertifikate, Software-Zertifizierung, SPICE

1 Einführung

In unserer heutigen Welt mit ihrer immer größer werdenden Vielfalt an Software- und IT-Produkten ist Transparenz von großer Wichtigkeit. Hierfür wurden Zertifikate entwickelt, die Funktionen von Produkten überprüfen und somit die notwendige Transparenz herstellen. In diesem Zusammenhang spielen Sicherheitsaspekte eine sehr große Rolle. Da die Datenbestände und gleichzeitig die Vernetzung durch Internet und Intranet immer weiter zunehmen, fällt auf den Datenschutz ein Hauptaugenmerk. In diesem Zusammenhang sind

- Vertraulichkeit von Informationen
- Unversehrtheit/Integrität von Daten
- Authentizität von Daten

von großer Bedeutung, wenn man bedenkt wie sehr der Datendiebstahl und -missbrauch zugenommen hat.

Dies alles muss von Soft- und Hardware gewährleistet werden. Um dies sicherzustellen wurden, technische Richtlinien und unterschiedliche Sicherheitskriterien wie zum Beispiel die Common Criteria (CC; siehe Abs. Sicherheitskriterien) entwickelt um anhand dieser Software zu überprüfen und festzustellen, ob sie diesen Kriterien entsprechen und somit oben genannte Sicherheitsmerkmale beinhalten. Hierfür werden schließlich unterschiedliche Zertifikate von privaten und staatlichen Stellen vergeben.

In den folgenden Abschnitten werden zuerst die Frage, was Zertifikate eigentlich sind und anschließend die unterschiedlichen Zertifizierungsmöglichkeiten dargestellt. Anschließend werden Vor- sowie Nachteile untersucht und gesetzliche Grundlagen dargestellt. Abschließend werden Vor- sowie Nachteile untersucht und gesetzliche Grundlagen dargelegt

1.1 Was sind Zertifikate?

Zertifikate sind 'Bescheinigungen', die versichern, dass das überprüfte Produkte bzw. der überprüfte Prozess die zertifizierten Eigenschaften hat. Sie gibt es in den unterschiedlichsten Bereichen für unterschiedlichste Zwecke. Sie hatten ursprünglich die Bedeutung der Güte- und Qualtitätssicherung. Es wurden hierzu früher schon einige Organisationen, die auch heute noch bestehen, gegründet (z.B. RAL Deutsches Institut für Gütesicherung und Kennzeichnung e. V.), die mit unterschiedlichsten Gütezeichen dies 'zertifizierten' und so die Unternehmen überwachten und den Verbrauchern Transparenz boten. Dies bezeichnet man heute als Produktzertifizierung [Zie, aus S.83, Abs.2]. Zertifizierung lässt sich zusätzlich noch anhand der DIN EN 45011 als eine Maßnahme eines unparteiischen Dritten definieren, die aufzeigt, dass ein Erzeugnis, Verfahren oder Dienstleistung in Übereinstimmung zu bestimmten Normen durchgeführt wird [Zie, aus S.85, Abs.2.1]. Hierbei kann es sich um verschiedenste Normen handeln, die sich beispielsweise mit der Sicherheit von Maschinen oder Werkseinrichtungen befassen.

In der Informationstechnik können außerdem noch Eigenschaften wie Echtheit oder IT-Sicherheit hinzukommen. Letzteres bedeutet, dass Anwender oder Kunden vor Bedrohungen, wie zum Beispiel Verlust der Verfügbarkeit, Integrität, Vertraulichkeit und Authentizität von Daten, Dienstleistungen und Informationen geschützt werden sollen und Software somit von zuständigen Stellen (zum Beispiel vom Bundesamt für Sicherheit in der Informationstechnik, kurz BSI) zertifiziert wird [fSidI05, aus S.6 Abs.1.1].

Software-Zertifikate lassen sich hierbei in verschiedene Klassen einteilen. Zum einen gibt es Zertifikate, die die Qualität von Prozessen, wie beispielsweise den Software-Entwicklungsprozess bescheinigen (Refegradmodelle, wie z.B. CMM oder Spice) und zum anderen gibt es Zertifikate, die anhand von vorbestimmten Sicherheitskriterien (CC oder ITSec) bescheinigen ob ein Produkt diesen Kriterien entspricht. Ein kurze Einführung in die unterschiedlichen Arten der Software-Zertifizierung ist für die spätere Analyse der Vor- und Nachteile vorteilhaft.

Reifegradmodelle Die Zertifizierung nach Reifegradmodellen ist wie oben schon erwähnt behilflich die Qualität von (Software-)Prozessen zu untersuchen. Hierbei wird die Flexibilität, Dauer und Kosten eines Prozesses untersucht. Ebenfalls wird das Management eines Unternehmens auf die Fähigkeit, die Prozesse zu überwachen, zu verbessern und zu verändern und auf andere Produkte anzuwenden, beurteilt.

Als Beispiel dient hierfür Spice. Dieses Modell besitzt 6 Stufen (Reifegrade), die den Prozess kategorisieren. Diese Reifegrade geben an wie gut ein Prozess und dessen Qualität geplant, entwickelt, überwacht und verbessert wird. Hierbei ist Stufe 0 die schlechteste (incomplete) und Stufe 5 die beste (optimizing).

Sicherheitskriterien Die Begutachtung eines Produktes anhand verschiedener Sicherheitskriterien soll aufzeigen in wie weit sicherheitsspezifische Eigenschaften

erfüllt sind. Als Beispiel dienen hierzu die Information Technology Security Evaluation Criteria (ITSec) und die Common Criteria (CC). Die ITSec wurden auf Initiative von Deutschland, Großbritanien, den Niederlanden und Frankreich zur Harmonisierung der vorhanden nationalen Sicherheitskriterien[ITS91, Abs.0.7] als gemeinsame europäische Sicherheitskriterien entwickelt und im März 1998 in Kraft gesetzt. Diese Kriterien basieren auf Sicherheitsfunktionalitäten und Vertrauenswürdigkeit. Nach den ITSec wird die Funktionalität nach 10 Oberbegriffen definiert und in Funktionalitätsklassen unterteilt. Die Oberbegriffe sind:

- Identifizierung und Authentisierung
- Zugriffskontrolle
- Beweissicherung
- Protokollauswertung
- Wiederaufbereitung
- Unverfälschtheit
- Zuverlässigkeit der Dienstleistung
- Übertragungssicherheit

Die Funktionalitätsklassen definieren schließlich inwiefern diese Funktinalitäten vorhanden sein müssen.

Die Vertrauenswürdigkeit ist wiederum in 2 Teile unterteilt. Nämlich in Wirksamkeit und Korrektheit. Die Wirksamkeit gibt an ob die sicherheitsspezifischen Funktionen sich eignen die Sicherheitsziele zu erreichen[ITS91, Abs.1.14] Sie werden bewertet in wie weit sie einem direkten Angriff widerstehen können und demnach in niedrig, mittel und hoch eingestuft[ITS91, Abs.3.5].

Schließlich gibt es noch die Korrektheit der Vertrauenswürdigkeit. Sie gibt an ob die sicherheitsspezifischen Funktionen und Maßnahmen korrekt implementiert wurden[ITS91, Abs.1.16]. Sie werden in 7 Evaluationsstufen eingeteilt wobei E0 für die die niedrigste Vertrauenswürdigkeit und E6 für die höchste steht[ITS91, Abs.4.3].

Die CC wurden entwickelt, damit internationale Standards für die Zertifizierung von Sicherheitskriterien vorhanden sind. Die aktuellste Version 3.1 ist vom September 2006. Die CC sind den ITSec sehr ähnlich. Auch hier wird die Möglichkeit geboten Vertrauenswürdigkeit und Funktionalität getrennt von einander zu überprüfen und zu bewerten. Die Funktionalitätsklassen werden allerdings von Anforderungen an bestimmte Produktklassen, wie beispielsweise Firewalls oder Betriebssysteme abgelöst. Diese sind die so genannten Schutzprofile. Bei einer Zuweisung eines Produktes zu einer Produktklasse können dann anhand der passenden Schutzprofile spezielle Sicherheitsanforderungen für das Produkt erstellt werden[fSidI05, aus S.20].Die Vertrauenswürdigkeit wird bei den CC ähnlich wie bei den ITSec in 7 Vertrauenswürdigkeitsstufen (EAL-Stufen) unterteilt[fSidIb, S.2]

Die Zertifizierung nach beiden Kriterien wird in Deutschland sehr häufig vom BSI durchgeführt.

Andere Arten der Zertifizierung Neben der Zertifizierung nach Sicherheitskriterien bzw. Reifegradmodellen gibt es noch anderen Formen der Zertifizie-

rung im IT-Bereich. Hierzu gehören zum Beispiel Zertifizierungen von nicht-sicherheitsrelevanten Funktionalitäten[fSidId] oder der vollständigen Übereinstimmung mit den Herstellerangaben[Phi99, S.2 Abs.1.2]. Ersteres wird zum Beispiel vom BSI unter dem Namen: 'Zertifizierung nach technischen Richtlinien' angeboten. Hierbei geht es darum, dass Produkte, die hauptsächlich in hoheitlichen Bereichen zum Einsatz kommen, nicht-sicherheitsrelevante Funktionalitäten bieten. Hierbei wird besonderes Augenmerk auf die Interoperabilität und Zuverlässigkeit gelegt. Die technischen Richtlinien, die die Funktionalitäten bestimmen, werden vom BSI aufgrund von öffentlichem Interesse oder der nationalen Sicherheit entwickelt. Das Produkt wird dann auf die zuständigen technischen Richtlinien von einer unabhängigen, privatwirtschaftlichen Prüfstelle evaluiert. Ein Beispiel hierfür sind die technischen Richtlinien im Rahmen der elektronischen Reisepässe[fSidId, vgl.mit].

Außerdem besteht die Möglichkeit ein Produkt darauf zu überprüfen, ob es mit denen vom Hersteller angegebenen Produktbeschreibungen übereinstimmt. So genannte Prospektprüfungen werden von privaten Unternehmen nach den Prüfbestimmungen RAL-GZ 901 durchgeführt[Phi99, S.2 Abs.1.2].

Ein anderer Ansatz ist die Zertifizierung von Personen bzw. den Fähigkeiten einer Person mit bestimmten Software-Produkten umzugehen. Hier gibt es eine Reihe von Programmen, Programmiersprachen u.ä. in denen man seine Fähigkeiten zertifizieren lassen kann. Hier kann man zum Beispiel Microsoft oder IBM nennen, die eine Unmenge von Kursen und Tests anbieten, Hierbei werden Kenntnisse in verschiedenen Produkten (Microsoft Windows) erlernt und überprüft. Auch kann man sich als Software-Entwickler oder -Programmierer zertifizieren lassen. Hier bietet zum Beispiel Sun die Möglichkeit sich zum Sun Certified Java Programmer (SCJP) oder zum Sun Certified Java Developer (SCJD) auszeichnen zu lassen.

1.2 Wo gibt es Zertifikate?

Zertifikate gibt es so gut wie in allen Bereichen der Wirtschaft. Angefangen in der Nahrungsmittelindustrie bis hin zur Baubranche in der es wohl die meisten gibt, verteilen sich die Zertifikate über alle möglichen Gebiete. Für die Vergabe vieler dieser Zertifikate ist die RAL Gütergemeinschaft zuständig[e.V, S.5].

So werden häufig Baustoffe auf ihre Eigenschaften hinsichtlich bestimmter Kriterien, wie zum Beispiel auf die Anti-Graffiti Eigenschaft, überprüft und erhalten gegebenenfalls ein Zertifikat.

In der Informationstechnik werden Produkte (Datenbanken, Betriebssysteme, Chipkartenleser usw.) hauptsächlich auf ihre sicherheitsrelevanten Merkmale überprüft und zertifiziert[fSidI05, aus S.34 Abs.5,Welche Produkte können zertifiziert werden? Nur Betriebssysteme oder auch Anwendungssoftware?]. Dies geschieht meistens nach ITSec oder den Common Criteria und wird in der Regel vom Bundesamt für Sicherheit in der Informationstechnik in Verbindung mit einer von ihr lizensierten Prüfstelle durchgeführt. Zu diesen beiden Sicherheitskriterien gibt es zudem auch internationale Abkommen, die eine Mehrfachzertifizierung in unterschiedlichen Ländern verhindern soll[fSidI05, aus S.8 Abs. 1.2].

Daneben gibt es wie oben schon genannt auch andere Möglichkeiten der Zertifizierung in der IT-Branche wie zum Beispiel bei der beruflichen Weiterbildung.

2 Bedeutung von Zertifikate

2.1 Für Inhaber der Zertifikate

Werbung und Vermarktung In der Werbung haben Zertifikate je nach Wirtschaftszweig unterschiedliche Bedeutung. Bei Konsumgütern wird oft mit 'Zertifikaten' geworben, die allerdings häufig von privaten Instituten vergeben werden und somit keinen gesetzlichen Regelungen unterworfen sind. Sehr häufig auftretende Beispiele sind hierfür zum Beispiel die Bewertungen von Stiftung Warentest oder Zertifikate, die die Umwelt- oder Gesundheitsverträglichkeit bescheinigen. Hiermit werden sehr viele Produkte allerdings mit Einschränkungen (siehe Werbeverbot weiter unten) beworben.

In der IT-Industrie nimmt die Zertifizierung verschiedene Rollen ein. Für Werbung bei Privatanwendern ist diese nicht so hoch einzuschätzen. In der IT-Industrie spielt die Zertifizierung eine größere Rolle bei großen Systemen und Firmen-Software. Hier bietet die Zertifizierung unterschiedliche Möglichkeiten. Zum einen zeigt sie dem Kunden, dass die Software je nach Zertifikat und Prüfstelle mindestens einmal auf ihre Sicherheitskriterien überprüft wurde und diesen auch entsprach. Durch dieses Ergebnis verbessert sich das Vertrauensverhältnis zwischen Kunde und Hersteller und somit können wichtige Marktanteile und Kunden gehalten werden. Durch das verbesserte Vertrauensverhältnis können diese Kunden auch in anderen Bereichen als Neukunden gewonnen werden. Ebenfalls könnten dadurch generell neue Kunden angeworben werden, wenn beispielsweise ein anderer Hersteller Probleme bei der Zertifizierung bekommt und die Kunden somit die Software wechseln müssen oder wollen. Abhängig von der Software und des Kundens sind solche Zertifikate bei diesen Vorgängen mehr oder weniger wichtig. Für eine Firma bzw. deren Rechnersysteme oder vereinzelte Großrechner, die nicht mit der Außenwelt vernetzt sind und somit auch keinen Angriffen von außen ausgesetzt sind, spielen solche Zertifikate eher eine untergeordnete Rolle. Allerdings ist dies heutzutage der Ausnahmefall und bei der fast vollständigen Vernetzung und der Allgegenwart des Internets im Zusammenhang mit den immer größer werdenden Datenbeständen von persönlichen und geheimen Daten, gewinnen auch die Sicherheitskriterien und die dazugehörigen Verfahren der Zertifizierung an Bedeutung.

Auch kann die internationale Anerkennung der Zertifikate ein Argument in der Werbung darstellen. Durch diese Anerkennung lässt sich das Produkt besser international vermarkten wodurch eine Verbesserung der Marktposition im Ausland bzw. auf dem Weltmarkt zustande kommen kann. Dies sollte bei der immer wichtiger werdenden Rolle der Globalisierung ein starkes Argument für bzw. gegen ein Produkt sein. Allerdings zeigt ein Studie der schweizerischen Akkreditierungsstelle SAS in Zusammenarbeit mit dem Wirtschaftsverband EKONOMIC,Swiss, die den Nutzen und die Zufriedenheit mit akkreditierten Zertifizierungsstellen untersuchte, dass die internationale Anerkennung der Zerti-

fizierungsstellen und der Zertifikate nicht ausschlaggebend war [Zie, aus S.89 Abs.4.1]

Es ist also durchaus möglich und auch wichtig Zertifikate in das Marketing und Werbekampagnen einzubinden um Kunden anzusprechen. Es ist allerdings nicht möglich mit dem Zertifikat bzw. mit dem Namen der Prüfungsgesellschaft zu werben. Dies liegt daran, dass es ein berufständisches Werbeverbot für Wirtschaftsprüfungsgesellschaften gibt. In der Praxis hat es sich etabliert lediglich etwas in der Richtung wie: 'Dieses Produkt wurde von einer führenden Wirtschaftsprüfungsgesellschaft zertifiziert'. Auf Anfrage eines Anwenders darf das Zertifikate aber ausgehändigt bzw. vorgezeigt werden. Allerdings darf es nicht Bestandteil einer Postwurfsendung sein und auch nicht im Internet veröffentlicht werden.[Phi99, S.7]

Qualitätssicherung Die Zertifizierung eines Produktes bietet eine ausgezeichnete Möglichkeit die Qualität seines Produktes zu erhalten und zu verbessern. Schon durch die Überprüfung während der Entwicklung eines Software-Produkts, in der vernünftigerweise begleitend die Zertifizierung läuft[fSidI05, S.35], lassen sich möglichst schnell Schwachstellen, Programmierfehler oder andere Schwierigkeiten auffinden und beheben wodurch Updates oder Rückrufaktionen vermindert werden[VH07, aus S.766]. Durch die Zertifizierung eines Produktes kann außerdem Rechtssicherheit erlangt werden, da die Qualität und die sicherheitsspezifische Funktionalität nachgewiesen wurde, wodurch unter Umständen eine Haftung für mögliche Anwenderfehler vermieden werden kann[VH07, aus S.768,Abschnitt Empfehlung]. Allerdings ist dies keine pauschale Haftungsbefreiung, da lediglich eine Mindestanforderung durch die Zertifizierung erfüllt wird[Spi07, S.58, RN 157].

Auch hilft die Dokumentation, die während des Zertifizierungsprozesses notwendig ist dem Unternehmen bei eventuellen Personaländerungen in der Produktentwicklung. Die Einarbeitung wird dadurch erheblich vereinfacht wodurch ebenfalls Fehler verhindert werden können, was wiederum der Qualität zu gute kommt[VH07, aus S.767]. Allerdings kann ein Produkt auch zertifiziert werden, wenn es sich schon auf dem Markt befindet[fSidIa]. Jedoch kann hier nur von einer Qualitätsbestätigung gesprochen werden. Falls dort Fehler entdeckt werden, kann bei der Behebung eine neue Version oder gar ein neues Produkt entstehen, was natürlich ebenfalls wieder (vereinfacht)zertifiziert werden kann[fSidI05, S.34]. Hierbei spielt die Zertifizierung nach technischen Richtlinien oder die Prospektprüfung eine wichtige Rolle. Man kann dabei sicherstellen, dass Vorgaben an das Produkt (technische Richtlinien vom BSI) eingehalten werden. Außerdem können zivil- oder gar strafrechtliche Klagen (Betrug) vermieden werden, wenn eine Prospektprüfung mit positiven Ergebnis für das Unternehmen durchgeführt wurde.

Ein weiterer Punkt ist die Bewertung der Prozesse durch Reifegradmodelle. Hier besteht die Möglichkeit die Qualität des Software-Entwicklungsprozesses einstufen zu lassen, wodurch die Schwächen oder Stärken des Prozesses genau aufgezeigt werden. Dadurch kann der Entwicklungs- und auch der Entwurfspro-

zess gezielt verbessert werden, wodurch natürlich auch die Qualität zukünftiger Projekte erhöht wird. Zum Beispiel werden einfache Programmierfehler durch einen guten Entwicklungsprozess häufiger verhindert. Auch bei der Entwicklungszeit und den Kosten kann ein Unternehmen von einer Untersuchung nach einem bestimmten Reifegradmodell profitieren. Eine hohe Einstufung des Entwicklungsprozesses verlangt auch eine schnelle Entwicklung und ein gutes Kostenmanagement. Soll also eine hohe Stufe erreicht werden, muss auch dies verbessert werden, was natürlich einen positiven Effekt auf das ganze Unternehmen hat. So kann ein höherer Gewinn erzielt werden, falls die Preise gleich gelassen werden bzw. weil ein Produkt schneller entwickelt werden konnte. Dadurch und durch eine eventuelle Preissenkung entsteht zusätzlich ein Marktvorteil, der zu Neukunden führen kann.

Persönliche Vorteile Bei Zertifikaten, die Personen und ihre Fähigkeiten betreffen kann man natürlich nicht von Qualitätssicherung oder Marktanteilen sprechen. Hier geht es vielmehr um die Möglichkeit sich persönlich und beruflich weiter zu bilden und zu entwickeln. Neben eventuellem persönlichem und privatem Interesse entstehen durch ein erfolgreiches Bestehen einer solchen Prüfung Chancen auf einen besseren Arbeitsplatz mit höherer Vergütung und Aufstiegschancen. Durch die zertifizierten Fähigkeiten steigert sich der Wert der persönlichen Arbeitskraft wodurch man auch an interessanteren und komplexeren Projekten mitarbeiten kann.

2.2 Bedeutung für Kunden der Inhaber

Für die Kunden der Hersteller der Produkte haben Zertifikate und die Zertifizierung vielfältige Bedeutungen. Zuerst ist hier einmal die Qualität zu nennen. Als Kunde ist der Entwicklungsvorgang eines Produktes und speziell eines Software-Produktes nur schwer nachzuvollziehen. Firmen haben meist nicht das Wissen und die personellen Möglichkeiten ein Softwareprodukt zu analysieren und zu testen. Vor allem bei den sicherheitsrelevanten Teilen, die den Datenschutz und Ähnliches wie oben schon beschrieben betreffen ist dies in der Regel für einen Kunden nicht möglich und wegen Firmengeheimnissen und Konkurrenzkampf von den Entwicklern auch gar nicht gewollt. Allerdings sind das Bundesamt für Sicherheit in der Informationstechnik und die beteiligten Prüfstellen zur Vertraulichkeit verpflichtet[fSidI05, S.8]. Auf Grund dieser Tatsache sind Hersteller bereit die Entwicklung eines Produktes dieser unabhängigen Stelle offen zulegen und es überprüfen zu lassen. Durch die Zertifikate kann der Kunde Vertrauen in die Qualität eines Produktes soweit sie getestet wurde, haben. Er hat somit die Gewissheit, dass bestimmte sicherheitsspezifische Funktionalitäten und Bedürfnisse getestet wurden und auch vorhanden sind. Ein Zertifikat zeigt auch, dass bestimmte gesetzliche Vorschriften wie zum Beispiel Regelungen aus dem Datenschutzgesetz eingehalten werden.

Auch entsteht für den Kunden eine Art Vergleichbarkeit der Produkte. Es besteht die Möglichkeit, dass er Produkte anhand der erhaltenen Zertifikate in

Bezug auf die Funktionalitäten und sicherheitsrelevanten Merkmale vergleicht und er aufgrund dessen eine Kaufentscheidung trifft und ein bestimmtes Produkt wählt.

Dies spielt natürlich auch eine Rolle in Hinsicht auf nicht-sicherheitsspezifische Funktionen. Hier ist die Zertifizierung nach technischen Richtlinien für den Kunden (oftmals Staat) eine sehr wichtige Unterstützung. Da diese Produkte häufig in sehr sensiblen Bereichen verwendet werden ist es umso wichtiger, dass alle vorher in den Richtlinien spezifizierten Funktionen vorhanden sind und somit eine reibungslose Verwendung des Produktes sichergestellt ist. Einen ähnlichen Nutzen birgt auch die Prospektprüfung. Sie gibt dem Kunden ähnlich wie bei der Zertifizierung nach Sicherheitskriterien ebenfalls eine Hilfestellung bei dem Kauf von Produkten. Er kann den Herstellerbeschreibungen vertrauen und kennt somit die vorhandenen Funktionen und kann sich deren auch sicher sein. Somit lassen sich Produkte leichter vergleichen und man kann anhand dessen eine Kaufentscheidung treffen.

Eine Zertifizierung der Prozesse nach Reifegradmodellen können für den Kunden ebenfalls eine sehr wichtige Rolle spielen. Durch eine hohe Einstufung des Unternehmens bzw. des Prozesses wird erreicht, dass durch den guten Entwicklungsprozess Fehler in der Software verhindert werden. Durch eine hohe Einstufung kann eine Kunde somit erkennen, dass das Produkt eines Unternehmens von höherer Qualität ist. Da für eine hohe Einstufung auch eine Optimierung der Kosten eines Prozesses notwendig ist, entsteht dem Unternehmen ein Kostenvorteil, den es an den Kunden weitergeben kann. So entsteht eventuell je nach Unternehmenspolitik ebenfalls eine Ersparnis für den Kunden.

Handelt es sich um eine Zertifizierung bestimmter Fähigkeiten einer Person, sind die Kunden natürlich aktuelle und eventuell neue Arbeitgeber. Für diese haben die Zertifikate unterschiedliche Bedeutung. Zum einen dient es als Zeugniss über erlernte und vorhandene Kenntnisse und ist somit eine Hilfestellung bei der Frage nach neuem Personal und Einstellungen. Zum anderen dienen solche Workshops und Zertifizierungen der Weiterbildung und Motivation der vorhandenen Mitarbeiter. Es ist oftmals eine Abwechslung zum Alltagsgeschäft für einen Mitarbeiter wodurch seine Arbeitsmoral und die Sympathie gegenüber dem Unternehmen steigt. Dies führt, wie auch das eigentliche, neu erlernte Wissen zu einem Anstieg der Produktivität und Leistung der Mitarbeiter.

2.3 Bedeutung für Aussteller der Zertifikate

Im Zuge der Zertifizierung haben sich unterschiedlichste Geschäftsmodelle entwickelt. Eine der ersten Einrichtungen ist hier RAL. RAL wurde 1925 mit dem Ziel die technische Lieferung zu präzisieren und zu vereinheitlichen gegründet. Hierzu wurden Qualitätsanforderungen und ihre Kontrolle festgelegt für die RAL zuständig war. Heutzutage überwacht RAL eine Vielzahl von Gütezeichen aus den unterschiedlichsten Branchen[e.V, S.2].

In dem Gebiet der IT-Sicherheit gab es erste Unternehmungen schon Mitte der 80er Jahre. Allerdings wurde erst Ende 1990 mit dem BSI-Errichtungsgesetz der Forderung nach einer staatlichen Stelle, die der steigenden Bedeutung und

Vereinheitlichung der Sicherheit Rechnung tragen konnte, nachgekommen. Neben der Vergabe von Zertifikaten[BSI90, §3 Abs.1 Alt.3] hat das BSI noch vielfältige Aufgaben im Rahmen der IT-Sicherheit(siehe auch §3 Abs.1 BSIG). Neben diesen beiden gibt es noch eine Menge anderer privater Unternehmen und Vereinigungen, die Zertifizierungen und Qualitätsprüfungen anbieten. Mit einigen hat beispielsweise das BSI Anerkennungsvereinbarungen[fSidI05, S.35] (beispielsweise Siemens AG).

Daneben besteht die Möglichkeit der Akkreditierung von Unternehmen und Prüfstellen. Die Akkreditierung eines Unternehmen besagt, dass es die Arbeiten, wie z.B. prüfen, kompetent ausführen kann[Akk05, S.2]. Beim BSI wird eine Prüfstelle für einen bestimmten Kriterienkatalog(CC oder ITSec) lizensiert, wofür eine Akkreditierung nach DIN EN ISO/IEC 17025 Voraussetzung ist[fSidI05, S.14]. Diese Prüfstelle unternimmt dann die eigentliche Prüfung und das Ergebnis gibt dann den Ausschlag, ob das Produkt nach den gewählten Kriterien zertifiziert werden kann oder nicht.

Private Unternehmen ziehen somit natürlich ihren Nutzen aus dem Gewinn den ihr Betrieb mit der Zertifizierung ab wirft. Am Beispiel der Siemens AG kann man sich aber gut auch noch andere Vorteile wie zum Beispiel Steigerung des Ansehens oder Ähnliches vorstellen.

Das Anbieten von Workshops und Zertifikaten für Personen und ihre Fähigkeiten hat für die Aussteller unterschiedliche Gründe. Zum eine schulen sie ihre eigenen Mitarbeiter in den Produkten und verbessern somit ihre Arbeitsleistung. Zum anderen erhöht es den Bekanntheitsgrad und Marktanteil der eigenen Produkte. Bietet man viele solcher Workshops und Zertifikate an und schult man viele Menschen in dem Umgang mit seinen Produkten, erreicht man, dass Firmen häufiger auf die eigene Produkte zurückgreifen, da sie wissen, dass ihre Mitarbeiter damit arbeiten können. Dies hat kein unerheblichen Einfluss auf den Absatz von Produkten, da kein Unternehmen diese Produkte verwenden und kaufen würde, wenn es keine Mitarbeiter hätte, die damit umgehen können. Dies sieht man auch daran, dass alle großen Software-Hersteller (Microsoft, IBM, Oracle, SAP) solche Zertifikate anbieten.

Beim Staat, der durch Institutionen, wie RAL oder das BSI am Vorgang der Zertifizierung beteiligt ist, lassen sich ganz andere Beweggründe identifizieren. Zum einen trägt er zur Wahrung und Erhaltung, wie auch zur Entwicklung und Steigerung des Sicherheitsniveaus bei. Auf der anderen Seite verhindert er dadurch auch volkswirtschaftlichen Schaden, der durch fehlerhafte und nicht sichere Software, aber auch aufgrund anderer Mängel enstehen könnte.

Es gibt also vielfältigste Möglichkeiten und Beweggründe bei der Zertifizierung.

3 Wie sind Zertifikate gesetzlich verankert?

Gesetzlich gesehen haben Zertifikate unterschiedliche Bedeutung. Zuerst einmal der Punkt des Sachmangels. Ein Produkt ist mangelhaft, wenn es nicht die Beschaffenheit hat, die vertraglich bestimmt wurde oder aber es sich nicht für den

Gebrauch für den es vorgesehen wurde eignet[BGB04, §434]. Dies ist insofern bei Produkten wichtig bei denen die sicherheitsspezifischen Funktionalitäten wesentlicher Bestandteil des Produkts sind. Da ein Zertifikat das Vorhandensein dieser bescheinigt ist auch der Sachmangel in diesem Zusammenhang soweit entkräftigt, als dass keine weiter gehenden Aspekte im Vertrag vereinbart wurden.

Die Zertifizierung für die IT-Sicherheit ist in der BSI-Zertifizierungsverordnung (BSI-ZertV) und speziell für die Zertifizierungen für Signaturen im Signaturgesetz (SigG) geregelt. Alles was das BSI selbst betrifft ist hingegen im Gesetz über die Errichtung des Bundesamtes für Sicherheit in der Informationstechnik (BSIG) geregelt. Hier sind zum Beispiel die Aufgaben des BSI [BSI90, §3] wie auch Rahmenbedingungen, wie zum Beispiel, dass Anträge in der Reihenfolge des Eingangs auch bearbeitet werden müssen oder nach welchen Sicherheitskriterien zertifiziert werden soll, geregelt[BSI90, §4]. Im BSIZertV ist dann der eigentliche Vorgang der Zertifizierung geregelt, wie zum Beispiel:

- der Antrag und was er enthalten muss[BSI92, §1]
- die Mitwirkungspflicht des Antragstellers[BSI92, §2]
- die Veröffentlichung der Sicherheitskriterien für die Zertifizierung[BSI92, §3]
- die Erteilung und der Inhalt des Sicherheitszertifikat[BSI92, §4]

sowie

- die Mitteilungspflicht bei Erteilung oder Versagung[BSI92, §5]

4 Welche Bereiche beeinflussen die Zertifizierung

Die Zertifizierung und speziell die Software-Zertifizierung werden durch das immer größer werdende Sicherheitsbewusstsein der Bevölkerung allgemein beeinflusst. Durch die immer größere Verbreitung von Internet in Verbindung mit E-Mail als Kommunikationsmittel werden sich auch immer mehr Menschen den Risiken die im Bezug auf Vertraulichkeit und Datenschutz enstehen bewusst, was dazu führt, das auch das Sicherheitsbewusstsein steigt[fSidIc, Abs.: Warum Zertifizierung?]. Aber nicht nur bei diesen offensichtlichen Produkten sondern auch bei nicht gleich ersichtlichen Zusammenhängen gewinnt die Zertifizierung an Bedeutung und wird von ihnen beeinflusst. Macht man sich einmal klar, in welchen Bereichen es überall darauf ankommt, dass Software zuverlässig funktioniert wird man sich sehr schnell bewusst, dass auch die Zertifizierung der Sicherheit unablässig ist. Als Beispiel könnte man hier das Verkehrswesen heranziehen, bei dem Ampeln ausfallsicher sein müssen und ohne die der Verkehr in Großstädten völlig zum Erliegen kommen würde. Auch die Vorstellung, dass Fahrzeuge in Zukunft miteinander kommunizieren und beispielsweise ihre Geschwindigkeit aneinander anpassen erfordert ein gewisses Maß an Ausfall- bzw. an Eingriffssicherheit. Auch bei der Bahn, bei der die Weichen oft vollautomatisch und elektronisch gestellt werden ist die Sicherheit nach außen ein zentraler

Punkt. Auch in der Medizin oder bei der Steuerung von Industrieanlagen und Atomkraftwerken lassen sich sehr einfach ähnliche Beispiele finden[Spi07, aus S.2].

All dies beeinflusst die Zertifizierung dahingehend, dass immer neue Sicherheitskriterien entwickelt werden müssen, um die Sicherheit weiterhin gewährleisten zu können. Dies bedingt natürlich auch eine Zunahme der Zertifizierungsanträge.

5 Nachteile der Zertifizierung

Neben dem Nutzen, den die Zertifizierung bringt, gibt es natürlich auch Nachteile. Hier fallen einem zuerst die Kosten ein, die solch ein Verfahren verschlingt. Hier gibt es 2 verschiedene Kostenarten, die entstehen. Zum einen die direkten Kosten, die speziell auf das eigentliche Verfahren und die dadurch entstehenden Kosten zurückzuführen sind. Auf der anderen Seite gibt es die indirekten Kosten, die durch mögliche Umstellungen oder erhöhten Aufwand entstehen können.

Die direkten Kosten stehen im Zusammenhang mit dem aufgetretenen Aufwand. Beispielsweise werden vom BSI die Kosten für die Evaluierung, Zertifizierung und für sonstige Leistungen (Drucke des Zertifikates u.ä), die bei der Zertifizierung eines Produktes entstehen, dem Antragssteller am Ende des Zertifizierungsprozesses berechnet[fSidI05, S.17 Abs.3.6].

Die indirekten Kosten kann man im Gegensatz dazu meist nur schwer abgrenzen. Dies sind zum Beispiel Kosten, die durch notwendige Veränderungen im Produkt oder im Entwicklungsprozess entstehen. Falls hierzu Investitionen in Systeme oder Personal notwendig sind kann man diese noch bestimmen. Schwieriger wird es, wenn man den Verlust beziffern muss, der durch die eventuell anfallende Verzögerung bei der Auslieferung oder Entwicklung des Prozesses entsteht. Zum einen können hier Vertragsstrafen bei zu später Lieferung entstehen. Zum Anderen können bei Produkten, die für den öffentlichen Markt bestimmt sind, Kunden sich für ein Konkurrenzprodukt entscheiden. Auch der Reputationsverlust, der einem Unternehmen entsteht ist immens und lässt sich ebenfalls nur schwer in Zahlen darstellen.

Gleiches gilt in ähnlichem Maße auch für die Prospektprüfung. Diese verursacht ebenfalls direkte und indirekte Kosten, die die selben oder ähnliche Auswirkungen haben.

Zudem gibt es noch Nachteile, die hauptsächlich bei einer Bewertung nach Reifegradmodelle auftreten. Zum einen kommt hier der Zeitverlust, der durch kompliziertere Prozesse entstehen und zu oben genannten monetären Verlusten führen kann. Außerdem kommt dazu noch die möglicherweise notwendige psychologische Veränderung. Diese kann erforderlich werden, wenn bei den Mitarbeitern Prozesse fest eingearbeitet sind. Falls nun eine Veränderung dieser Prozesse notwendig ist, um eine Zertifizierung nach einer angestrebten Stufe zu erreichen, muss der Mitarbeiter sein Verhalten überdenken und seine Arbeitsweise umstellen und sich in neue Arbeitsschritte einarbeiten. Der Mitarbeiter könnte zum einen das Vertrauen in seine eigene Leistungsfähigkeit verlieren, da er denkt,

dass vorher alles falsch war. Zum anderen könnte eine Art Trotzreaktion von der Seite der Mitarbeiter entstehen, die den veränderten Prozess nicht annehmen wollen. Hier muss dann schließlich Überzeugungsarbeit geleistet werden.

Nachteile der technischen Richtlinien liegen in den Schwierigkeiten diese überhaupt zu erfüllen. Da es meist keinen offenen Markt für solche Produkte gibt, sondern lediglich einen Abnehmer, der diese Eigenschaften fordert, werden diese technischen Richtlinien zu Marktzutrittsbarrieren. Diese Richtlinien, die unter Umständen zu sehr hohen Kosten bei der Produktion des Produktes führen kann, müssen also von Unternehmen erfüllt werden. Bei Unternehmen, die finanziell angeschlagen sind kann es so unter Umständen dadurch zur Insolvenz kommen.

Die Nachteile bei der Zertifizierung von Programmierern oder IT-Fachkräften liegen hauptsächlich bei den Personen selbst. Falls die Workshops bzw. Zertifizierungen in Bezug auf eine interne Weiterbildungsmaßnahme in der Firma angeboten werden, besteht die Möglichkeit, dass die Firma die Kosten übernimmt. Will man sich allerdings privat fortbilden muss man diese zum Teil sehr erheblichen Kosten selbst übernehmen. Neben diesem Nachteil entsteht auch noch eine Art Zwang eine solchen Qualifikationsnachweis zu erbringen. Dies liegt daran, dass sehr viele Kurse solcher Art angeboten werden und somit eine solche Qualifizierung von einer Besonderheit zu einem Muss wird, da quasi jeder eine solche Qualifizierung besitzt und somit die Konkurrenz sehr groß ist. Dadurch schwinden die Chance auf dem Arbeitsmarkt bzw. im Unternehmen aufzusteigen, falls man ein solches Zertifikat nicht besitzt bzw. es entsteht ein Zwang sich einer solchen Weiterbildung zu unterziehen.

6 Schlussfolgerung

Die Vorteile der Zertifizierung liegen also in der Transparenz, Vergleichbarkeit und in den Marktvorteilen durch Qualitätssicherung für die Unternehmen, die eine Zertifizierung von Produkten nach beispielsweise den CC oder nach Reifegradmodellen mit sich bringt, während Nachteile hauptsächlich in den Kosten der Zertifizierung oder denen, die durch Verluste von Kunden u.ä. entstehen, liegen.

Vergleicht man Vor- und Nachteile für die Beteiligten lässt sich zeigen, dass die Zertifizierung doch lohnenswert ist. Die Kosten für die Zertifizierung oder für die Änderung eines Prozesses können natürlich enorm sein. Aber der Verlust, der einem Unternehmen entsteht, verzichtet man auf eine Zertifizierung und somit auch auf Kunden, dürfte deutlich höher sein. Und die Zertifizierung nach einer bestimmten Stufe eines Reifegradmodells fordert ja auch Einsparungen wodurch sich Kosten für das Verfahren und die Änderung des Prozesses wieder ausgleichen lassen. Ein Unternehmen kann es sich also kaum leisten auf die Zertifizierung zu verzichten, da die Marktposition erheblich geschwächt würde.

Aus Sicht des Kunden ist es unablässig auf die Zertifizierung zu achten, da es bei der heutigen Situation vor allem bei den immer komplexer werdenden Softwareprodukten unmöglich ist diese auf alle ihre sicherheitsspezifischen Aspekte zu untersuchen. Die Zertifizierung nimmt dies dem Kunden ab und der hat somit

ein gewisses Maß wie weit er sich auf die Software verlassen kann. Die Kosten der Zertifizierung, die eventuell an den Kunden weitergegeben werden sind gering im Vergleich zu den Verlusten, die durch unsichere Software entstehen würden.

Für die Ersteller der Zertifikate liegt natürlich der Hauptnutzen in dem Betrieb ihres Unternehmens und des daraus resultierenden Gewinns bzw. beim Staat in der Vereinheitlichung und Sicherung der Sicherheitskriterien und somit auch im volkswirtschaftlichen Gewinn. Die Folgen und der Verlust, der einer Volkswirtschaft aufgrund einer Sicherheitslücke in einer Softwarekomponente beispielsweise in einem Atomkraftwerk entstehen könnten sind dabei kaum vorstellbar. Die Aufwendungen ein Bundesamt wie das BSI zu betreiben sind hierbei nur ein Bruchteil davon.

Literatur

Akk05. Deutscher Akkreditierungsrat. *IAF-Leitfaden zur Anwendung des ISO/IEC Guide 62:1996*, Dezember 2005.

BGB04. Bürgerliches Gesetzbuch, 2004. 55. Auflage.

BSI90. Gesetz über die Errichtung des Bundesamtes für Sicherheit in der Informationstechnik, 17.Dezember 1990.

BSI92. Verordnung über das Verfahren der Erteilung eines Sicherheitszertifikats durch das Bundesamt für Sicherheit in der Informationstechnik, 07. Juli 1992.

e.V. RAL e.V. Historie, Kompetenzfelder, Ziele.

fSidIa. Bundesamt für Sicherheit in der Informationstechnik. BSI-Faltblatt: Kurzinformation zu BSI-Sichereitszertifikate Hersteller. http://www.bsi.bund.de/literat/faltbl/F04SicherheitszertifikateH.htm.

fSidIb. Bundesamt für Sicherheit in der Informationstechnik. Common Criteria-ISO/IEC15408. http://www.bsi.de/literat/faltbl/F06CommonCriteria.pdf.

fSidIc. Bundesamt für Sicherheit in der Informationstechnik. Zertifizierung. http://www.bsi.bund.de/zertifiz/zert/ancczer.htm.

fSidId. Bundesamt für Sicherheit in der Informationstechnik. Zertifizierung nach technischen Richtlinien. http://www.bsi.de/zertifiz/tr/index.htm.

fSidI05. Bundesamt für Sicherheit in der Informationstechnik. BSI - Zertifizierung und BSI -Produktbestätigung; Hinweise für Hersteller und Vertreiber, Februar 2005.

ITS91. Itsec. http://www.bsi.de/zertifiz/itkrit/itsec-dt.pdf, Juni 1991.

Phi99. Dr. Mathias Philipp. Software-Zertifizierung nach IDW PS 880. In *Proceedings 3rd Conference on Quality Engineering in Software Technology and VDE-ITG Workshop on Testing Non- Functional Software-Requirements (CONQUEST '99) Nürnberg*, pages 154–162, 27.-29. September 1999.

Spi07. Prof. Dr. Gerald Spindler. Verantwortlichkeit von IT-Herstellern, Nutzern und Intermediären, 2007.

VH07. Melanie Volkamer and Harald Hauff. Zum Nutzen hoher Zertifizierungsstufen nach den Common Criteria (II). *IT-Datenschutz & Sicherheit*, pages 766–768, Oktober 2007.

Zie. Prof. Dr. Kurt Wolfgang Michael Ziegler. Akkreditierung und Zertifizierung - ein Gegensatz?

Common Criteria

Gemeinsame Kriterien für die Prüfung und Bewertung der Sicherheit von Informationstechnik

Achim Kuwertz

Betreuer: Henning Groenda

Zusammenfassung Die "Common Criteria" sind ein internationaler Standard für die Entwicklung sicherer informationstechnischer Produkte. Sie wollen Vertrauen in die sichere Funktionsweise eines IT-Produktes herstellen durch aktive Prüfung und Bewertung seiner Sicherheitseigenschaften gegen eine formale Anforderungsspezifikation. Dazu definieren die Common Criteria standardisierte Formulierungen zur Spezifikation von Anforderungen an die sicherheitsbezogenen Funktionen sowie die Vertrauenswürdigkeit eines IT-Produktes. Eines der Hauptziele der Common Criteria besteht darin, Anwendern eine transparente Entscheidungshilfe für die Anschaffung sicherheitsrelevanter IT-Produkte zur Verfügung zu stellen und somit das Kundenvertrauen in diese Produkte zu erhöhen. In diesem Beitrag werden ein Überblick über die Prinzipien hinter den Common Criteria gegeben und die wichtigsten Teile des Standards vorgestellt. Abschließend wird auf Erfahrungen mit der praktischen Anwendung der Common Criteria anhand von Fallstudien eingegangen.

1 Einleitung

1.1 Motivation

Sicherheit stellt ein menschliches Grundbedürfnis dar. Sicherheit wird hierbei als idealisierter Zustand des Freiseins von Bedrohungen betrachtet. Unter Anwesenheit von Bedrohungen lässt sich dieser Zustand durch aktive Schutzmaßnahmen erreichen. Um den gesellschaftlichen Ansprüchen gerecht zu werden, muss im Zeitalter der Informationstechnologie das Sicherheitsgrundbedürfnis vor allem bei der Entwicklung informationstechnischer Produkte berücksichtigt werden. Sicherheit muss in der Informationstechnik als fortwährender Prozess betrachtet werden, welcher sensible Daten und wertbehaftete Ressourcen vor missbräuchlicher Nutzung schützen will. Die Notwendigkeit der Umsetzung von Sicherheitsmaßnahmen für IT-Produkte ergibt sich aus deren gestiegener Bedeutung im Alltag. IT-Systeme sind inzwischen in allen gesellschaftlichen Bereichen vertreten [Bund08]. Wirtschaft und Verwaltung sind einer wachsenden Abhängigkeit von einer funktionierenden IT-Infrastruktur unterworfen [Bund07c]. So kann beispielsweise die Verfügbarkeit von WWW-Diensten einen geschäftskritischen Faktor für Unternehmen darstellen. Auch im privaten Sektor werden immer mehr

sensible Daten IT-Systemen zur Verwaltung anvertraut. Das gesellschaftliche Interesse am Schutz dieser Daten vor unbefugtem Zugriff ist deswegen gesteigert.

Die Berücksichtigung des Sicherheitsgrundbedürfnisses muss sowohl bei der Konzeption als auch bei der Realisierung von IT-Produkten stattfinden. Außer der Schaffung eines Sicherheitsbewusstseins, welches die notwendige Voraussetzung für die Entwicklung sicherer IT-Produkte darstellt, muss auch das tatsächlich entwickelte Produkt gängigen Sicherheitsanforderungen genügen. Die Problematik der Realisierung sicherer IT-Produkte muss dabei unter den Randbedingungen einer stetig wachsenden und sich weiter entwickelnden Informationstechnologie betrachtet werden. Neu entwickelte IT-Produkte weisen eine tendenziell ansteigende Komplexität auf, woraus hohe Anforderungen an die Entwicklung sicherer IT-Produkte resultieren. Zusammen mit der Komplexität des Themas Sicherheit im Allgemeinen machen diese Gegebenheiten eine strukturierte Vorgehensweise erforderlich, um die gewünschten Sicherheitseigenschaften eines IT-Produktes garantieren zu können [Bund07b, Kap. 1.1]. Diese Vorgehensweise soll es Kunden ermöglichen, Vertrauen in die Angemessenheit des Sicherheitsniveaus eines IT-Produktes zu haben. Eine mögliche Grundlage dieses Vertrauens stellt dabei eine unabhängige Prüfung und Bewertung der Sicherheitseigenschaften eines Produktes anhand von Sicherheitskriterien dar.

1.2 Sicherheitskriterien in der Informationstechnik

Bei einem Kriterium handelt es sich allgemein um ein unterscheidendes Merkmal. Bei der Auswahl eines Objektes aus einer Menge ähnlicher Objekte stellt ein Kriterium ein wohldefiniertes Merkmal dar, anhand dessen sich die Objekte unterscheiden lassen. Sicherheitskriterien beschreiben sicherheitsrelevante Merkmale von IT-Produkten. Die Unterscheidung der Produkte nach diesen Merkmalen ermöglicht eine objektive Beurteilung ihrer Funktionalität und Qualität. Sicherheitskriterien dienen somit als einheitlicher Maßstab für die Beurteilung der Sicherheit von IT-Produkten und ermöglichen Vergleichbarkeit [Bund07c].

Sicherheitskriterien stellen überprüfbare Anforderungen an ein IT-Produkt. Diese reglementieren die produkteigene Sicherheitsfunktionalität sowie unter anderem Bereiche wie Entwicklungsprozess und Entwicklungsumgebung, Anwender-Dokumentation und Betrieb eines IT-Produktes. Durch diese Anforderungen sollen vor allem die Sicherheitsziele Vertraulichkeit, Integrität und Verfügbarkeit in IT-Produkten umgesetzt werden [Bund07c].

Die einheitliche Formulierung von sicherheitsrelevanten Merkmalen ermöglicht Transparenz bei der Bewertung durch unabhängige Prüfstellen [Bund08]. Als Ergebnis dieser Bewertung kann ein Zertifikat über das geprüfte IT-Produkt ausgestellt werden. Dieses unterstützt mögliche Kunden bei der Auswahl eines ihrem persönlichen Sicherheitsbedarf angepassten Produktes mit garantierten Sicherheitseigenschaften. Entwicklern von IT-Produkten dienen Sicherheitskriterien als Wegweiser für die geradlinige Realisierung sicherheitsbewusster Projekte. Sicherheitskriterien tragen somit zur Erhöhung der Qualität der entwickelten Produkte bei und können zugleich die zugrunde gelegten Entwicklungsprozesse verbessern.

1.3 Historie der Sicherheitskriterien

Nachdem eine strukturierte Vorgehensweise in Form von Sicherheitskriterien motiviert wurde, soll ein kurzer Überblick über die Entwicklungsgeschichte von Sicherheitskriterien gegeben werden [Bund07c].

- Die ersten Sicherheitskriterien wurden 1983 in den USA unter dem Namen "Trusted Computer Security Evaluation Criteria (Orange Book)" entwickelt. Sie definieren verschiedene Klassen funktionaler Anforderungen an IT-Systeme und unterscheiden diese anhand einer geforderten Sicherheitsstufe.
- Im Jahre 1989 veröffentlichte die Zentralstelle für Sicherheit in der Informationstechnik (ZSI) eine nationale Entsprechung: "Deutsche IT-Sicherheitskriterien (Grünbuch)". Dieser Kriterienkatalog enthält zusätzliche Funktionsklassen und ist für die nachträgliche Definition weiterer Klassen offen. Als wichtige Neuerung wird das Prinzip der Trennung von Funktionalität und Vertrauenswürdigkeit eingeführt.
- Die "Information Technology Security Evaluation Criteria" (ITSEC) wurden 1991 unter Beteiligung des Bundesamt für Sicherheit in der Informationstechnik (BSI) als europäische Weiterentwicklung existierender nationaler Sicherheitskriterien herausgebracht. Basierend auf den Konzepten des Grünbuchs wurde mit den ITSEC eine Harmonisierung dieser Standards verfolgt.
- 1993 wurden die kanadischen Kriterien "Canadian Trusted Computer Product Evaluation Criteria" (CTCPEC) als Kombination aus ITSEC und Orange Book entwickelt.
- 1998 wurde mit den "Common Criteria for Information Technology Security Evaluation" in Version 2.0 eine erste offizielle Fassung international anerkannter Evaluationskriterien veröffentlicht. Die neueste Version aus dem Jahr 2006 wird im Common Criteria Projekt unter der Versionsnummer 3.1 geführt und ist bei der ISO zur Standardisierung eingereicht. Sie soll über einen gegebenen Zeitrahmen die älteren Versionen (2.x) ablösen.

1.4 Gemeinsame Kriterien

Die "Common Criteria" repräsentieren den derzeitigen Stand der Technik für Sicherheitskriterien in der Informationstechnologie. Sie stellen eine Weiterentwicklung und Harmonisierung der europäischen Kriterien ITSEC, des US-amerikanischen Orange Books sowie der kanadischen Kriterien CTCPEC dar [Bund07a]. Außer inhaltlichen Verbesserungen sollten mit den Common Criteria Sicherheitskriterien geschaffen werden, welche die existierenden Standards zu einem gemeinsam und international genutzten Kriterienkatalog vereinheitlichen. Die Notwendigkeit dieser Harmonisierung wird in [Euro91] beschrieben und soll nachfolgend erläutert werden.

Aufgrund unterschiedlicher gesetzlicher Anforderungen und Zielsetzungen wurden in verschiedenen Ländern eigene, sich unterscheidende Kriterienkataloge

veröffentlicht. Für die Erstellung international genutzter und zertifizierter IT-Produkte wirkte diese Vielfalt hemmend, da bei der Entwicklung viele nationale Standards berücksichtigt werden mussten. Diese teils widersprüchlichen Anforderungen erschwerten den Entwurf und die Realisierung eines Produktes und waren aufgrund des erhöhten Personal- und Zeitaufwands mit deutlichen Mehrkosten verbunden. Des Weiteren stellt die Zertifizierung eines IT-Produktes durch eine zuständige Behörde einen an sich kostspieligen Vorgang dar. Muss dieser Vorgang aufgrund sich unterscheidender Anforderungen in allen Ländern, in denen ein Produkt auf den Markt gebracht werden soll, erneut durchgeführt werden, so trägt dies ebenfalls erheblich zu den Entwicklungskosten eines Produktes bei. Damit reduziert sich zugleich die Rentabilität des Produktes. Da die grundlegenden Konzepte der existierenden Standards ähnlich waren und man aus bereits gewonnenen Erfahrungen hohe Synergieeffekte zu erzielen hoffte, entschloss man sich zur Konsolidierung der existierenden Kriterienwerke. Auf Basis dieser harmonisierten gemeinsamen Kriterien wurde ein internationales Abkommens zur System- und Softwarezertifizierung geschlossen.

In Kapitel 2 dieses Beitrags wird eine Einführung in den offiziellen "Common Criteria"-Standard gegeben. Dabei sollen wichtige Schlüsselkonzepte vorgestellt und die Intentionen hinter den Common Criteria dargelegt werden. Ziel dieses Kapitels ist es, dem Leser ein grobes Verständnis für die Möglichkeiten einer kriterienbasierten Produktevaluation zu vermitteln. Anschließend werden in Kapitel 3 praktische Erfahrungen im Umgang mit den Common Criteria beschrieben. Anhand von Fallstudien und Erfahrungsberichten soll die Praxistauglichkeit der Common Criteria untersucht werden. Kapitel 4 schließt den Beitrag mit einer Zusammenfassung der dargestellten Inhalte und Schlussfolgerungen.

2 Die Common Criteria

Vorausgehend wurde eine einheitliche und strukturierte Vorgehensweise zur Entwicklung von IT-Produkten motiviert, welche dem bestehenden allgemeinen Sicherheitsbedürfnis Rechnung trägt. In diesem Kapitel werden die gemeinsamen Sicherheitskriterien "Common Criteria for Information Technology Security Evaluation" (kurz: **CC**) vorgestellt. Dabei werden die Intentionen und Ziele der CC dargelegt und die zu lösende Sicherheitsproblematik vorgestellt. Anhand der eingeführten Konzepte und Herangehensweisen zur Lösung dieser Problematik sollen die Möglichkeiten der CC aufgezeigt werden. Zunächst wird anhand des Aufbaus des Standards eine Übersicht über das Kapitel gegeben.

2.1 Aufbau des Standards

Die Common Criteria wurden vom Common Criteria Projekt als mehrteiliger Standard veröffentlicht. Dieses Projekt stellt einen Zusammenschluss aller an der Entwicklung der CC beteiligten Nationen dar. Die derzeit aktuellste Version der CC besteht aus drei Teilen und referenziert ein ebenfalls vom CC Projekt

veröffentlichtes Begleitdokument zur Methodik der kriterienbasierten Evaluation. Die einzelnen Teile des Standards sowie das Begleitdokument sind auf der Homepage des CC Projektes [Comm07a] verfügbar. Die in diesem Kapitel dargestellten Informationen beziehen sich ausschließlich auf die referenzierten Teile des CC-Standards sowie das Faltblatt des BSI zu den Common Criteria [Bund07a].

Der erste Teil des Standards mit dem Titel "Part 1: Introduction and general model" [Comm06] gibt eine allgemeine Einführung. Hierin werden die Intentionen und Ziele der CC beschrieben und die wichtigsten Zielgruppen vorgestellt. Das enthaltene allgemeine Modell thematisiert die Sicherheitsproblematik informationstechnischer Produkte und leitet die in den CC gewählte Herangehensweise an diese Problematik ab. Dieses Modell dient als Grundlage der Beschreibungen der anderen Teile des Standards. Der Anhang definiert die Konzepte der Schutzprofile und Sicherheitsvorgaben. Dieser erste Teil des Standard wird in den Abschnitten 2.2 bis 2.4 dieses Beitrags behandelt.

Der zweite Teil des CC-Standards trägt den Titel "Security functional components" [Comm07b] und stellt einen Katalog von Anforderungskriterien an die Funktionalität eines IT-Produktes dar. Das enthaltene Paradigma funktionaler Anforderungen verfeinert das im ersten Teil des Standards vorgestellte Sicherheitsmodell. Die funktionalen Sicherheitsanforderungen werden im Abschnitt 2.5 des Beitrags behandelt.

Der dritte Teil des Standards, "Security assurance components" [Comm07c], beschreibt ebenfalls einen Kriterienkatalog. Die hierin enthaltenen Sicherheitskomponenten bilden die Grundlage der Spezifikation von Anforderungen an die Vertrauenswürdigkeit. Das allgemeine Sicherheitsmodell wird um die in den CC umgesetzte Philosophie der Vertrauenswürdigkeit von IT-Produkten erweitert. Als Bewertungsskala der Vertrauenswürdigkeit werden sieben Vertrauenswürdigkeitsstufen eingeführt. Diese Konzepte werden in den Abschnitten 2.5 und 2.6 dieses Beitrags erörtert.

Die als Begleitdokument zum CC-Standard veröffentlichte "Common Methodology for Information Technology Security Evaluation" [Comm07d] stellt die empfohlene Methodik zur Evaluation eines IT-Produktes nach den CC vor. Hierin werden die an einer Evaluation beteiligten Interessengruppen und deren Aufgaben beschrieben. Die Vorgehensweise bei der Prüfung eines IT-Produkts wird anhand der durchzuführenden Tätigkeiten dargelegt und die als Ergebnis einer Prüfung zu erstellenden Dokumente werden definiert. Abschnitt 2.7 legt die wichtigsten Aspekte dieses Dokuments dar.

2.2 Grundlegende Konzepte und Begriffe

Zur Einführung in den Standard und die verwendete Ausdrucksweise sollen zunächst grundlegende Konzepte und Begriffe vorgestellt werden. Als IT-Sicherheitskriterien ist es die *Intention* der Common Criteria, eine international standardisierte, unabhängige Prüfung und Bewertung der Sicherheitseigenschaften von IT-Produkten zu ermöglichen. Besonderes Gewicht wird hierbei auf die Sicherheitseigenschaften Vertraulichkeit und Integrität der Daten eines IT-Produktes sowie Verfügbarkeit der vom Produkt bereitgestellten Dienste gelegt.

Anhand der genannten *Zielgruppen* Verbraucher, Entwickler und Prüfer lassen sich die Ziele der CC darlegen [Comm06, “Target audience of the CC”]. Die wichtigste Zielgruppe stellen die Verbraucher dar. Diese sollen durch Evaluation und Zertifizierung in die Lage versetzt werden, unterschiedliche IT-Produkte untereinander und gegen ihre Sicherheitsbedürfnisse abgleichen zu können. Somit soll eine objektive Begründung der Entscheidung für ein bestimmtes IT-Produkt ermöglicht werden. Für Entwickler wollen die CC ein Leitfaden zur Erstellung vertrauenswürdiger IT-Produkte sein. Dieser kann, unabhängig von einer anstehenden Bewertung, ein wertvolles Vorgehensmodell zur Entwicklung sicherer Systeme darstellen. Der Hauptnutzen des Standards für Entwickler liegt in der Unterstützung der Vorbereitung eines Produktes auf seine Evaluation. Der Standard erhöht die Chancen einer erfolgreichen Evaluation gegen Kundenvorgaben deutlich, indem er die Bestimmung und Aufteilung der notwendigen Aktionen und Verantwortlichkeiten ermöglicht. Prüfern dienen die in den CC enthaltenen Kriterien als Entscheidungsgrundlage über die Erfüllung der Anforderungen durch ein Produkt. Weiterhin legt der Standard die für eine Prüfung und Bewertung notwendigen allgemeinen Prüfertätigkeiten fest.

Bevor das grundlegende Modell hinter den CC und dessen Umsetzung erläutert wird, werden an dieser Stelle einige *wichtige*, im Standard definierte *Begriffe* vorgestellt. Besondere Aufmerksamkeit soll hierbei auf das Konzept des Evaluationsgegenstands gelegt werden, welches im Abschnitt “The TOE” von [Comm06] definiert wird.

- **TOE**: Das “Target of Evaluation” (Evaluationsgegenstand) ist der in den CC bevorzugte Begriff für das zu evaluierende Produkt. Die Flexibilität in den CC erlaubt die Evaluation von allgemeinen Produkten und ist nicht auf IT-Produkte begrenzt. Daher wird im Folgenden der Begriff TOE benutzt werden. Der Standard definiert einen TOE als “Menge von Software, Firmware und/oder Hardware, eventuell in Verbindung mit einer Anleitung”. Ein TOE kann in folgenden Beziehungen zu einem IT-Produkt stehen: Es kann sich um ein eigenes IT-Produkt handeln, in einem IT-Produkt enthalten sein oder aus verschiedenen IT-Produkten zusammengesetzt sein. Es kann sich ebenfalls um eine Technologie handeln, welche das Potenzial für ein IT-Produkt besitzt. Auch Kombinationen der oben genannten Optionen sind möglich und nicht unüblich. Als Beispiele für einen TOE nennt der Standard Anwendungen, Betriebssysteme, SmartCards, LANs samt aller Netzkomponenten, Anwendungen in Kombination mit einem Betriebsystem und entsprechender Hardware, etc.

Weitere wichtige Begriffe aus [Comm06, “Terms and definitions”] sind:

- **Vertrauenswürdigkeit**: Grundlage des Vertrauens, dass der TOE die funktionalen Sicherheitsanforderungen erfüllt.
- **Wertgüter**: Entitäten, auf welche der Besitzer des TOE mutmaßlich Wert legt.
- **Betriebsumgebung**: Die Umgebung, in welcher der TOE betrieben wird.

– **Organisatorische Sicherheitsrichtlinien**: Eine Menge von Regeln, Prozeduren oder Leitlinien, welche aktuell oder zukünftig von einer tatsächlichen oder hypothetischen Organisation in der Betriebsumgebung festgelegt werden.

Die CC wurden als flexibler Standard entworfen, welcher Anwendung bei unterschiedlichen sicherheitsrelevanten Eigenschaften von unterschiedlichen IT-Produkten finden soll. Trotz dieses Anspruchs werden gewisse sicherheitsbezogene Themen nicht behandelt. Beispiele für nicht durch die Kriterienkataloge abgedeckte Sicherheitseigenschaften sind: physikalische Aspekte wie elektromagnetische Abstrahlung, administrative Sicherheitsmaßnahmen, welche nicht direkt auf die Sicherheitsfunktionalität des IT-Produktes bezogen sind (beispielsweise organisatorische und personelle Kontrollen und Rechteeinschränkungen) sowie inhärente Eigenschaften kryptografischer Verfahren.

2.3 Allgemeines Modell

Nach der Definition der grundlegenden Begriffe soll nun das allgemeine Sicherheitsmodell der CC vorgestellt werden [Comm06, “General model”].

Ein allgemeines Sicherheitsziel ist der Schutz von Wertgütern. Ob und in wie weit einer Entität ein Wert zugeordnet wird, stellt eine meist subjektive Einschätzung durch den Besitzer der entsprechenden Entität dar. Dieser Wert kann je nach Art der Entität aus verschiedenen Anforderungen resultieren. Beispiele hierfür sind die Vertraulichkeit persönlicher Daten, die Authentizität von geschäftlichen Verträgen, die Verfügbarkeit von kommerziellen Webdiensten oder die Zugriffsrechte auf teure Ressourcen. Für die CC sind besonders die in IT-Produkten verarbeiteten Informationen als Wertgüter wichtig. Diese gilt es vor eventuellen Bedrohungen durch angemessene Gegenmaßnahmen zu schützen. Gegenmaßnahmen sollen das Risiko eines Wertverlustes minimieren. Diese Konzepte werden in Abbildung 1 dargestellt.

Die CC gehen davon aus, dass der Schutz der Wertgüter in die Verantwortlichkeit ihres Besitzers fällt. Für ihn ist der Erhalt des Wertes, welcher die Entität als Wertgut auszeichnet, von ausgeprägter Wichtigkeit. Jedoch erweckt dieser Wert auch das Interesse anderer Gruppen, deren Handeln auf eine missbräuchliche, den Interessen des Besitzers entgegengerichtete Nutzung eines Wertgut abzielt. Diese Gruppen beinhalten von Hackern und böswilligen Nutzern bis hin zu unerfahrenen Nutzern und Unfällen ein breites Spektrum an Ursachen und Auslösern von Bedrohungen. Bedrohungen bergen für den Besitzer eines Wertguts das Risiko einer Wertminderung. In Abhängigkeit von der Eintrittswahrscheinlichkeit einer Bedrohung und dem Ausmaß ihrer schädigenden Wirkungen müssen zur Reduzierung des resultierenden Risikos Gegenmaßnahmen durch den Besitzer eines Wertguts angewendet werden. Die Bandbreite der Gegenmaßnahmen ist dabei nicht auf IT-Sicherheitsmaßnahmen beschränkt. Die Anwendung von Gegenmaßnahmen kann Risiken minimieren, aber nicht unbedingt eliminieren. Die Auswahl der Gegenmaßnahmen sowie die Entscheidung, die Wertgüter trotz eventuell verbleibenden Restrisikos den Bedrohungen auszusetzen, müssen daher

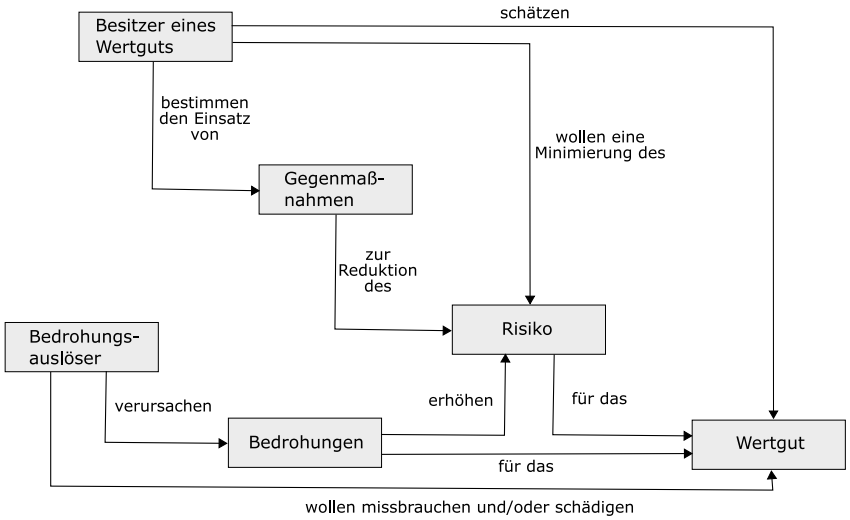


Abbildung 1. Sicherheitskonzepte und Beziehungen, nach [Comm06, S.35]

von den Verantwortlichen begründbar sein. In dieser Begründung sind die Hinlänglichkeit und die Korrektheit der Gegenmaßnahmen die maßgeblichen Faktoren. Die in Verantwortung für die Wertgüter stehenden Personen besitzen in der Regel weder die nötige Erfahrung noch die angemessenen Ressourcen für eine fundierte Beurteilung der Sicherheitseigenschaften der gewählten Gegenmaßnahmen. Auch für den Fall, dass sie sich in ihrer Begründung nicht ausschließlich auf die Zusicherungen der Entwickler eingesetzter IT-Produkte verlassen möchten, sind weitere Mechanismen zur Erhöhung des Vertrauens in die Hinlänglichkeit und Korrektheit der Gegenmaßnahmen nötig. An dieser Stelle setzen die Common Criteria an und bieten den Besitzern von Wertgütern eine qualifizierte Alternative zur Beurteilung von Funktionalität und Vertrauenswürdigkeit eingesetzter Gegenmaßnahmen. Diese Alternative besteht in einer strukturierten und transparenten Evaluation durch unabhängige Experten. Diese Zusammenhänge werden in Abbildung 2 verdeutlicht.

Die *Hinlänglichkeit* von Gegenmaßnahmen ist in den CC über das unten vorgestellte Konzept der Sicherheitsvorgaben realisiert. In diesem wird dargelegt, wie die gewählten Sicherheitsmaßnahmen den bekannten Bedrohungen entgegenwirken. Nach Definition des Standards gilt für hinlängliche Gegenmaßnahmen: Unter der Voraussetzung, dass die Gegenmaßnahmen derartig wirken, wie sie zu wirken behaupten, wirken sie den Bedrohungen für die Wertgüter entgegen. Die *Korrektheit* von Gegenmaßnahmen wird in den CC durch die Evaluation des TOE gegen seine zugehörigen Sicherheitsvorgaben realisiert. Ein TOE kann dabei fehlerhaft entworfen bzw. implementiert sein, so dass die Sicherheitsmaßnahmen nur eingeschränkt wirksam sind. Über derart entstandene Schwachstellen erhöht sich das Risiko einer missbräuchlichen Nutzung von Wertgütern und

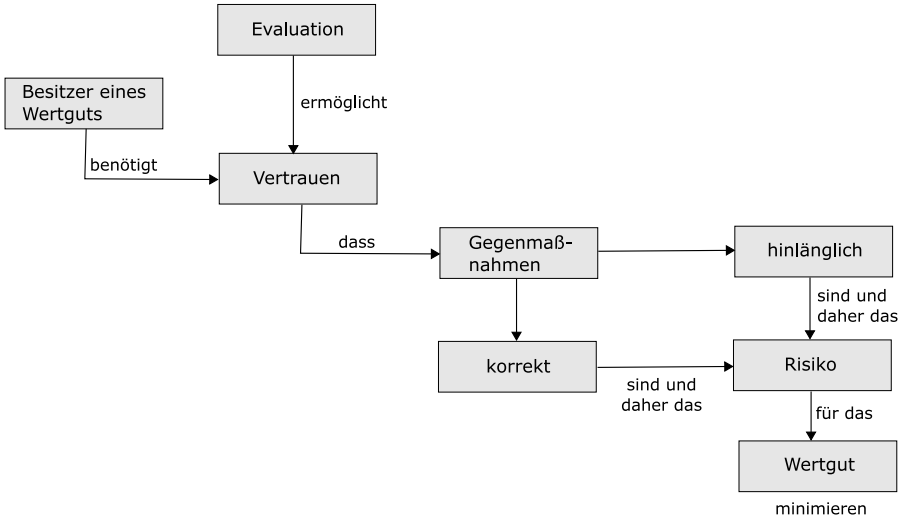


Abbildung 2. Evaluationskonzepte und Beziehungen, nach [Comm06, S.36]

steigert somit deren Verwundbarkeit. Der Standard definiert Gegenmaßnahmen als korrekt, wenn sie derartig wirken, wie sie zu wirken behaupten.

2.4 Sicherheitsvorgaben und Schutzprofile

Die Common Criteria definieren zwei wichtige Dokumente zur Anforderungsspezifikation: *Sicherheitsvorgaben* [Comm06, "Specification of Security Targets"] und *Schutzprofile* [Comm06, "Protection Profiles and Packages", "Specification of Protection Profiles"]. In diesem Kapitel soll das Konzept der CC zur Realisierung hinlänglicher Sicherheitsmaßnahmen anhand von Zweck und Inhalt dieser Dokumente dargelegt werden. Sicherheitsvorgaben und Schutzprofile ähneln sich stark in ihrem Aufbau, unterscheiden sich aber ihrer Intention nach. Zur Verfeinerung des allgemeinen Sicherheitsmodells wird zunächst auf Sicherheitsvorgaben eingegangen. Abschließend werden diese mit Schutzprofilen in Beziehung gesetzt.

Sicherheitsvorgaben (SV) stellen ein zentrales Konzept in den Common Criteria dar. Sie bilden die Grundlage der Prüfung und Bewertung eines Evaluationsgegenstands (TOE) und erfüllen dabei verschiedene Aufgaben:

- SV definieren das spezielle Sicherheitsproblem eines TOE durch Angabe der konkreten Bedrohungen und Sicherheitsrichtlinien. Daraus abgeleitet stellen sie die zu realisierenden Gegenmaßnahmen dar. Die Evaluation der SV stellt Vertrauen in der Hinlänglichkeit der Gegenmaßnahmen her.
- SV dienen als Grundlage der Vereinbarung zwischen Entwicklern und Prüfern über die zu bewertenden Eigenschaften des TOE und den genauen Umfang der Evaluation. Die Evaluation des TOE gegen seine SV stellt Vertrauen in die Korrektheit der implementierten Sicherheitsmaßnahmen her.

– Nach erfolgreicher Prüfung und Bewertung geben SV für Verbraucher die genauen Inhalte der Prüfung an und dienen somit als Vertragsgrundlage zwischen Verbraucher und Entwickler.

Sicherheitsvorgaben definieren die Anforderungen an einen TOE. Sie beinhalten außer einer Einleitung unterschiedlich formale Definitionen der zu lösenden Sicherheitsproblematik. In der Einleitung werden das Dokument und der zugehörige TOE durch eindeutige Benennungen gekennzeichnet. Weiterhin werden über verschieden detaillierte textuelle Beschreibungen Typ und sicherheitsspezifischer Funktionsumfang des TOE angegeben. Auch die für den erfolgreichen Betrieb des TOE vorausgesetzte Hard-, Soft- und Firmware wird beschrieben. Diese Einleitung soll Verbrauchern einen schnellen Überblick über den TOE ermöglichen.

Den wichtigsten Teil der Sicherheitsvorgaben in Bezug auf die Hinlänglichkeit der Gegenmaßnahmen stellen die verschiedenen Definitionen der Sicherheitsproblematik dar. Die Sicherheitsproblemdefinition, die Sicherheitsziele und die Sicherheitsanforderungen formulieren dabei das zu lösende Sicherheitsproblem in an Formalität zunehmender Art und Weise. Zunächst wird das Sicherheitsproblem anhand der drei Teilbereiche Bedrohungen, organisatorische Sicherheitsrichtlinien und Annahmen definiert. Die Beschreibung einer zu begegnenden Bedrohungen beinhaltet Verursacher, Wertgut und einer dem Wert dieses Guts zuwidergerichteten Handlung des Verursachers. Die Sicherheitsrichtlinien stellen eine Menge von Regeln und Prozeduren dar, welche durch den TOE umzusetzen sind und von einer bestimmten Organisation, wie zum Beispiel dem Betreiber des TOE oder einer staatlichen Behörde, festgelegt wurden. Die Annahmen beschreiben die Bedingungen physikalischer, personeller oder anbindungsspezifischer Art, welche von der Betriebsumgebung des TOE zu erfüllen sind, um dessen definierte Sicherheitsfunktionalität garantieren zu können.

Eine natürlichsprachlich formulierte Lösung der definierten Sicherheitsproblematik wird in den Sicherheitszielen präsentiert. Die abstrakt gehaltene Beschreibung teilt die dargestellte Problemlösung in Anforderungen an den TOE und die Betriebsumgebung auf. Die Anforderungen an den TOE sind hierbei durch dessen Sicherheitsfunktionalität zu erfüllen. Diese wird bei der Evaluation des TOE auf Korrektheit geprüft. Die Anforderungen an die Betriebsumgebung beinhalten technische und prozedurale Maßnahmen, welche den TOE bei Bereitstellung seiner Sicherheitsfunktionalität unterstützen sollen beziehungsweise für sie vorausgesetzt werden. In diese Kategorie fallen sämtliche Sicherheitsmaßnahmen, welche nicht IT-spezifisch sind. Da die CC für die Bewertung dieser Sicherheitsmaßnahmen nicht geeignet sind, werden die Sicherheitsziele für die Betriebsumgebung bei einer Bewertung nicht geprüft. Abschließend muss in einer Begründung dargelegt werden, in welcher Beziehung die gewählten Sicherheitsziele zu den zu begegnenden Bedrohungen, umzusetzenden Richtlinien und getroffenen Annahmen stehen. Diese Beziehungen sind grundlegend für eine Rechtfertigung der Auswahl der Sicherheitsziele in Bezug auf die Hinlänglichkeit der Gegenmaßnahmen: Werden alle Sicherheitsziele erreicht, dann wird das zuvor definierte Sicherheitsproblem gelöst.

Die formale Definition der Sicherheitsziele erfolgt in den Sicherheitsanforderungen. Dazu werden die natürlichsprachlich formulierten Sicherheitsziele des TOE in formal spezifizierte Anforderungen umgesetzt, unterteilt in *funktionale Sicherheitsanforderungen* und *Anforderungen an die Vertrauenswürdigkeit*. Da Sicherheitsziele für die Betriebsumgebung nicht evaluiert werden, benötigen sie keiner Formalisierung. Die formale Spezifikation der zu evaluierenden Anforderungen dient der Vermeidung von Mehrdeutigkeiten und Missverständnissen bei einer Evaluation des TOE. Des Weiteren wird eine bessere Vergleichbarkeit zwischen unterschiedlichen Sicherheitsvorgaben ermöglicht.

In Abbildung 3 werden die vorgestellten Konzepte in Beziehung zueinander gesetzt. Die Hinlänglichkeit der gewählten Gegenmaßnahmen ergibt sich aus der Hinlänglichkeit der Sicherheitsziele für den TOE in Verbindung mit den Sicherheitszielen für die Betriebsumgebung. Setzen also die funktionalen Sicherheitsanforderungen die gewählten Sicherheitsziele korrekt um, und werden die Ziele für die Betriebsumgebung erfüllt, dann wird das vorgegebene Sicherheitsproblem in Bezug auf alle Bedrohungen, organisatorische Sicherheitsrichtlinien und Annahmen gelöst.

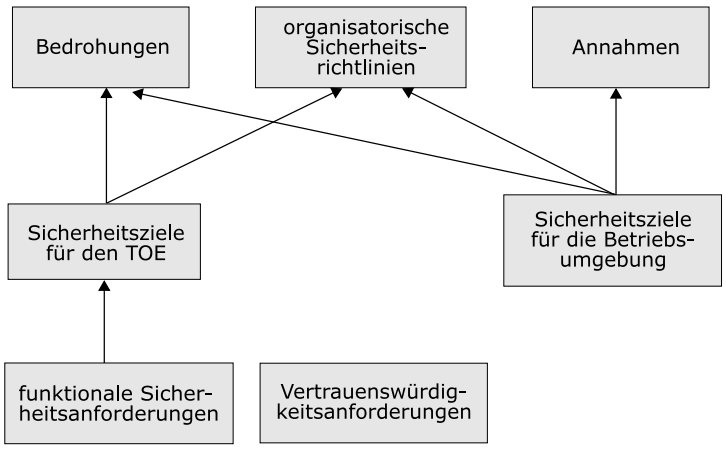


Abbildung 3. Beziehungen zwischen der Sicherheitsproblematik, den Sicherheitszielen und den Sicherheitsanforderungen, nach [Comm06, S.62]

Die zweite wichtige Klasse von Anforderungsdokumenten stellen Schutzprofile dar. Im Unterschied zu Sicherheitsvorgaben dienen Schutzprofile der Spezifikation der gewünschten Eigenschaften eines anzuschaffenden oder neu zu entwickelnden IT-Produktes. Dabei wird nicht ein konkreter TOE beschrieben, sondern allgemein ein Typ von TOEs (beispielsweise Firewalls, Betriebssysteme

oder Smartcards). Schutzprofile sollen Verbrauchern die formale, standardisierte Ausschreibung eines an ihre Sicherheitsbedürfnisse angepassten abstrakten IT-Produktes ermöglichen. Die Formalität dieser Anforderungsbeschreibung ist gegenüber anderen Formen vorteilhaft, da Schutzprofile in Bezug auf Vollständigkeit, Konsistenz und technische Korrektheit evaluiert werden können. Evaluierte Schutzprofile können als Grundlage der Definition von Sicherheitsvorgaben dienen. Diese gelten dann als konform zu einem Schutzprofil, wenn sie mindestens alle im Schutzprofil spezifizierten Anforderungen enthalten. Wie Sicherheitsvorgaben beinhalten auch Schutzprofile eine Rechtfertigung der gewählten Sicherheitsanforderungen an ein Produkt.

2.5 Sicherheitskomponenten

Nachdem auf das grundlegende Sicherheitsmodell und dessen Anforderungsdokumente eingegangen wurde, soll nun die standardisierte Formulierung dieser Anforderungen näher betrachtet werden. Die Formalisierung der Sicherheitsziele des TOE erfolgt durch *funktionale Sicherheitskomponenten*, welche im zweiten Teil des CC-Standards [Comm07b] katalogisiert sind. Anforderungen an den Evaluationsprozess und die Inhalte dieser Prüfung werden durch *Vertrauenswürdigkeitskomponenten* formalisiert. Diese bilden die Grundlage des Vertrauens in einen TOE und sind im dritten Teil des CC-Standards [Comm07c] katalogisiert.

Beide Kriterienkataloge der CC repräsentieren den aktuellen Stand der Technik in den Disziplinen Anforderungsspezifikation und Evaluation. Sie stellen eine Auswahl geeigneter Anforderungen dar, ohne Anspruch auf die vollständige Abdeckung aller möglichen Sicherheitsanforderungen zu erheben. Die Kataloge listen die Komponenten unterteilt in Klassen und Familien auf. Die Einteilung der Komponenten in Klassen erfolgt dabei anhand der umgesetzten Sicherheitsziele. Innerhalb einer Klasse werden thematisch ähnliche Komponenten zu Familien gruppiert. Jede Familie besitzt eine Verhaltensbeschreibung, in welcher die von ihr adressierten Sicherheitsziele dargelegt und die enthaltenen Komponenten vorgestellt werden. Jede Komponente fasst eine Menge von Anforderungselementen untrennbar zusammen. Die einzelnen Anforderungselemente einer Vertrauenswürdigkeitskomponente lassen sich in drei Gruppen einteilen: vom Entwickler durchzuführende Tätigkeiten, Anforderungen an notwendige Nachweisdokumente und vom Prüfer durchzuführende Tätigkeiten. Die Komponenten einer Familie sind in einer halbgeordneten Hierarchie angeordnet. Dabei ist eine Komponente hierarchisch zu einer anderen Komponente, wenn sie alle in der Komponente definierten Sicherheitsanforderungen enthält und zusätzlich eigene Anforderungen definiert. Eine hierarchische Komponente kann in Anforderungsspezifikationen an Stelle einer ihr untergeordneten Komponente verwendet werden.

Funktionale Sicherheitsanforderungen regulieren unter anderem Bereiche wie Sicherheitsprotokollierung, Schutz der Benutzerdaten, Privatsphäre, Kommunikation, Authentifizierung und Ressourcennutzung. Als Beispiele von Vertrauenswürdigkeitsklassen lassen sich die an Schutzprofile und Sicherheitsvorgaben gestellten Anforderungen nennen. Die Klassen definieren für jeden Bestandteil der Dokumente eigene Familien.

2.6 Vertrauenswürdigkeitsstufen

Der CC-Standard stellt verschiedene vordefinierte Kombinationen von Vertrauenswürdigkeitskomponenten zur Verfügung. Die wichtigsten Kombinationen werden als *Vertrauenswürdigkeitsstufen* ("Evaluation Assurance Level", **EAL**) bezeichnet. Eine EAL besteht aus einem eindeutigen Namen, einer Erklärung ihrer Ziele und Absichten und einer Menge von Komponenten. Insgesamt werden sieben hierarchisch gegliederte EALs definiert, welche sich in den Anforderungen an die Tiefe, den Umfang und die Strenge der Prüfung eines TOE unterscheiden. Eine höhere EAL beinhaltet dabei die Vertrauenswürdigkeitskomponenten einer tieferen EAL beziehungsweise Komponenten, die zu denen der tieferen EAL hierarchisch sind. Weiterhin können zusätzliche Komponenten enthalten sein.

Die grundlegende Philosophie der CC für der Herstellung von Vertrauen in ein IT-Produkt besteht in der aktiven Untersuchung der Sicherheitseigenschaften dieses Produktes. Die Art und Weise dieser Untersuchung wird über die Vertrauenswürdigkeitskomponenten beschrieben und kann durch eine EAL festgelegt werden. Die Anforderungen dieser EALs spezifizieren dabei Aspekte des Entwicklungsprozesses (z.B. Architektur, Design, entsprechende Dokumentation), Aspekte des Lebenszyklus (z.B. sichere Auslieferung, Installation, Betrieb, Nutzung eines Konfigurationsmanagementsystems), die Begleit- und Anwenderdokumentation, die Prüfung der Sicherheitsvorgaben, die Prüfung der Sicherheitsfunktionalität und die Behandlung eventueller Schwachstellen. In [Comm07c, "Evaluation assurance levels"] wird eine Auflistung der Anforderungen gegeben, die nachfolgend kurz vorgestellt wird:

- EAL1 - funktional getestet: Es werden unabhängige, d.h. von unparteiischen Dritten durchgeführte Tests gegen eine gegebene Spezifikation sowie eine Untersuchung verfügbarer Anleitungsdokumente gefordert.
- EAL2 - strukturell getestet: Zusätzlich zu den Anforderungen von EAL1 werden Tests durch Entwickler und unabhängige Tests auf Grundlage einer Architekturdokumentation gefordert. Weiterhin muss eine Schwachstellenanalyse durchgeführt werden.
- EAL3 - methodisch getestet: Unabhängiges Testen auf Grundlage einer Designdokumentation und größere Testabdeckung werden gefordert. Ferner müssen Anforderungen an die Entwicklungsumgebung erfüllt werden.
- EAL4 - methodisch entworfen, getestet und begutachtet: Es werden weitere Designbeschreibungen, verbesserter Schutz der Entwicklungsumgebung und unabhängige Tests auf Grundlage von Implementierungsdetails gefordert.
- EAL5 - semiformal entworfen und getestet: Semiformale Designbeschreibungen und eine strukturierte Architektur des TOE werden gefordert.
- EAL6 - semiformal geprüftes Design: Es werden eine Strukturierung des Entwicklungsprozesses und der Implementierungsbeschreibung, eine verbesserte Strukturierung der Architektur des TOE sowie eine semiformale Funktionsbeschreibung gefordert. Durchgeführte Analysen nehmen an Umfang zu.

- EAL7 - formal geprüftes Design: Alle Analysen müssen unter Nutzung formaler Darstellungen und einer formalen Korrespondenz ausgeführt werden.

Je größer die Anforderungen einer EAL an die Evaluation eines TOE sind, desto höher ist das erwartungsgemäß von einem Verbraucher in ein IT-Produkt gesetzte Vertrauen. Allerdings steigt mit dem Vertrauen auch der Entwicklungs- und Prüfungsaufwand eines IT-Produktes.

2.7 Gemeinsame Methodik zur Evaluation

Als Begleitdokument zu den CC haben deren Verfasser eine "*gemeinsame Methodik zur Prüfung und Bewertung der Sicherheit in der Informationstechnik*" ("Common Evaluation Methodology", **CEM**) [Comm07d] veröffentlicht. Diese beinhaltet eine auf die CC abgestimmte gemeinsame Evaluationsmethodik, welche die Vorgehensweise bei der Evaluation eines TOE festlegt. Nach der Vorstellung der Sicherheitskomponenten soll in diesem Abschnitt aufgezeigt werden, wie die Überprüfung der einzelnen Komponenten erfolgt und Vertrauen in die Korrektheit der implementierten Sicherheitsmaßnahmen hergestellt werden kann.

An der Prüfung und Bewertung eines TOE sind verschiedene Interessengruppen beteiligt: Sponsoren, Entwickler, Prüfstellen und eine autoritative Zertifizierungsstelle. Sponsoren stellen die Initiatoren und Auftraggeber einer Evaluation dar. Sie müssen dafür Sorge tragen, dass den Prüfstellen alle notwendigen Nachweise zur Verfügung gestellt werden. Der Entwickler realisiert den TOE und muss die geforderten Nachweisdokumente erstellen. Die Prüfer der Prüfstelle führen die Evaluation des TOE durch. Die Zertifizierungsstelle stellt die oberste Evaluationsautorität dar. Sie erstellt Zertifizierungsschemata, welche die in der CEM definierte Vorgehensweise erweitern und spezialisieren, und ist dafür verantwortlich, die Einhaltung dieser Schemata durch die Prüfstellen zu überwachen.

Die CC unterscheiden drei verschiedene Konzepte, welche evaluiert werden können: Schutzprofile, Sicherheitsvorgaben und TOEs. Schutzprofile und Sicherheitsvorgaben werden bei einer Evaluation auf Vollständigkeit, Konsistenz und technische Korrektheit geprüft. Als Ergebnis einer erfolgreichen Evaluation werden diese Dokumente katalogisiert und veröffentlicht. Beanspruchen Sicherheitsvorgaben Konformität zu einem oder mehreren Schutzprofilen, so müssen diese Ansprüche ebenfalls überprüft werden. Erfolgreich evaluierte Sicherheitsvorgaben dienen als Grundlage der Prüfung und Bewertung eines TOE. Diese Evaluation soll das Vertrauen in die Erfüllung der spezifizierten Sicherheitsanforderungen durch den TOE rechtfertigen. Existiert ein ausreichendes Maß an Vertrauen, so wird der Evaluationsgegenstand in einen Katalog evaluierter IT-Produkte aufgenommen.

Die in der CEM empfohlene Vorgehensweise zur CC-konformen Evaluation wird über Aufgaben und Aktivitäten definiert. Dabei besteht eine Evaluation aus vier Aufgaben: Eingang von Nachweisdokumenten, Durchführung von Prüfungsaktivitäten, Erstellen von Ausgangsdokumenten und die Demonstration der technischen Kompetenz. Zu Beginn einer Evaluation muss der Prüfer als Eingangsaufgabe sicherstellen, dass er über die korrekten Versionen aller geforderten

Nachweisdokumente verfügt und diese angemessen vor unautorisiertem Zugriff oder Modifikationen gesichert sind. Anforderungen an die Beseitigung der Nachweisdokumente nach erfolgter Evaluation und die Vertraulichkeit von sensiblen Informationen werden im Rahmen eines Zertifizierungsschemas festgelegt. Die Ausgangsaufgabe beschreibt die als Resultat einer Evaluation zu erstellenden Dokumente. Das wichtigste in der CEM definierte Dokument ist der technische Prüfbericht. Dieser enthält die Ergebnisse aller durchgeführten Prüftätigkeiten samt einer technischen Begründung dieser Ergebnisse. Die CEM legt die minimalen Bestandteile eines Prüfberichts fest. Die Demonstration der technischen Kompetenz einer Prüfstelle wird gegenüber der übergeordneten Zertifizierungsautorität durchgeführt, welche für die Festlegung der Bedingungen einer erfolgreichen Demonstration verantwortlich ist.

Die eigentliche Prüfung der spezifizierten Anforderungen an ein Dokument oder einen TOE erfolgt in den Prüfungsaktivitäten. In einer Aktivität wird dabei jeweils eine Klasse von Vertrauenswürdigkeitsanforderungen überprüft. Dabei muss für jede Vertrauenswürdigkeitskomponente einzeln entschieden werden, ob all ihre Anforderungselemente erfüllt sind. Für jedes Anforderungselement müssen alle vom Entwickler durchzuführenden Tätigkeiten sowie die bereitgestellten Nachweisdokumente überprüft werden. Als Ergebnis der Prüfung eines Anforderungselements unterscheidet die CEM drei mögliche Beurteilungen. Ein untersuchtes Anforderungselement gilt als bestanden, falls die entsprechende Prüftätigkeit mit der Feststellung beendet wurde, dass alle gestellten Anforderungen erfüllt wurden. Dies beinhaltet, dass alle geforderten Nachweise vorhanden und sowohl untereinander als auch in sich konsistent sind. Trifft eine dieser Bedingungen nicht zu, muss das entsprechende Anforderungselement als fehlerhaft beurteilt werden. Initial werden alle Anforderungselemente als ergebnislos beurteilt, bis sich ihnen eine der beiden anderen Beurteilungen zuordnen lässt. Die Überprüfung einer Vertrauenswürdigkeitskomponente gilt als bestanden, falls all ihre Anforderungen erfüllt wurden. Gleiches gilt für die Beurteilung einer Klasse auf Grundlage der Beurteilung all ihrer Komponenten. Das Gesamturteil einer Evaluation ergibt sich aus der Beurteilung aller enthaltenen Klassen. Um eine Evaluation zu bestehen, müssen all diese Klassen als bestanden beurteilt werden.

3 Die Common Criteria in der Praxis

Nachdem im letzten Kapitel die Konzepte und Möglichkeiten der Common Criteria theoretisch vorgestellt wurden, betrachtet das folgende Kapitel den praktischen Einsatz des Standards. Anhand der in verschiedenen Fallstudien dargestellten Erfahrungen sollen die allgemeine Anwendbarkeit des Standards sowie der sich aus der Anwendung ergebende Nutzen untersucht werden. Zunächst wird die Anwendung der Common Criteria zur internationalen Zertifizierung am Bundesamt für Sicherheit in der Informationstechnik vorgestellt. Daraufhin soll durch zwei Fallstudien die Sichtweise von Produktentwicklern bei Nutzung der CC in Projekten unterschiedlicher Größe und Tragweite wiedergegeben werden. Abschließend werden mit einer Bestandsaufnahme der derzeitig in IT-Produkten

realisierten Sicherheit sowie der aktuellen gesetzlichen Anforderungslage mögliche Nutzen des Standards aufgezeigt.

3.1 Common Criteria am BSI

Das Bundesamt für Sicherheit in der Informationstechnik war als deutscher Vertreter an der Entwicklung sowohl der ITSEC als auch der CC beteiligt. Damit zählt Deutschland neben den europäischen Staaten Frankreich, Großbritannien und den Niederlanden sowie den nordamerikanischen Länder USA und Kanada zu den Gründungsländern der CC. In diesem Kapitel soll die CC-basierte Zertifizierung von IT-Produkten am BSI betrachtet werden und ein internationales Abkommen zur gegenseitigen Anerkennung von zertifizierten Produkten vorgestellt werden.

Die Sicherheit eines IT-Produktes kann nicht nur ein für Kunden wichtiges Entscheidungskriterium bei der Anschaffung neuer Systeme darstellen, sondern wird je nach angestrebtem Einsatzgebiet durch gesetzliche Vorschriften verpflichtend gefordert. Um die verschiedenen gesetzlichen Anforderungen nachweislich erfüllen zu können, ist die Prüfung und Zertifizierung von IT-Produkten durch entsprechend autorisierte Behörden notwendig. In Deutschland wird diese Aufgabe gemäß BSI-Errichtungsgesetz vom Bundesamt für Sicherheit in der Informationstechnik übernommen [Bund08, “Zertifizierung”]. Das BSI stellt Sicherheitszertifikate für IT-Produkte auf Grundlage der Ergebnisse einer Evaluation aus. Diese Evaluation muss gemäß der vom BSI anerkannten Sicherheitskriterien erfolgen, zu denen die CC und ITSEC zählen. Die genauen Anforderungen an eine Evaluation werden durch das Zertifizierungsschema des BSI festgelegt. Darin ist beispielsweise die Begleitung einer durchgeführten Evaluation durch Mitarbeiter des BSI vorgesehen [Bund08, “Zertifizierung”]. Die Durchführung einer Evaluation obliegt akkreditierten Prüfstellen. Eine Prüfstelle muss zur Akkreditierung durch das BSI ein auf internationalen Normen (ISO 17025, EN 45000) beruhendes Akkreditierungsverfahren durchlaufen, um ihre Kompetenz, Unabhängigkeit und Objektivität sicherzustellen [Bund08, “Akkreditierung”]. Die Ergebnisse einer Evaluation werden von der jeweiligen Prüfstelle in Form eines Prüfberichts an das BSI weitergereicht. Als zentrale und oberste Zertifizierungsautorität muss dieses für die Einheitlichkeit der durchgeführten Evaluationen und die Gleichwertigkeit der ermittelten Prüfergebnisse Sorge tragen. Unter diesen Bedingungen wird für erfolgreich evaluierte IT-Produkte ein Zertifizierungsreport erstellt, welcher ein vom BSI ausgestelltes Sicherheitszertifikat und den Prüfbericht der Prüfstelle samt Produktbeschreibung beinhaltet. Bei Zustimmung des Herstellers wird der Report vom BSI veröffentlicht [Bund08, “Zertifizierung”]. Als Ergebnis dieses Zertifizierungsprozesses steht Anwendern ein Produkt mit garantierten Sicherheitseigenschaften zur Verfügung.

Durch das übergeordnete Zertifizierungsschema soll die Einheitlichkeit der durchgeführten Evaluationen garantiert werden. Um international Vergleichbarkeit zwischen zertifizierten IT-Produkten zu ermöglichen, ist ein nationales Schema nicht ausreichend. Eines der Ziele bei der Entwicklung der CC war die Harmonisierung existierender (supra)nationaler Standards, um eine Mehrfachzertifizie-

rung gleicher IT-Produkte bzw. gleicher Teile von IT-Produkten in verschiedenen Ländern zu vermeiden. Dieses Ziel wurde durch ein internationales Abkommen über die gegenseitige Anerkennung von IT-Sicherheitszertifikaten formalisiert [Comm00]. In diesem Abkommen vereinbaren die Gründungsländer des Common Criteria Projekts sowie Staaten wie Australien, Israel, Indien und Japan, IT-Sicherheitszertifikate und Schutzprofile auf Basis der CC bis einschließlich der vierten Vertrauenswürdigkeitsstufe (EAL4) gegenseitig anzuerkennen. Zusätzlich wurde ein europäisches Abkommen zur gegenseitigen Anerkennung von ITSEC-Zertifikaten auf CC-Zertifikate bis einschließlich EAL7 erweitert.

3.2 Fallbeispiel eines CC-basierten Softwareprojekts

Software-Entwicklung findet in Projekten unterschiedlicher Größe und Tragweite statt. Die folgenden zwei Abschnitte stellen Erfahrungen im Umgang mit den CC in verschiedenen Projekten vor. Zunächst sollen anhand des "PalME Project" [VeWW02] die Vor- und Nachteile der CC-basierten Software-Entwicklung im Bereich eines eigenständigen Einzelprojekts diskutiert werden. Das Projekt, an welchem unter anderen die TU München beteiligt war, wurde über eine Dauer von drei Monaten von einem 18-köpfigen Entwicklerteam durchgeführt. Ziel war die Entwicklung einer sicheren, Palm-basierten elektronischen Geldbörse ("Palm-based Money Exchange"). Das Projekt war als Fallstudie ausgelegt und sollte zeigen, wie auf Basis eines Sicherheitsstandards wie den CC sichere IT-Produkte entwickelt werden können. Dazu wurden die von den CC definierten Anforderungen und Tätigkeiten in einen phasenorientierten, iterativen Software-Entwicklungsprozess integriert. Dieser detailliert beschriebene fünfphasige Prozess liefert am Ende jeder Iteration ein funktionsfähiges System, welches gegen eine Menge definierter Anforderungen evaluiert wird. Die entwickelte Software sollte konform zur zweiten Vertrauenswürdigkeitsstufe (EAL2) sein, welche strukturelle Tests des TOE verlangt. Außerdem wurden für das entworfene Kommunikationsprotokoll formale Techniken eingesetzt, die konform zur höchste Vertrauenswürdigkeitsstufe (EAL7) sind.

In [VeWW02, Kap. 4&5] werden die durch das Projekt im Umgang mit den CC gewonnenen Erfahrungen dargelegt. Als zu abstrakt gehalten wird die Beschreibung wichtiger Begriffe kritisiert. Es sei oft schwierig, Sicherheitsanforderungen richtig zu interpretieren und es werde zu viel Spielraum für Mehrdeutigkeiten gelassen. Die von den CC geforderten Nachweisdokumente stellten einen Mehraufwand gegenüber einem normalen Entwicklungsprozess dar und wären aufgrund ihrer Abhängigkeiten nur mit intensivem Arbeitsaufwand konsistent zu halten. Die bei höheren EALs geforderten Entwurfsdokumente für verschiedene Abstraktionsebenen enthielten die Schwierigkeit, dass sich die einzelnen Details nur schwer einer angemessenen Abstraktionsebene zuordnen ließen. Trotz dieser Schwierigkeiten wurde die geforderte Erstellung unterschiedlicher Entwurfsdokumentationen als nützlich empfunden, da sich die Entwickler mit wichtigen Sicherheitsaspekten mehrfach und aus verschiedenen Blickwinkeln beschäftigen müssten. Kritik wird an dem Sachverhalt geübt, dass trotz der zahlreichen funktionalen Sicherheitsanforderungen die Gebiete Kommunikationsnetzwerke und

elektronischer Handel nicht angemessen abgedeckt seien. Als Ergebnis ihrer Fallstudie schlussfolgern die Autoren des Artikels, dass sich die Nutzung der Vertrauenswürdigkeitsanforderungen der CC als praktisch erwiesen hat. Werde ein Projekt von Anfang an konform zum CC-Standard durchgeführt, so ließe sich eine besser strukturierte Implementierung des TOE erreichen und die benötigte Dokumentation müsste nicht in späteren Projektphasen angepasst werden. Die Fallstudie habe die Anwendbarkeit der entwickelten CC-basierten Methodik gezeigt, da das Projekt in relativ kurzer Zeit ein hohes Sicherheitsniveau für die entwickelte Anwendung erreicht habe, mit einem in diesem Bereich unerfahrenen Team. Dabei sei laut der Mitglieder des Entwicklerteams die Entwicklung auf Basis der Common Criteria vor allem für eine frühzeitige Erkennung von Sicherheitsproblemen sehr nützlich gewesen. Abschließend empfehlen die Autoren die von ihnen gewählte Vorgehensweise.

3.3 Erfahrungen mit den CC in nationalen Projekten

Nach der Betrachtung des Nutzens der CC für kleine bis mittelgroße Einzelprojekte wird in diesem Abschnitt ein Erfahrungsbericht über den Umgang mit den Common Criteria auf Ebene nationaler Projekte vorgestellt. Er soll die Stärken und Schwächen der CC in einem von komplexen Systemen und gesetzlichen Anforderungen geprägten Einsatzbereich aufzeigen. Grundlage dieser Darlegung ist der Artikel [KeSu06], welcher als Reaktion auf einen Vorschlag des US-amerikanischen "National Institute of Standards and Technology" (NIST) veröffentlicht wurde. Dieser Vorschlag sah die Nutzung der CC und systemweiter Schutzprofile zur Spezifikation der Sicherheitsanforderungen großer IT-Systeme vor. Der Artikel beschreibt die Erfahrungen der US-Luftfahrtaufsichtsbehörde FAA mit systemweiten Schutzprofilen für ihr nationales Luftraumsystem NAS.

Das NIST stellt eine für Standardisierungen zuständige Bundesbehörde dar und war als US-Vertreter an der Entwicklung der CC beteiligt. Der Vorschlag des NIST, die CC zur Anforderungsspezifikation zu nutzen, stellt insofern ein Novum dar, als dass es sich bei den betroffenen Systemen um umfangreiche, bundesweit genutzte IT-Produkte handelt. Ein Beispiel für ein derartiges System stellt das von der FAA zur Flugsicherung eingesetzte NAS dar. Im Kontext attestierter Schwachstellen und gestiegener Sicherheitsanforderungen hatte die FAA bereits vorausgehend die Sicherheit des NAS überarbeitet und modernisiert. Hierbei wurden die Sicherheitsrichtlinien der FAA um die Forderung nach Schutzprofilen und Sicherheitsvorgaben für neu zu entwickelnde IT-Systeme ergänzt. Anhand der seither mit den CC gemachten Erfahrungen kommentiert der Artikel den Vorschlag des NIST mit einer Reihe verschiedener Hindernisse, welche die Autoren in der Anwendung der CC bei der Entwicklung landesweiter Systeme sehen.

Als ein erstes Hindernis wird die Komplexität und Größe landesweiter Projekte genannt, welche die effiziente Einführung grundlegender Änderungen in etablierten Richtlinien hindere. Bei einer Projektgröße von Tausenden von Mitarbeitern und Millionen von Kunden verursachen Änderungsanforderungen einen

hohen Kosten- und Zeitaufwand, beispielsweise für die Weiterbildung der Mitarbeiter oder die Überarbeitung der Geschäftsprozesse. Daraus resultiert nach Meinung der Autoren ein langwieriger Anpassungsprozess und das Zurückbleiben vieler landesweiter Projekte hinter ihrem Zeitplan. Vor allem gelte dies für die Umsetzung von auf Führungsebene beschlossener Änderungen. Diese benötigen viel Erfahrung, Zeit und Arbeitsaufwand, um an der Basis anzukommen und dort produktiv, korrekt und vorteilhaft umgesetzt werden zu können. Die Autoren vertreten die Meinung, dass diese spezifischen Erfahrungen für die Nutzung von Schutzprofilen nicht ausreichend vorhanden sind, vor allem in Bezug auf einen umfassenden und verpflichtenden Einsatz. So könne die Nutzung einer formalen Sprache zur Definition der Anforderungen an ein zu entwickelndes System zu Missverständnissen führen. Die Autoren sehen dabei die Gefahr, dass durch eine verfrühte Formalisierung von erhobenen Anforderungen einerseits die mangelnde Erfahrung im Umgang mit einer formalen Sprache und andererseits sich unterscheidende Sichtweisen zwischen Entwicklern und Kunden zu fehlerhaften Anforderungsspezifikationen führen können. Beispielhaft wird dieser Sachverhalt anhand von für Teile des NAS erstellten Schutzprofilen belegt, in welchen Sicherheitsanforderungen spezifiziert wurden, die über die akzeptierten Sicherheitspraktiken der FAA hinausgehen.

Als ein weiteres Hindernis führen die Autoren unflexible Standards an, welche oft ihre eigene Umsetzung innerhalb von landesweiten Projekten behindern. Derartige Standards können zu teuren und komplexen Anschaffungen führen, da sie teilweise die Nutzung wünschenswerter Produkte und Technologien verbieten. Dies gilt nach Meinung der Autoren aufgrund von Faktoren wie den hohen Evaluationskosten und der Verzögerung der Markteinführung auch für die CC. Die Autoren stellen fest, dass bisher keine US-Bundesbehörde die ausschließliche Nutzung von evaluierten Komponenten für ihre Systeme fordert. Weiterhin stellt die Flugsicherung ein Gebiet dar, in welchem sich Anforderungen, Erfahrungen und daraus resultierende eingesetzte Praktiken über Jahrzehnte hinweg entwickelt und spezialisiert haben. Bisher hatten die für das NAS entwickelten Schutzprofile diese langjährigen Sicherheitskonzepte der Luftfahrt nur ungenügend umgesetzt. Dies Autoren führen dies darauf zurück, dass den CC das sprachliche Ausdrucksvermögen zur Umsetzung der Luftfahrtsicherheitskonzepte fehlen könnte. Auch stünden Schutzziele der CC teilweise in Konflikt mit diesen Konzepten. Das Schutzziel der Vertraulichkeit widerspreche beispielsweise der Redundanz von Informationen und geteilten Zugangsmöglichkeiten, welche für den erfolgreichen Betrieb der Flugsicherung notwendig seien.

Als drittes Hindernis wird die fehlende Beziehung von CC-Schutzprofilen zu etablierten Prozessen der Systementwicklung beschrieben. Disziplinen wie die System- und Software-Entwicklung bieten ausgereifte Prozessmodelle zur Anforderungserhebung, wohingegen die CC keinen Entwicklungsprozess für die Erstellung von Schutzprofilen zur Verfügung stellen. Die Verfasser des Artikels fordern nicht nur, bestehende analytische Methoden wie Risikoanalyse und Sicherheitsplanung zu einem dynamischen Entwicklungsprozess zusammenzufassen, sondern diesen weiterhin in etablierte Systementwicklungsprozesse zu integrieren. Die

Spezifikation der Anforderungen in Form von Schutzprofilen stehe im Gegensatz zur Dynamik eines Sicherheitsprozesses. Dieser führe zu Weiterentwicklungen und Änderungen der zugrunde liegenden Systeme, während Schutzprofile statische Strukturen darstellen, deren Schwerpunkt auf einer stabilen Menge von Anforderungen im Hinblick auf eine Evaluation liege.

Insgesamt betrachten die Verfasser des Artikels das Konzept der Anforderungsdefinition über Schutzprofile als unausgereift, da derzeit keine Studien existieren, welche überzeugende Beweise für die Verbesserung der Sicherheit gegenüber der klassischen Anforderungserhebung erbringen. Weiterhin sind der Zeitaufwand für die Erstellung von Schutzprofilen sowie die Kosten einer Evaluation hoch. Trotz dieser Kritik existiere ein Bedürfnis für korrekte und verifizierbare Sicherheitsanforderungen. Der Weg, diese zu erreichen, ist für die Autoren des Artikels allerdings noch unklar.

3.4 Nutzen hoher EALs in der Software-Entwicklung

Die Betrachtung des praktischen Nutzens der Common Criteria soll nach den dargelegten Erfahrungsberichten mit einer Bestandsaufnahme der aktuellen Sicherheitslage bei IT-Produkten und den Effekten der Anwendung hoher Vertrauenswürdigkeitsstufen bei der Entwicklung von IT-Produkten abgeschlossen werden. Nachdem der letzte Abschnitt die Schwierigkeiten der CC auf Ebene nationaler Projekte vorgestellt hat, sollen hier die Nutzeffekte hoher EALs dargelegt und ihre Vor- und Nachteile herausgestellt werden. Grundlage dieser Betrachtung sind die beiden in "*Datenschutz und Datensicherheit*" veröffentlichten Artikel [VoHa07a] und [VoHa07b].

Im ersten Teil des Beitrags betrachten die Verfasser die aktuelle Situation in der Software-Entwicklung. Diese stellt die Grundlage für die Beurteilung der Effekte hoher EALs dar. Besonderes Gewicht wird auf die geltenden gesetzlichen Anforderungen an IT-Produkte sowie auf die erreichte Qualität gelegt. Diesbezüglich stellen die Autoren fest, dass Computersysteme heutzutage oft in kritischen Prozessen eingesetzt werden, ohne dass Qualitätskontrollen für Soft- und Hardware verbindlich vorgeschrieben sind. Diese Kontrollen seien in anderen Ingenieurwissenschaften üblich. Als Konsequenz sehen die Autoren die Notwendigkeit von häufigen Aktualisierungen dieser Systeme. Zur Vermeidung dieser Problematik empfehlen die Autoren, für Eigenschaften von Computersystemen Garantien auf Grundlage wissenschaftlich abgesicherter Vorgehensweisen zu geben. Vertrauenswürdigkeitsstufen ab Stufe vier fordern den Einsatz formaler Entwicklungsmethoden, mit welchen nach Meinung der Autoren derartige Garantien realisiert werden können. Da der Einsatz formaler Methoden mit erheblichem Aufwand verbunden ist, erschließe sich ihr voller Nutzen nur bei verbindlich vorgeschriebener Anwendung. Aktuell erkennen die Autoren wesentliche gesetzliche Anforderungen zur Evaluation von IT-Produkten nur in den Bereichen digitale Signaturkomponenten, digitale Tachographen, Gesundheitskarte und elektronischer Reisepass. Weiterhin existieren Selbstverpflichtungen diverser Berufsverbände, welche die Zertifizierung eingesetzter IT-Produkte verlangen. In

der freien Wirtschaft hingegen werde eine Evaluation nur zu Werbungszwecken eingesetzt, da man sich mit zertifizierten Produkten vor eventuellen Konkurrenten auszeichnen könne. Die Autoren haben ermittelt, dass weltweit hauptsächlich Produkte aufgrund gesetzlicher Anforderungen evaluiert werden, auch wenn sich bei für die IT-Infrastruktur wichtigen Systemen (wie beispielsweise Firewalls) eine Zunahme der Evaluationen abzeichnet. In dieser derzeitigen Lage wird nach Meinung der Autoren ein wichtiger Faktor größtenteils ignoriert: Haftung für fehlerhafte Produkte in Form von Produkthaftung und persönlicher Haftung der Entwickler. Ursächlich sei dies unter anderem darin begründet, dass der Nachweis, dass z.B. ein Datenverlust tatsächlich durch das Produkt verursacht wurde und nicht auf unsachgemäße Nutzung zurückzuführen ist, nur schwer zu führen sei. Einen ersten Schritt in Richtung der Produkthaftung sehen die Verfasser in der teils in nationalen Richtlinien zum Risikomanagement enthaltenen Forderung nach einer nachweislich verlässlichen Funktionsweise der unternehmenseigenen Datenverarbeitung. So könne sich allmählich ein Zwang zur Evaluation von IT-Produkten nach formalen Kriterien ergeben.

Im zweiten Teil des Beitrags werden Vor- und Nachteile der Evaluation von IT-Produkten dargelegt. Als ein Vorteil wird die Transparenz genannt, welche die Nachvollziehbarkeit der Funktionsweise der Software sowie ihres Entwicklungsprozesses ermöglicht. Kunden können somit den Nutzen zertifizierter Produkte besser einschätzen. Hersteller evaluierter Produkte können außer von einem Imagegewinn von konkreten technischen und wirtschaftlichen Vorteilen profitieren. Diese Vorteile sehen die Autoren beispielsweise in einer möglichen Senkung der Zahl der nicht eingeplanten Produktaktualisierungen. Als Begründung dieser Vorteile nennen die Autoren die aktive Untersuchung möglicher Schwachstellen, welche Entwurfs- und Programmierfehler aufdecken und Möglichkeiten zur Umgehung der implementierten Sicherheitsmechanismen aufzeigen kann. Durch die Einbeziehung der Dokumentation in den Prüfungsprozess werde eine umfassende, detailreiche und exakte Beschreibung des zu evaluierenden Produkts gefordert. Die Autoren vertreten die Ansicht, dass dies die Wiederverwendbarkeit einer Software als Ganzes oder von Teilen dieser Software bei der Entwicklung neuer Produkte erhöht. Durch die Dokumentation lassen sich einzelne Komponenten auch langfristig gut auffinden und verwalten. Auch für die Verwaltung des im Laufe eines Projektes erzeugten Wissens bringen die strikten Anforderungen an die Dokumentation Vorteile mit sich. Nach Auffassung der Autoren können Projekte trotz des Ausfalls von Personal rechtzeitig beendet werden, da sich die Einarbeitungszeit für neue Mitarbeiter verkürzen lässt. Weiterhin geht das Wissen über projektspezifische Details oder allgemeine Erfahrungen nicht mit der Zeit oder dem Ausscheiden von Mitarbeitern verloren. Auch für die Komposition eines Systems aus verschiedenen nach den CC zertifizierten Komponenten sehen die Verfasser des Beitrags einen Vorteil. Hierbei werde für die Auswahl zueinander passender Komponenten eine erhöhte Transparenz in Bezug auf die Sicherheitseigenschaften des resultierenden Gesamtsystems geschaffen. Somit lassen sich erfolgreich neue zusammengesetze Systeme mit höheren Sicherheitsanforderungen realisieren.

Der Beitrag nennt weiterhin die Nachteile der Zertifizierung von IT-Produkten nach hohen Vertrauenswürdigkeitsstufen. Diese werden unterteilt in konzeptionelle beziehungsweise technische Nachteile einerseits und ökonomische andererseits. In erstere Kategorie fallen die folgenden zwei Punkte. Als übertrieben bezeichnen die Autoren der Einsatz formaler Methoden in Bereichen, in welchen diese Methoden sich nicht für die Realisierung bestimmter Funktionen eignen. Weiterhin benötigen formale Methoden aufgrund ihrer Komplexität den Einsatz angepasster Werkzeuge. Sinnvoll sei dies nur, wenn diese Werkzeuge vor ihrer Nutzung ihrerseits zertifiziert beziehungsweise qualifiziert worden seien. In die zweite Kategorie der Nachteile fallen die hohen Kosten der Hersteller für Entwicklung, Evaluation und Zertifizierung nach den CC. Hohe EALs erfordern oft die Neuentwicklung von Produkten, da sich die Arbeitsschritte zur Erfüllung der Anforderungen an die Dokumentation nur selten nachholen lassen. Auch nennt der Artikel einen Mangel an Fachkräften, welche formale Spezifikationen erstellen und zugehörige Beweise führen können, als einen sich hinderlich auf die Umsetzung hoher EALs auswirkenden Faktor. Nachteilig sei weiterhin, dass die Zertifizierung die Festlegung des Funktionsumfangs der Produkte erfordere und diese somit relativ statisch mache. Durch diese Festlegung wird die Weiterentwicklung von Produkten behindert, da relevante Änderungen eine erneute Evaluation und Zertifizierung mit all ihren Kosten und Zeitaufwand nach sich ziehen. Abschließend empfehlen die Autoren den Einsatz geeigneter formaler Methoden in der Entwicklung von IT-Produkten, da sich entstehende Kosten aufgrund der frühzeitigen Erkennung und Vermeidung von Schwachstellen mittelfristig rentieren. Weiterhin plädieren sie für eine gesetzliche Regelung bezüglich der Zertifizierung sicherheitskritischer Systeme. Davon profitierten sowohl Kunden als auch Hersteller. Kunden erhielten klare Informationen über den Leistungsumfang eines Produktes, Hersteller erlangten Rechtssicherheit über die Qualität ihrer Produkte. Dieser Mehrwert rechtfertige auch höhere Preise.

4 Fazit

In diesem Beitrag wurden Kriterien zur Evaluation der Sicherheit von IT-Produkten vorgestellt und anhand des “Common Criteria”-Standards vertiefend behandelt. Es wurden die Absichten und Konzepte hinter Sicherheitskriterien im Allgemeinen und den CC im Speziellen dargelegt und die historische Entwicklung der Kriterien vorgestellt. In Kapitel 2 wurde ein Überblick über den CC-Standard gegeben, das zugrunde liegende Sicherheitsmodell mit Bedrohungen und Wertgütern vorgestellt und die Philosophie der Herstellung von Vertrauen in die Sicherheit von Produkten durch aktive Untersuchung und Bewertung erläutert. Weiterhin wurden wichtige Konzepte wie Schutzprofile und Sicherheitsvorgaben sowie die gemeinsame Evaluationsmethodik vorgestellt. In Kapitel 3 wurde auf die Anwendung der CC in der Praxis eingegangen. Unter anderem wurde das Zertifizierungsverfahren des BSI auf Grundlage der CC vorgestellt. Weiterhin wurde versucht, einen Überblick über Nutzen und Schwierigkeiten der Anwendung der CC zur Entwicklung sicherer Produkte zu geben, einerseits anhand

von Projekten unterschiedlicher Tragweiten, andererseits anhand der aktuellen Sicherheitslage eingesetzter Computersysteme und der derzeitigen Gesetzeslage. Als Ausblick bleibt zu sagen, dass die Entwicklung zukünftiger IT-Produkte nach formalen Kriterien wie den CC sich als nützlich und erforderlich erwiesen hat, allerdings für den Einsatz in großem Umfang noch Nachbesserungen an den Kriterien und größere Erfahrung mit ihrem Umgang notwendig erscheinen.

Literatur

Bund07a. Bundesamt für Sicherheit in der Informationstechnik (BSI). Faltblatt zu den Common Criteria. http://www.bsi.de/literat/faltbl/F06CommonCriteria.pdf, 2007. [Letzter Zugriff: 2008-02-08].

Bund07b. Bundesamt für Sicherheit in der Informationstechnik (BSI). IT-Sicherheit auf Basis der Common Criteria - ein Leitfaden. http://www.bsi.de/cc/cc_leitf.pdf, 2007. [Letzter Zugriff: 2008-02-08].

Bund07c. Bundesamt für Sicherheit in der Informationstechnik (BSI). IT-Sicherheitskriterien. http://www.bsi.de/zertifiz/itkrit/itkrit.htm, 2007. [Letzter Zugriff: 2008-02-08].

Bund08. Bundesamt für Sicherheit in der Informationstechnik (BSI). Zertifizierung und Akkreditierung. http://www.bsi.de/zertifiz/index.htm, 2008. [Letzter Zugriff: 2008-02-08].

Comm00. Common Criteria Project. Arrangement on the Recognition of Common Criteria Certificates in the field of Information Technology Security. http://www.commoncriteriaportal.org/public/files/cc-recarrange.pdf, Mai 2000. [Letzter Zugriff: 2008-02-08].

Comm06. Common Criteria Project. Common Criteria for Information Security Evaluation. http://www.commoncriteriaportal.org/public/files/CCPART1V3.1R1.pdf, September 2006. Version 3.1, revision 1. Part 1: Introduction and general model. [Letzter Zugriff: 2008-02-08].

Comm07a. Common Criteria portal. The official website of the Common Criteria Project. http://www.commoncriteriaportal.org, 2007. [Letzter Zugriff: 2008-02-08].

Comm07b. Common Criteria Project. Common Criteria for Information Security Evaluation. http://www.commoncriteriaportal.org/public/files/CCPART2V3.1R2.pdf, September 2007. Version 3.1, revision 2. Part 2: Security functional requirements. [Letzter Zugriff: 2008-02-08].

Comm07c. Common Criteria Project. Common Criteria for Information Security Evaluation. http://www.commoncriteriaportal.org/public/files/CCPART3V3.1R2.pdf, September 2007. Version 3.1, revision 2. Part 3: Security assurance requirements. [Letzter Zugriff: 2008-02-08].

Comm07d. Common Criteria Project. Common Methodology for Information Technology Security Evaluation. http://www.commoncriteriaportal.org/public/files/CEMV3.1R2.pdf, September 2007. Version 3.1, revision 2. [Letzter Zugriff: 2008-02-08].

Euro91. Europäische Kommission. Information Technology Security Evaluation Criteria – ITSEC. http://www.bsi.de/zertifiz/itkrit/itsec-dt.pdf, Juni 1991. Version 1.2. [Letzter Zugriff: 2008-02-08].

KeSu06. Feisal Keblawi und Dick Sullivan. Applying the Common Criteria in Systems Engineering. *IEEE Security & Privacy*, 4(2), 2006, S. 50–55.

SS-F02. *SIGSOFT '02/FSE-10: Proceedings of the 10th ACM SIGSOFT symposium on Foundations of software engineering*, New York, NY, USA, 2002. ACM. General Chair-Mary Lou Soffa and Program Chair-William Griswold.

VeWW02. Monika Vetterling, Guido Wimmel und Alexander K. Wißpeintner. Secure systems development based on the common criteria: the PalME project. In *SIGSOFT FSE* [SS-F02], S. 129–138. General Chair-Mary Lou Soffa and Program Chair-William Griswold.

VoHa07a. Melanie Volkamer und Harald Hauff. Zum Nutzen hoher Zertifizierungsstufen nach den Common Criteria (I). *Datenschutz und Datensicherheit - DuD*, 31(9), September 2007, S. 692–695.

VoHa07b. Melanie Volkamer und Harald Hauff. Zum Nutzen hoher Zertifizierungsstufen nach den Common Criteria (I). *Datenschutz und Datensicherheit - DuD*, 31(10), October 2007, S. 766–768.

Microsoft Treiberzertifizierung

Daniel Popovic

Betreuer: Anne Martens

Zusammenfassung Diese Arbeit behandelt die Zertifizierung von Treibern für Microsoft Betriebssysteme. Im ersten Kapitel wird die geschichtliche Entstehung und Entwicklung dieser Treiber und die Funktionsweise der Treibermodelle für MS DOS-basierende und NT-basierende Microsoft Betriebssysteme erläutert. Das folgende Kapitel befasst sich mit den Modellen und dem Ablauf der Treiberzertifizierung und zeigt den Umfang und die Komplexität dieses Prozesses anhand zweier Beispiele und der für diese Beispielhardware erforderlichen Tests auf. Das dritte Kapitel analysiert die Vor- und Nachteile der Treiberzertifizierung aus Sicht von Microsoft, der Hardware- und Treiberhersteller und der Anwender. Anschließend werden Geschäftsmodelle im Umfeld der Microsoft Treiberzertifizierung vorgestellt.

1 Einführung

Microsoft bietet seit Windows 2000 die Möglichkeit, Gerätetreiber zu zertifizieren, um Anwendern die Sicherheit zu geben, dass ein Treiber problemlos mit der entsprechenden Betriebssystem-Version arbeitet. Hardwarehersteller und Treiberentwickler können mit Hilfe einer von Microsoft zur Verfügung gestellten Testsoftware die für die Zertifizierung notwendigen Punkte testen und die Testergebnisse zur Prüfung an Microsoft senden. Nach erfolgreicher Prüfung darf der Hersteller die Logos der Microsoft Treiberzertifizierung zu Marketingzwecken verwenden, wird auf diversen Hardwarelisten von Microsoft geführt und der Treiber wird in Windows Update zur automatischen Installation aufgenommen.

2 Geschichte und Funktionsweise von Windows-Treibern

In den folgenden Unterkapiteln wird der Aufbau von Gerätetreibern kurz dargestellt. Einige der Arbeitsweisen von Treibern stammen aus den Anfangszeiten der Microsoft-Betriebssysteme. Daher behandelt das nächste Kapitel zunächst die Geschichte der Microsoft- und speziell der Windows-Treiber. Die Arbeitsweisen der Treiber in verschiedenen Microsoft Betriebssystem-Familien werden in den darauffolgenden Kapiteln beschrieben.

2.1 Geschichte der Windows-Treiber und Treiberzertifizierung

Die Geschichte der Windows-Treiber beginnt mit Einführung von MS DOS als Betriebssystem für den IBM PC im Jahr 1981. Insgesamt gibt es zwei parallele Treiberkonzepte, die in den Microsoft-Betriebssystemen zum Einsatz kommen. Die Betriebssysteme lassen sich nach dem verwendeten Treiberkonzept gruppieren. Die erste Gruppe von Betriebssystemen basiert auf MS DOS (nachfolgend als „MS DOS-Systeme“ bezeichnet). Mitglieder dieser Gruppe sind Windows 3.x, Windows 95, Windows 98 und Windows ME. Für diese Gruppe bot Microsoft mit Ausnahme von Windows ME keine Treiberzertifizierung an. Die zweite Gruppe von Betriebssystemen wurde basierend auf OS/2 entwickelt und beinhaltet Windows NT, Windows 2000, Windows XP, Windows Server 2003, Windows Vista und Windows Server 2008 (nachfolgend als „NT-Systeme“ bezeichnet). Microsoft bietet seit dem Erscheinen von Windows 2000 die Zertifizierung von Treibern an und hat hierfür im Jahr 1996 das Windows Hardware Quality Lab (WHQL [WHQ08]) eingeführt, welches für die Erstellung, Verwaltung und Durchführung von Treiberzertifizierungen verantwortlich ist.

2.2 Funktionsweise von Windows-Treibern

Die Informationen in diesem Abschnitt stammen hauptsächlich aus [One03], [Dem06] und [DN99]. Windows-Treiber können auf mehrere Arten klassifiziert werden. Im diesem Artikel wird zum einen die Klassifikation nach ihrer Aufgabe vorgenommen, wobei hier ausschließlich die Windows Driver Model (WDM) Treiber untersucht werden. Hierin enthalten sind die sogenannten Klassentreiber,

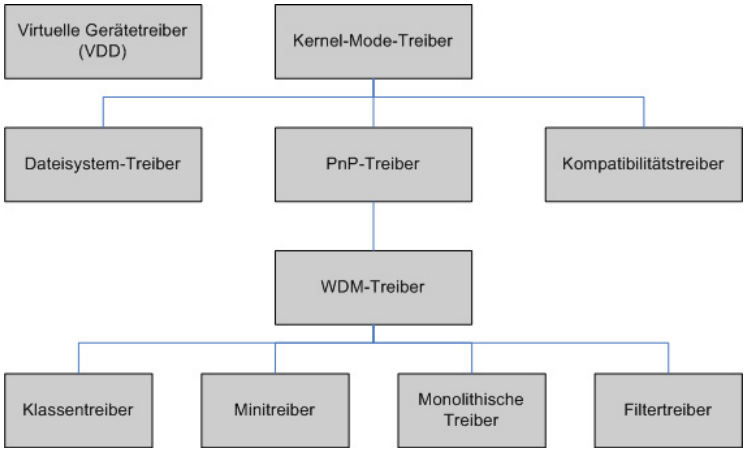


Abbildung 1. Treibertypen am Beispiels Windows XP ([One03, S. 11])

welche die grundsätzlichen Funktionen für eine Klasse von Hardwareprodukten

zur Verfügung stellen. Daneben gibt es die Minitreiber, die die Funktionalität eines Klassentreibers um herstellerspezifische Funktionen erweitern, die monolithischen Treiber, die die komplette Funktionalität zum Betrieb der Hardware implementieren, und die Filtertreiber. Diese filtern die I/O-Operationen eines Klassentreibers für eine bestimmte Hardware, um Funktionalitäten hinzuzufügen oder zu ändern. Es existieren, wie im vorigen Kapitel erwähnt, verschiedene Treiberkonzepte für MS DOS-Systeme und NT-Systeme, die im Folgenden vorgestellt werden. Zunächst werden die zum Verständnis der Treiberkonzepte relevanten Betriebsmodi der x86-kompatiblen Prozessoren beschrieben.

Real-Mode Reak-Mode, auch real address mode oder Real-Modus genannt, ist der Betriebsmodus der x86-kompatiblen Prozessoren und basiert auf der Bildung von 20-bit Speicheradressen über zwei 16-bit Register für jeden Befehl, der Speicher adressiert. In diesem Modus gibt es keinen geschützten Zugriff auf den Hauptspeicher und die Hardware, so dass jedes Programm auf den kompletten Hauptspeicher und die komplette Hardware zugreifen kann.

Protected-Mode Protected-Mode, auch geschützter Modus genannt, ist ein Betriebsmodus der x86-kompatiblen Prozessoren. Dieser seit Einführung des Intel 80286-Prozessors existierende Modus erlaubt den Zugriff auf 16 MB Hauptspeicher (32 MB Hauptspeicher seit dem Intel 80396-Prozessor) und unterstützt Segmentierung, Paging und Speicherschutz mit Hilfe von vier Schutzebenen, den sogenannten Ringen. Prozesse werden in einem dieser Ringe ausgeführt, in denen unterschiedlich Zugriffsrechte auf Speicher und Hardware gewährt werden. Dies ermöglicht die Unterscheidung von Kernel-Mode und User-Mode.

User-Mode Der User-Mode, auch Benutzer-Modus genannt, ist ein Betriebsmodus der x86-kompatiblen Prozessoren. Prozesse, die in diesem Modus laufen, werden von den verbreiteten Betriebssystemen wie Linux oder Windows in der unprivilegierten Schutzebene Ring 3 ausgeführt, in der nur eingeschränkter Zugriff auf Speicher und Hardware gewährt wird.

Kernel-Mode Der Kernel-Mode ist ein Betriebsmodus der x86-kompatiblen Prozessoren. Prozesse in diesem Modus werden in der Schutzebene Ring 0 ausgeführt, was der höchsten Privilegierungsstufe entspricht und freien Zugriff auf Speicher und Hardware erlaubt. In diesem Modus werden vor allem der Betriebssystemkern und Gerätetreiber ausgeführt.

MS DOS-Systeme: MS-DOS arbeitet nur im Real-Mode. Real-Mode-Treiber werden durch Definition und Konfiguration in der CONFIG.SYS geladen und sind in Assembler implementiert. Ihre Aufgabe liegt in der Kommunikation mit BIOS und Systemdiensten über Interrupts. Die Zuweisung von Interrupts (oder genauer gesagt: Interrupt-Adressen) musste durch den Anwender manuell vorgenommen werden, so dass es zur Vermeidung von Hardwarekonflikten notwendig war, die bereits verwendeten Interrupt-Adressen bei Inbetriebnahme neuer Hardware-Komponenten zu kennen.
Der Aufbau und die Funktionsweise von Treibern änderte sich durch die Einführung von Windows zunächst nicht. Bis zur Version Windows 95 waren sämtliche Windows-Treiber für MS DOS-Systeme als Real-Mode-Treiber implementiert.

Um Multitasking zu ermöglichen, wurde ab Windows 3.0 mit Hilfe von virtuellen Maschinen und virtuellen Gerätetreibern (VxD) ermöglicht, Hardware durch mehrere Anwendungen parallel nutzen zu lassen. Zusätzlich wurde mit Windows 3.0 die Trennung zwischen User-Mode und Kernel-Mode eingeführt, um Zugriffsschutz auf die Hardware zu erlangen. Die Treiber für Windows 95 sind ebenfalls VxD-Treiber, die als Protected-Mode-Treiber implementiert sind. Mit Windows 98 wurde als neue Treibertechnologie das Windows Driver Model eingeführt, vor allem, um Treiber für MS DOS-basierte Windows-Versionen und NT-basierte Windows-Versionen zu vereinheitlichen. Der Aufbau von Treibern für MS DOS-Systeme ist sehr verschieden, es gibt keine einheitliche Architektur. Da dieser Artikel die Treiberzertifizierung behandelt, die fast ausschließlich für die NT-Systeme eingeführt wurde, wird die Architektur der Treiber für MS DOS-Systeme an dieser Stelle nicht weiter vertieft. Ein guter, wenn auch kurzer Überblick zu diesem Thema findet sich in [One03, S. 8-10].
NT-Systeme: Die Architektur von NT-Systemen sieht eine Trennung von User-Mode und Kernel-Mode vor. Programme, die im User-Mode ablaufen, rufen für

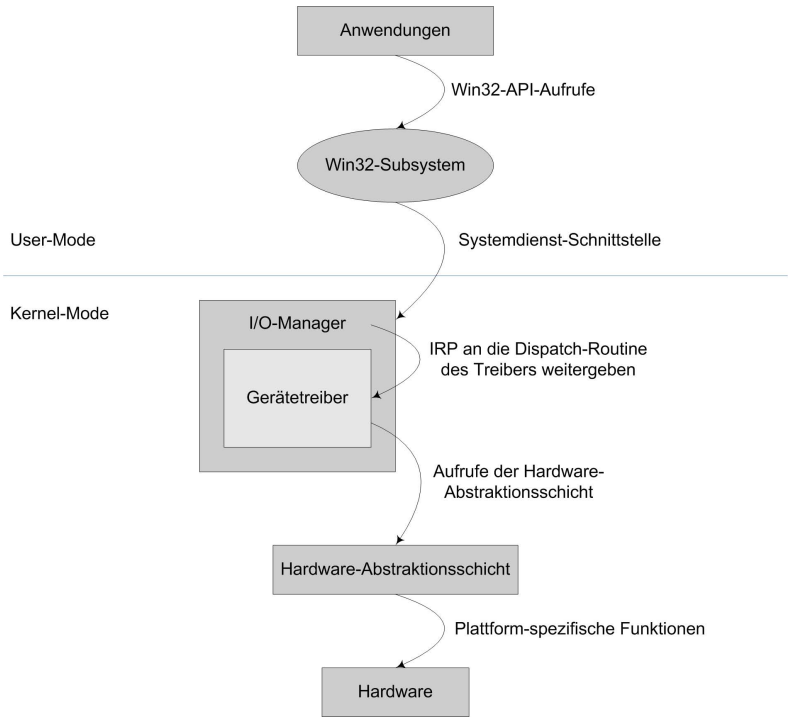


Abbildung 2. Windows XP-Treiberarchitektur ([One03, S. 6])

Hardware-Zugriffe bestimmte Win32-API-Funktionen auf, die den I/O-Manager über eine Systemdienst-Schnittstelle im Kernel-Mode aufrufen. I/O-Manager ist

die Bezeichnung für eine Menge von Diensten des Betriebssystems, die dann die Treiber mittels strukturierter sogenannter I/O-Anforderungspakete (IRP) aufrufen. Über den I-O-Manager wird auch direkter Speicherzugriff (DMA) und die Speicherung der Treiberkonfiguration in der Registrierungsdatei von Windows vorgenommen. Die Treiber wiederum greifen über eine Hardware Abstraktionsschicht (HAL) auf die Hardware zu. Dieser Prozess ist in Abbildung 2 dargestellt. Der I/O-Manager meldet die erfolgte oder noch laufende Abarbeitung des IRP an das aufrufende Programm zurück. Während der Abarbeitung des IRP kann das Programm weiterlaufen, da die Abarbeitung des IRP durch den I/O-Manager unabhängig abläuft. Die erfolgreiche Abarbeitung eines IRP wird einem wartenden Programm durch den Aufruf einer Routine eines Kernel-Mode-Dienstes angezeigt.
Seit Windows 2000 bietet Microsoft Treiberentwicklern die Möglichkeit, ihre Treiber mit einer digitalen Signatur zu versehen, um die Echtheit des Treibers zu garantieren. In der 64-bit-Version von Windows Vista ist die digitale Signatur eines Treibers, die nur durch Zertifizierung des Treibers erreicht werden kann, vorgeschrieben. Treiber ohne digitale Signatur erhalten keinen Zugriff auf den Kernel und können somit nicht geladen werden.

3 Treiberzertifizierung

In den folgenden Unterkapiteln wird genauer auf die Programme und Prozesse der Treiberzertifizierung eingegangen. Zur Veranschaulichung werden diese anhand des Betriebssystems Windows Vista beschrieben. Da die Gesamtzahl der Treibertests, die zur Zertifizierung verwendet werden, sehr hoch ist, werden zwei Beispieltreiber vorgestellt, für die eine Auswahl der geforderten Tests beschrieben wird. Anhand zweier konkreter Tests wird der Ablauf und die Zielsetzung solcher Tests aufgezeigt.

3.1 Übersicht

Die Zertifizierung eines Windows-Treibers ist ein mehrstufiger Prozess, den Abbildung 3 schematisch darstellt. Ein Hersteller, der seine Treiber zertifizieren lassen möchte, muss zunächst einige Vorbereitungen treffen. Zunächst wird ein digitales Signaturzertifikat von Verisign für die Übermittlung der Treiber und Testergebnisse an Microsoft benötigt.

Der Zertifizierungsprozess selbst besteht aus einer Reihe von Treibertests, welche durch Microsoft bereitgestellt und durch den Hersteller durchgeführt werden müssen, der anschließenden Lieferung der Treiber und Testergebnisse an Microsoft und der Prüfung und Freigabe durch Microsoft. Bei erfolgreicher Zertifizierung kann der Hersteller an den von Microsoft angebotenen Marketingprozessen, beispielsweise der Freigabe der Zertifizierungslogos oder der Aufnahme des Treibers in Windows Update, teilnehmen. Generell gibt es zwei Zertifizierungsstufen

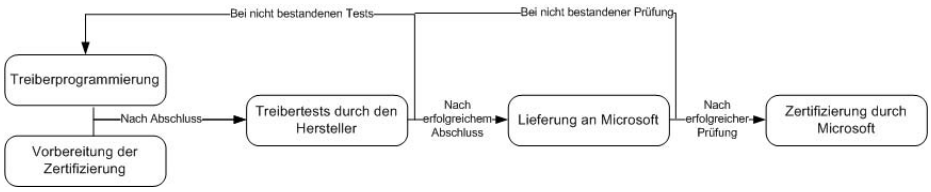


Abbildung 3. Zertifizierungsprozess

namens „Basic“ und „Premium“, deren konkrete Namen sich je nach Betriebssystem unterscheiden. Die Stufe „Basic“ soll dabei sicherstellen, dass zum Betrieb der Hardware erforderliche Grundfunktionalitäten für das getestete Betriebssystem korrekt funktionieren und die Sicherheits- und Stabilitätsanforderungen umgesetzt sind. Die Stufe „Premium“ verlangt darüber hinausgehend die korrekte und lauffähige Implementierung weiterer geräteklassenabhängiger Funktionalitäten. In den folgenden Abschnitten wird der Prozess der Treiberzertifizierung am Beispiel von Windows Vista vorgestellt.

3.2 Programme zur Treiberzertifizierung am Beispiel Windos Vista

Die Zertifizierung von Windows Vista-Treibern läuft unter dem Namen „Windows Vista Logo Program“. Sie sind unterteilt in das Basic-Programm „Works with Windows Vista“ und das Premium-Programm „Certified for Windows Vista“, die jeweils durch ein eigenes Logo ausgezeichnet werden ([WLP08]). Hardware und somit auch die zugehörigen Treiber müssen für das „Works with Windows Vista“ Programm grundsätzliche Funktionen ihrer Geräteklasse unterstützen. Eine Maus muss beispielsweise bei Bewegung der Maus in eine Richtung den Mauszeiger in dieselbe Richtung bewegen.
Für das „Certified for Windows Vista“ Programm sind weitergehende Funktionen definiert, die Hardware einer bestimmten Geräteklasse unterstützen muss ([Tul07]). So muss eine Maus unter anderem einen Scroll-Mechanismus, zum Beispiel in Form eines Scrollrads oder Touchpads, unterstützen.
Beiden Programmen ist gemeinsam, dass bestimmte Sicherheitsbestimmungen eingehalten werden müssen, um die Stabilität des Betriebssystems und den fehlerfreien Betrieb anderer optionaler Hardware zu gewährleisten. Dies wird neben anderen Punkten während des Treibertests geprüft. Ebenso gibt es bei beiden Programmen Vorschriften bezüglich der Treiberinstallation, um dem Anwender diese zu erleichtern.

3.3 Ablauf der Treiberzertifizierung am Beispiel Windows Vista

Treibertests durch den Hersteller Treibertests müssen von Herstellern selbst durchgeführt werden. Microsoft stellt Treiberentwicklern hierfür im „Windows Logo Kit“ (WLK) ein Softwarepaket namens „Driver Test Manager“ (DTM)

zur Verfügung, welches umfangreiche Treibertests für die verschiedenen Geräteklassen, also Typen von Hardware, in Form eines Assistenten beinhaltet. Die konkreten Tests, die für eine bestimmte Hardware durchgeführt werden müssen, hängen von ihrer Geräteklasse und dem Bus, über den die Hardware mit dem Computer kommuniziert, ab. Die folgenden Geräteklassen werden dabei unterschieden:
Audio (u.a. Soundkarten, Effektfilter, Synthesizer)
Bluetooth
Bus controllers (u.a. PCMCIA Controller, IEEE 1394 Controller, IrDA Controller, USB Controller, USB Hubs)
Display
Driver Quality (Antivirus-Treiber)
Imaging (u.a. Digitalkameras, Scanner, Drucker)
Input and HID (u.a. Tastaturen, Mäuse, Trackballs, Joysticks, Gamepads, Smartcard-Leser)
Modems (analoge, kabellose und ISDN Modems)
Network devices (u.a. LAN-Adapter, WLAN-Adapter, ATM-Adapter)
Storage controllers and devices (u.a. CD-Laufwerke, DVD-Laufwerke, Festplatten, Bandlaufwerke, RAID-Controller, SCSI-Controller)
Streaming media and broadcast (u.a. Videobearbeitung, DVD-Spieler)
Systems (u.a. Mainboards, Cluster)
Unclassified (für Geräte, die in keine andere Geräteklasse passen)
Der DTM beinhaltet sowohl manuelle als auch automatische Tests. Manuelle Tests erfordern Interaktionen durch einen Benutzer, also beispielsweise das Bewegen einer Maus, automatische Tests laufen nach dem Starten vollständig ohne Benutzerinteraktion. Für die Durchführung einiger Tests wird, ebenfalls abhängig von der Geräteklasse der zu testenden Hardware, zusätzliche Hardware benötigt. So müssen für den Kompatibilitätstest einer USB-Maus ein USB-Joystick, ein USB-Scanner oder USB-Drucker, eine USB-Tastatur und USB-Lautsprecher zur Verfügung stehen.
Neben der Einteilung in manuelle und automatische Tests können die Tests in statische und dynamische Tests unterteilt werden. Statische Tests prüfen den Code von Treibern, ohne dass die Treiber ausgeführt werden, während dynamische Tests den Treibercode während der Ausführung untersuchen. Dies geschieht, indem Aufrufe von Betriebssystemfunktionen durch einen Treiber an spezielle simulierte Funktionen zur Fehlersuche des Treibertests umgeleitet werden. Statische Tests arbeiten sehr genau und finden durch die Prüfung des gesamten Codes auch Fehler in Funktionen, die in der Praxis selten oder gar nicht zum Einsatz kommen, wobei jedoch auch falsch positive oder falsch negative Ergebnisse vorkommen können. Dynamische Tests haben dagegen ein sehr geringe Quote von falsch positiven Fehlermeldungen und finden Fehler, die nur zur Laufzeit des Treibers auftreten und somit von statischen Tests nicht gefunden werden können. Da jedoch natürlich nur Fehler in getesteten Bereichen des Treibers gefunden werden können, hängt die Vollständigkeit der gefundenen Fehler von der Abdeckung der Treiberfunktionen durch den dynamischen Test ab.

Die Tests sind in folgende Klassen eingeteilt. Jede Klasse enthält dabei mehrere Tests, die in Abhängigkeit des konkreten Geräts und des zu testenden Betriebssystems vorgeschrieben sind ([MSD08]).
Audio Device Testing: Diese Tests prüfen die korrekte Funktionalität von Audiogeräten.
Bus and Controller Testing: Die Tests dieser Testklasse werden zusätzlich zu den funktionsspezifischen Tests, also beispielsweise „Audio Device Testing“ für Audiogeräte, für den jeweils verwendeten Bustypen eines Geräts durchgeführt. Diese Testklasse ist in folgende Unterklassen unterteilt: 1394 Controller Testing, Bluetooth Controller Testing, CardBus-PCMCIA Testing, Secure Digital Host Controller Testing, USB Controller Testing und USB Hub Testing.
Display Testing: Diese Tests werden für Anzeigegeräte, also beispielsweise Monitore, verwendet. Die Testklasse ist infolgende Unterklassen unterteilt: Display Adapter and Chipset Testing, CRT Display Testing, Flat Panel Display Testing, Plasma Display Testing, Projector Display Testing und Windows SideShow Display Testing.
Game Devices Testing: Diese Tests werden für Spielcontroller eingesetzt. Die Testklasse ist weiter unterteilt in Common Controller Testing und USB Gaming Device Testing.
Imaging Testing: Diese Tests werden für Scanner verwendet.
Input Testing: Diese Tests werden für Eingabegeräte verwendet. Die Testklasse ist weiter unterteilt in Digitizer Testing, Keyboard Testing, Media Center IR Receiver Testing, Pointing Drawing Testing und Smart Card Reader Testing.
Network Controller Testing: Diese Tests werden für Netzwerkgeräte eingesetzt. Die Testklasse ist in folgende Unterklassen unterteilt: Lightweight Filter Testing, IrDA Device Testing, Modem Testing, LAN (Ethernet) Testing, Router Testing, Wireless LAN (802.11) Testing, Wireless Router Testing und EAP Certification Program (EAP) Testing.
Terminal Server PNP Redirection Testing: Diese Tests werden verwendet, um die Benutzung von Windows Portable Device (WPD)-Geräten über Terminal Server zu testen.
Portable Media Devices Testing: Diese Tests werden zur Prüfung tragbarer Geräte, also beispielsweise Digitalkameras oder Handys, eingesetzt.
Printer Testing: Diese Tests werden für die Prüfung von Druckern verwendet.
Storage Testing: Diese Tests dienen zur Prüfung von Speichermedien. Die Testklasse ist unterteilt in Adapter or Controller Testing, Optical Disk Drive (ODD) Testing, Hard Disk Drive (HDD) Testing, Medium Changer Testing, RAID Testing, Removable Storage Testing, Tape Drive Testing und Network Attached Storage Testing.
Streaming Media and Broadcast Testing: Diese Tests werden für Geräte und Controller zur Videoaufnahme, Videoempfang und Videodekodierung verwendet. Dabei werden die Unterklassen DVD Decoder Testing, Broadcast Received Testing, Video Capture Testing.
Daneben existieren **Treiber-Zuverlässigkeitstests, Installationstests und Systemtests**. Die Systemtests werden zum Testen eines gesamten Computer-

systems verwendet und sind nach Microsoft klassifiziert nach Qualifikationsebene des Logo-Programms (Basic oder Premium), Marktsegment (Business oder Consumer) und Systemart (Desktop, Mobile, Motherboard, Ultra-Mobile und Ultra-Portable) ([MSD08]). Im folgenden werden anhand von zwei Beispielen einige der zur Verfügung stehenden Test verschiedener Testklassen vorgestellt. Die genannten Tests und Beschreibungen stammen aus der Microsoft Developer Network Bibliothek ([MSD08]). Für einige Tests ist in der Beschreibung eine Laufzeit angegeben. Diese Laufzeit bezieht sich nur auf die Ausführung des Tests, nicht auf Vorbereitung des Tests oder Auswertung der Testergebnisse. Im ersten Beispiel wird eine USB-Maus behandelt, für welche unter anderem die folgenden Tests vorgesehen sind:

ACPI Stress Test (automatischer Test): Das Adcanced Configuration and Power Interface (ACPI) ist eine Schnittstelle für die Erkennung, die Konfiguration und das Power Management von Hardware. Dieser Test prüft, ob die zu testende Hardware und deren Treiber nach einem Computerstart aus dem Standbymodus oder Ruhezustand noch läuft und korrekt funktioniert.

ChkINF Test (automatischer Test): Dieser Test prüft mittels eines PERL-Skripts die Struktur und die Syntax der INF-Dateien eines Treibers. INF-Dateien enthalten Setup-Informationen und Konfigurationen des Treibers.

Disable Enable with IO (automatischer Test): Dieser Test prüft, ob das Gerät deaktiviert und aktiviert werden kann und anschließend fehlerfrei arbeitet.

Driver Verifier Test (manueller Test): Mit diesem Test wird geprüft, wie der Treiber mit Systemressourcen umgeht. Unter anderem wird dabei das Verhalten des Treibers in Bezug auf Speicherverwaltung, Interruptbehandlung, Deadlocks und fehlende benötigte Ressourcen getestet.

Mouse Functionality Test (manueller Test): Der Maus-Funktionstest ist eine Art Kalibrierung der Maus. Es werden sowohl die korrekten Bewegungen in alle Richtungen als auch die Funktion der Maustasten und des Scrollrads geprüft. Hierin enthalten ist die Prüfung des Formats der Daten, die der Treiber an das Betriebssystem übergibt.

Public Import (manueller Test): Dieser Test prüft, ob der Treiber veraltete APIs oder gesperrte Kernel-Mode-Funktionen aufruft.

USB-Tests (sowohl manuelle als auch automatische Tests): Hier wird in mehreren Tests die USB-Implementierung des Treibers geprüft, unter anderem durch durch den „USB Specification Compliance Test“, der die korrekte Umsetzung der entsprechenden USB-Spezifikation prüft, den „USB Address Description Test“, der prüft, ob das USB-Gerät über jede mögliche USB-Adresse (0 bis 127) korrekt angesprochen werden kann und über die korrekte USB-Adresse antwortet, und den „USB Enumeration Stress Test“, der das erstmalige Anschließen des USB-Geräts simuliert und testet. Detaillierte Angaben zu den USB-Tests folgen für das zweite Beispiel.

Das zweite Beispiel behandelt einen Microcontroller für USB-Lautsprecher:

AC-3 Test (automatischer Test): Dieser Test prüft, ob der Treiber das AC-3-Format korrekt umsetzt und AC-3-Audiostreaming über Microsoft Direct-

Sound innerhalb vorgegebener Latenzzeiten unterstützt. Die Latenzzeiten hängen vom Bus ab, so besteht für USB 1.1 eine höhere Latenzzeit als für USB 2.0.

Digital Rights Management Test (Vista) (automatischer Test): Dieser Test erzeugt Audiostreams mit verschiedenen Kopierschutzstufen, um zu prüfen, ob ein Treiber sämtliche möglichen Einstellungen des Digital Rights Management unterstützt.

Fidelity Test (automatischer Test): Dieser Test prüft mit Hilfe eines erzeugten Tons die Qualität der Audioausgabe, unter anderem durch Prüfung der maximalen elektrischen Spannung bei der Wiedergabe und Messung des Hintergrundrauschens. *Laufzeit: 15 Minuten.*

Full Duplex Test (automatischer Test): Mit diesem Test wird überprüft, ob gleichzeitige Aufnahme und Wiedergabe von Audiodaten möglich ist. *Laufzeit: 20 Minuten.*

Audio KS Position Test (automatischer Test): Dieser Test prüft, ob ein Audiogerät die korrekte Wiedergabeposition in Audiostreams erkennt. *Laufzeit: 10 Minuten.*

KS Topology Test (automatischer Test): Mit diesem Test wird die korrekte und den Microsoft-Vorgaben entsprechende Implementierung der Topologiefilter eines Audiogeräts verifiziert. Topologiefilter sind Steuerelemente, durch die Audiodaten empfangen, gesendet und gefiltert werden, beispielsweise CD Audio oder Wave-Ausgang. Jeder dieser Filter ist mit einem hardwareseitigen Anschluß verbunden (sogenannter Pin), es existiert beispielsweise ein Pin zum Anschließen des analogen Audioausgangs eines CD-ROM-Laufwerks.

Audio Power Management (Lullaby) (automatischer Test): Dieser Test spielt Audiodaten über das zu testende Gerät ab, während das System in Ruhezustand oder Standbymodus versetzt wird.

MIDI Driver Test (automatischer Test): Für die Durchführung dieses Test muss ein MIDI-fähiges Gerät zur Verfügung stehen, welches in einer Prüfschleife angeschlossen wird. Im Laufe des Tests werden mehrere Testfälle zur Prüfung der einzelnen MIDI-Befehle durchgeführt. *Laufzeit: 10 Minuten.*

Audio SysFx Test (automatischer Test): Dieser Test überprüft die Implementierung der System Effect (SysFx) Audio Processing Objects (APO). Diese stellen digitale Signalprozessoren zur digitalen Bearbeitung analoger Audiosignale zur Verfügung.

SysFx UI Test (automatischer Test): Dieser Test prüft mit Hilfe eines Skripts, ob in der Benutzerschnittstelle zur Konfiguration des Audiogeräts jeder physikalische Ein- und Ausgang per Checkbox aktiviert und deaktiviert werden kann. *Laufzeit: 5 Minuten.*

UAA Test (automatischer Test): Dieser Test ist für HDAudio-Controller vorgeschrieben. Im Test wird untersucht, ob das Gerät die Voraussetzungen erfüllt, so dass der Microsoft HDAudio Klassentreiber funktioniert. Hierbei werden verschiedene Treiber für das Gerät installiert, daher kann ein Neustart des Systems notwendig sein. Dieser Test ist daher nur bei aktiviertem automatischen Windows-Login als automatischer Test zu werten. *Laufzeit: 5 Minuten.*

USB Address Description Test (automatischer Test): Dieser Test untersucht, ob das USB-Gerät auf jede zugewiesene USB-Adresse über dieselbe Adresse antwortet. Das USB-Gerät darf dabei nicht auf Anfragen über Adressen reagieren, die ihnen nicht zugewiesen sind. Der Adressraum reicht von 0 bis 127. *Laufzeit: 10 Minuten.*

USB Descriptor Test (automatischer Test): Mit diesem Test wird geprüft, ob das USB-Gerät auf verschieden formulierte Anfragen nach dem Gerätedeskriptor und dem Konfigurationsdeskriptor korrekt reagiert. Der Gerätedeskriptor enthält die unterstützte USB-Version des Geräts, die Hersteller- und Produkt-ID, die für die automatische Auswahl des passenden Treibers durch das Betriebssystem benötigt werden, und die Anzahl der unterstützten Konfigurationen, die im Konfigurationsdeskriptor definiert sind. Die Konfigurationen beinhalten unter anderem Informationen zum maximalen Stromverbrauch des Gerätes in einem bestimmten Modus und zur Anzahl der Schnittstellen des Gerätes. *Laufzeit: 10 Minuten.*

USB Device Control Requests Test (automatischer Test): Dieser Test untersucht, ob USB-Geräte nach dem Zurücksetzen des Geräts korrekt auf eine definierte Anzahl von Steuerbefehlen reagiert. Die Steuerbefehle werden nach dem Zurücksetzen in verschiedenen Zeitabständen und über verschiedene USB-Adressen geschickt. *Laufzeit: 7 Minuten.*

USB Device Framework (CV) Test (automatischer Test): Mit diesem Test wird die korrekte Umsetzung des „USB Device Framework“ Befehlssatz der USB Spezifikation in der Version 2.0 geprüft. Dabei werden an den Treiber verschiedene Geräte-, Konfigurations- und Schnittstellenbeschreibungsbefehle gesendet. Für diesen Test muss das USB-Gerät angeschlossen sein. *Laufzeit: 5 Minuten.*

USB Enumeration Stress Test (automatischer Test): Dieser Test prüft die richtige Funktion des USB-Geräts nach seinem erstmaligen Anschließen. Hierzu muss das Gerät gültige Informationen über seine Steuerleitung senden und auf alle Standardgeräteanfragen korrekt antworten. Das USB-Gerät muss für diesen Test angeschlossen sein. *Laufzeit: 10 Minuten.*

USB HIDView (CV) Test (automatischer Test): Dieser Test prüft, ob das Gerät und der Treiber die Spezifikation für USB Human Input Devices (HID) korrekt umsetzen. Für diesen Test muss die zugehörige Hardware über USB 2.0 angeschlossen sein. *Laufzeit: 5 Minuten.*

USB Isochronous Alternate Interface Presence (automatischer Test): Dieser Test stellt sicher, dass USB-Geräte, die isochrone Datenübertragung benötigen, also Datentransfers gleicher Periodendauer, verschiedene Konfigurationen unterstützen, um die Isochronität gewährleisten zu können. Ist das USB-Gerät bei diesem Test nicht angeschlossen, wird der Test als nicht bestanden gewertet. *Laufzeit: 5 Minuten.*

USB Selective Suspend (automatischer Test): Mit diesem Test wird geprüft, ob das Gerät die Funktionen „Unterbrechen“ und „Fortsetzen“ korrekt unterstützt. Für USB-Geräte, die diese Funktion unterstützen, wird dabei bei Nichtbenutzung der Upstream-Port abgeschaltet, um Strom zu sparen. Nach dem Fortsetzen, also dem Anschalten des Upstream-Ports, muss das Gerät

korrekt funktionieren und auf Anfragen anderer Geräte reagieren. *Laufzeit: 10 Minuten.*

USB Serial Number Test (automatischer Test): Die Verwendung einer Seriennummer ist für USB-Geräte optional. Falls eine Seriennummer vergeben wird, muss diese eindeutig sein. Dieser Test prüft bei Vorhandensein einer Seriennummer, ob diese eindeutig ist. Für diesen Test ist es notwendig, zwei Geräte mit unterschiedlichen Seriennummern an den Testcomputer anzuschließen. Falls keine Seriennummer verwendet wird, gilt dieser Test als bestanden. *Laufzeit: 5 Minuten.*

USB Specification Compliance (automatischer Test): Dieser Test untersucht, ob das Gerät die USB-Spezifikation 1.1 oder die USB-Spezifikation 2.0 korrekt umsetzt. Dabei wird durch spezielle Anfragen ermittelt, welche USB-Spezifikation das Gerät maximal unterstützt. *Laufzeit: 3 Minuten.*

Wave Test (automatischer Test): Dieser Test prüft das korrekte Aufzeichnen und Abspielen von PCM-kodierten Audiodaten. PCM ist ein Audiocodec, der auch im Audio-CD- und Wave-Format verwendet wird. *Laufzeit: 10 Minuten.*

System - Common Scenario Stress with IO: Dieser Test stellt sicher, dass das Computersystem, an dem das zu testende Gerät angeschlossen ist, und das Gerät selbst Plug and Play und Power Management-Statusänderungen annehmen und korrekt behandeln. Zu diesen Statusänderungen gehören das Deaktivieren, das Aktivieren, das Unterbrechen (Suspend) und Reaktivieren (Wake) des Geräts. Der Test prüft ebenso, ob das System und das Gerät weiterhin korrekt funktionieren. *Laufzeit: 180 Minuten.*

Device Path Exerciser: Mit diesem Test wird die Stabilität und Zuverlässigkeit des Treibers untersucht. Hierzu werden verschiedene Funktionen des Treibers mit gültigen, ungültigen, sinnlosen und falsch formatierten Daten aufgerufen. Diese Aufrufe führen zum Absturz des Treibers, falls die fehlerhaften Aufrufe nicht korrekt abgefangen werden.

System - Disable Enable with IO: Dieser Test deaktiviert und aktiviert jedes im System installierte deaktivierbare Gerät. Anschließend wird geprüft, ob alle im Laufe des Tests deaktivierten Geräte korrekt funktionieren. *Laufzeit: 120 Minuten.*

System - Plug and Play Driver Test (automatischer Test): Dieser Test führt verschiedene Plug and Play-Operationen, die vom zu testenden Gerät und Treiber unterstützt werden müssen, aus. Dies sind unter anderem das Entfernen, ein doppelter Start oder unerlaubtes Entfernen des Geräts.

PREfast for Drivers: Hierbei handelt es sich um ein statisches Verifizierungswerkzeug, welches während des Kompilierens von C und C++-Code einfache Fehler findet. Die Fehlersuche wird vor allem für Kernel-Mode-Treibercode ausgeführt.

Run INFTest against a single INF: Dieser Test ist ein automatisierter Job, der den im ersten Beispiel vorgestellten ChkINF Test ausführt.

System - Sleep Stress with IO (automatischer Test): Dieser Test prüft, ob das Computersystem, an dem das zu testende Gerät angeschlossen ist, und das Gerät selbst nach Durchlaufen der möglichen Ruhezustände, die das Betriebssystem unterstützt, korrekt arbeiten. *Laufzeit: 40 Minuten.*

Das Testprotokoll für dieses Beispiel ist verfügbar in [Hol07].
Im Folgenden werden zwei der oben genannten Tests im Detail vorgestellt, um die Funktionsweise und den Ablauf solcher Tests zu verdeutlichen.

USB Address Description Test: Der USB Address Description Test besteht aus mehreren unabhängigen Tests und basiert auf dem USB Command Verifier-Testwerkzeug (USBCV) der USB Implementers Forum, Inc. ([USB08]), welches die Übereinstimmung eines USB-Geräts mit den Spezifikationen für USB Device Framework, Hub-Geräteklassen, HID-Geräteklassen und weiteren Spezifikationen überprüft. Die Einrichtung und Vorbereitung des USB Address Description Test besteht darin, das zu testende Gerät über einen USB-Hub mit maximal 7 USB-Ports direkt an den Testcomputer anzuschließen. Für die Initialisierung des Tests muss das zu testende USB-Gerät ausgewählt werden, anschließend werden der Gerätedeskriptor, der Konfigurationsdeskriptor und eventuell vorhandene Stringdeskriptoren, welche natürlichsprachliche Informationen bereitstellen, ausgelesen. Der Test selbst besteht aus zwei Adresstests, die jeweils für sämtliche mögliche USB-Adressen von 0 bis 127 durchgeführt werden.
Der erste Test besteht aus folgenden Schritten:

1. Zurücksetzen des USB-Geräts
2. Abfragen der Deskriptoren (Geräte-, Konfigurations- und Stringdeskriptoren) über USB-Adresse 0
3. Setzen der USB-Adresse auf die nächste freie USB-Adresse
4. Abfragen der Deskriptoren über USB-Adresse 0 (das Gerät darf nicht antworten)
5. Abfragen der Deskriptoren über die in Schritt 3 gesetzte neue USB-Adresse (das Gerät muss antworten)
6. Vergleichen der zurückgelieferten Deskriptoren mit den in der Initialisierung des Tests abgefragten Deskriptoren

Der zweite Test besteht aus folgenden Schritten:

1. Zurücksetzen des USB-Geräts (nur im ersten Durchlauf des Tests)
2. Setzen der USB-Adresse auf die nächste freie USB-Adresse
3. Abfragen der Deskriptoren über die vorige USB-Adresse (das Gerät darf nicht antworten)
4. Abfragen der Deskriptoren über die in Schritt 2 gesetzte neue USB-Adresse (das Gerät muss antworten)
5. Vergleichen der zurückgelieferten Deskriptoren mit den in der Initialisierung des Tests abgefragten Deskriptoren

Zum Bestehen des USB Address Description Tests muss das USB-Gerät auf alle Anfragen über gültige USB-Adressen mit den korrekten Geräte-, Konfigurations- und Stringdeskriptoren antworten und darf nicht über die vorherigen USB-Adressen antworten. Bei einem Fehler gilt der Test als nicht bestanden. Der Test gilt ebenfalls als nicht bestanden, falls das USB-Gerät nicht in der durch die USB-Spezifikation vorgegebenen Reaktionszeit antwortet.

Full Duplex Test: Der Full Duplex Test prüft die gleichzeitige Aufnahme und Wiedergabe von Audiodaten über verschiedene Kombinationen von Eingabe- und Ausgabekanälen. Der Test ermittelt die zur Verfügung stehenden Endpunkte, also Ein- und Ausgänge eines Audiogeräts, findet die passenden und möglichen Paare von Ein- und Ausgängen durch Erzeugen eines Sinustons auf jedem möglichen Ausgang und Messen der Eingangssignale und führt eine automatische Prüfung jedes Paares durch. Zur Durchführung des Tests sind unter anderem ein Computer mit mindestens 20 GB freiem Festplattenspeicher auf der Partition C: und ein Audiokabel zur Verbindung von LineOut und LineIn oder LineOut und Mikrofon-Eingang notwendig. Der Test verwendet für die Aufnahme und Wiedergabe die drei APIs Microsoft DirectSound, Wave In/Out und Kernel-Streaming, eine im Kernel-Mode ablaufende Technik zur Wiedergabe von Audiodaten in Echtzeit, in den folgenden Kombinationen:

1. DirectSound Wiedergabe / DirectSound Aufnahme
2. DirectSound Wiedergabe / WaveIn
3. WaveOut / WaveIn
4. WaveOut / DirectSound Aufnahme
5. Kernel-Streaming

Für jede dieser Kombinationen werden zwei Szenarien durchgespielt: Starten der Wiedergabe vor dem Starten der Aufnahme und Starten der Aufnahme vor dem Starten der Wiedergabe. Der Test selbst besteht aus verschiedenen Duplex-Szenarien, die für jedes Szenario jeder Kombination durchgeführt werden. Welche Ziele dabei eingehalten werden müssen, um den Test zu bestehen, ist nicht veröffentlicht.

Lieferung an Microsoft Die Lieferung der getesteten Treiber und der Testergebnisse erfolgt in Form von Kabinettdateien, welche durch den DTM erzeugt und über eine Webseite an Microsoft übermittelt werden. Kabinettdateien sind komprimierte Pakete zusammengehörender Dateien, die in der Windows-Welt vor allem als Installationspakete verwendet werden. Auch die Windows-Installationspakete liegen auf Installationsdatenträgern als Kabinettdateien vor. Über diese Webseite kann ebenfalls der Bearbeitungsstatus bereits übermittelter Treiberpakete abgefragt und eine Liste von durch Anwender gemeldeten Fehlern eingesehen werden. Die Fehler, die in dieser Liste verzeichnet sind, werden bei Treiberabstürzen durch Windows erzeugt und in Form des bekannten „Problemberichts“ an Microsoft gesendet.
Zur Lieferung eines Treibers muss ein aktives Konto bei WHQL bestehen. Nach Auswahl des Hardwaretyps, der unterstützten Betriebssysteme und Varianten, der PnP-ID (eine von Microsoft vergebene, aus Hersteller-ID und Produkt-ID zusammengesetzte eindeutige Nummer des Geräts) und weiterer hardwareabhängiger Daten, so beispielsweise die Anzahl der Tasten einer Maus oder die unterstützte Übertragungsgeschwindigkeit eines USB-Geräts, werden pro Betriebssystem der Treiber und die Testergebnisse hochgeladen. Für einige Treibertests benötigt das WHQL auch die Hardware, die der Treiber steuert. Daher existieren

mehrere WHQL-Testlabore weltweit. Der Hersteller wählt aus einer Liste das gewünschte Labor aus. An dieses Labor ist auch eventuell benötigte Hardware zu liefern. Zum Abschluss der Lieferung muss das Paket mit der Verisign-ID digital signiert und die Zahlungsweise für die Zertifizierungsgebühren ausgewählt werden. Die Zertifizierungsgebühren betragen momentan 250 Dollar pro Betriebssystem.

Prüfung durch Microsoft Nach der Lieferung eines Treibers und der zugehörigen Testergebnisse prüft Microsoft diese und führt gegebenenfalls selbst Tests durch. In welchem Umfang das geschieht, ob weitere Tests durchgeführt werden, die über die Herstellertests hinaus gehen, und der Zeitrahmen, den Microsoft für die Prüfung veranschlagt, ist nicht bekannt.

Schritte nach der Zertifizierung Mit erfolgreicher Zertifizierung eines Treibers erhält der Hersteller das Recht, das Logo der entsprechenden Zertifizierungsstufe (Basic oder Premium) der getesteten Betriebssysteme auf Produktverpackungen, in Produktdokumenten, in der Werbung und in der Software zu verwenden. Das zertifizierte Produkt wird in verschiedenen von Microsoft veröffentlichten Listen präsentiert. Im einzelnen sind dies der Windows Marketplace (ein von Microsoft betriebener Online-Shop für zertifizierte Hardware und Software [WMP08]), der Windows Server Catalog (eine von Microsoft betriebene Online-Datenbank mit Details zu zertifizierter Server-Hardware und angeschlossenem Online-Shop [WSC08]), die Liste getesteter Windows-Produkte (Tested Products List, TPL) und die Liste der zertifizierten Hardware (Hardware Compatibility List, HCL [WHC08]). In den Listen wird neben Produktdetails auch die erreichte Zertifizierungsstufe gezeigt.
Der Treiber wird auf Wunsch des Herstellers in das Windows Update-Programm aufgenommen. Über Windows Update werden die Anwender dann automatisch mit Betriebssystem-Updates und Treibern für installierte Hardware versorgt. Falls kein zertifizierter Treiber des Herstellers dort vorhanden ist, installiert Windows nach einem vorgegebenen Algorithmus einen anderen, eventuell unzertifizierten Treiber des Herstellers oder - falls vorhanden - den von Microsoft gelieferten Klassentreiber für ein Gerät ([Tul07, S. 556]). Microsofts Klassentreiber implementieren meist nur die grundlegenden Funktionen eines Geräts, so dass der Funktionsumfang stark eingeschränkt sein kann. Steht ein zertifizierter Treiber über Windows Update für das Gerät zur Verfügung, so wird dieser installiert. Das Gerät wird bei der Installation über die PnP-ID vom Betriebssystem identifiziert. Da die PnP-ID auch bei der Treiberlieferung an Microsoft mit angegeben werden muss, kann Windows Update somit automatisch den passenden Treiber finden und installieren.
Ein zertifizierter Treiber erhält von Microsoft eine digitale Signatur. Bei der Installation des Treibers prüft Windows, ob diese digitale Signatur vorhanden ist. Falls nicht, wird der Anwender in einem Hinweisdialog darüber informiert, dass für den Treiber der Windows Logo Test nicht durchgeführt wurde, und dieser muss die Installation des Treibers manuell bestätigen. Interessanterweise wur-

de in deutschen Windows-Versionen seit Windows 2000 die warnende Wirkung dieser Meldung noch verstärkt, indem das englische Wort „pass“ in der Warnmeldung „The software you are installing for this hardware: xxx has not passed Windows Logo testing to verify its compatibility with Windows XP.“ anstatt als „durchführen“ fälschlicherweise als „bestanden“ übersetzt wurde.

4 Vorteile und Nachteile der Treiberzertifizierung

Die Treiberzertifizierung hat für Microsoft, die Hardware- und Treiberhersteller und die Anwender Vor- und Nachteile. In den folgenden Kapiteln werden diese für die Genannten einzeln dargelegt.

4.1 Vorteile und Nachteile für Microsoft

Der offensichtlichste Vorteil der Treiberzertifizierung für Microsoft besteht darin, dass durch geprüfte und somit zertifizierte Treiber die *Stabilität des Betriebssystems* erhöht wird. Da Treiber in aktuellen Windows-Versionen größtenteils im Kernel-Mode laufen, haben sie vollen Zugriff auf die Hardware und auf Betriebssystemdienste und -funktionen, so dass es bei Treiberfehlern zu Systemabstürzen und sogar zu Hardwareschäden kommen kann. Daher ist es besonders wichtig, dass Treiber möglichst fehlerfrei implementiert sind. Die Treiberzertifizierung war jedoch bisher nicht von Microsoft vorgeschrieben, so dass viele Treiber aus Zeit- und Kostengründen nicht zertifiziert sind. Erst die 64-bit Version von Windows Vista verlangt zwingend die digitale Signatur eines Treibers, die dieser bei erfolgreicher Zertifizierung von Microsoft erhält.
Ein weiterer, vergleichsweise kleiner Vorteil für Microsoft liegt darin, dass durch die Zertifizierung *Umsatz* generiert wird. Microsoft betreibt verschiedene Online-Shops, in denen zertifizierte Hardware verkauft wird. Daneben erhält Microsoft 250 Dollar pro versuchtem Treibertest für jedes überprüfte Betriebssystem (bei 28.000 gelisteten Geräten im Windows Server Katalog ergibt dies mindestens 70 Mio. Dollar Umsatz).

4.2 Vorteile und Nachteile für Hardwarehersteller

Die Vorteile für Hardwarehersteller liegen zum einen in der erhöhten *Zuverlässigkeit der eigenen Treiber* und zum anderen in der Unterstützung durch Microsoft. Die Tests, die für die Treiberzertifizierung durchgeführt werden müssen, werden von Microsoft zur Verfügung gestellt und bieten eine gute *Möglichkeit zur Qualitätssicherung*. Nach erfolgreicher Zertifizierung erhalten Hardwarehersteller *Zugang zu marketingunterstützenden Programmen von Microsoft*. So dürfen die Logos der erreichten Zertifizierungsstufe „Basic“ oder „Premium“ auf Produktverpackungen, in der Software und in Marketingmaterialien verwendet werden, die zertifizierte Hardware wird in diverse Listen und Shops auf Microsoft Webseiten und der Treiber auf Herstellerwunsch in Windows Update aufgenommen, so dass er automatisch vom Betriebssystem ausgewählt und installiert werden

kann.
Als Nachteile kann man den *hohen Zeit- und Kostenaufwand* nennen. Die Treibertests und der weitere Zertifizierungsprozess sind zeitintensiv und erfordern teilweise zusätzliche Hardware, die nur für die Treibertests benötigt werden. Für die Durchführung der seit Windows Vista vorgeschriebenen DTM-Tests ist ein Testlabor mit einem mit Windows Server 2003 betriebenen DTM-Controller, mindestens ein DTM-Client-Computer und mindestens ein weiterer Computer zur Aufzeichnung der Testergebnisse notwendig. Daher haben sich einige Unternehmen, die über die notwendige Hardware und Erfahrung im Zertifizierungsprozess verfügen, als Dienstleister auf die Durchführung von Treiberzertifizierungen spezialisiert. Kapitel 5 enthält Details zu diesen Geschäftsmodellen.

4.3 Vorteile und Nachteile für Anwender

Die Treiberzertifizierung *erleichert den Installationsprozess* neuer Hardware für Anwender, da ein zertifizierter *Treiber über Windows Update automatisch ausgewählt* wird, falls der Hardwarehersteller an Windows Update teilnimmt. Damit entfällt die Treiberinstallation von beigelegten CDs oder die manuelle Suche nach einem passenden Treiber auf der Homepage des Hardwareherstellers. Durch die Zertifizierung erhalten Anwender zudem die Sicherheit, dass Treiber getestet und im Allgemeinen fehlerfrei sind, was die *Systemstabilität erhöht* und somit die Gefahr von Schäden an Hard- und Software reduziert.
Nachteilig für Anwender ist, dass die Zertifizierung bisher - mit Ausnahme der 64-bit-Version von Windows Vista - bisher *nicht von Microsoft vorgeschrieben* ist. Es kann also vorkommen, dass nicht ausreichend getestete und eventuell feherlhafte Treiber installiert werden. Die Sicherheitswarnung, die Microsoft bei der Installation nicht zertifizierter Treiber ausgibt, kann vor allem von unerfahrenen Anwendern so verstanden werden, dass der Treiber bei einem Test durchgefallen ist und daher nicht funktioniert, was für *Verunsicherung* sorgt. Entsprechende Fragen sind in zahlreichen Internet-Foren zu finden (ein Beispiel in findet sich unter [Can08]).

5 Geschäftsmodelle im Rahmen der Microsoft Treiberzertifizierung

Im Umfeld der Microsoft Treiberzertifizierung haben sich zahlreiche Unternehmen etabliert, die verschiedene Dienstleistungen von der Beratung bei der Treiberentwicklung oder im Zertifizierungsprozess bis hin zur kompletten Durchführung der Treiberzertifizierung anbieten.
Die Programmierung von Treibern ist ein komplizierter Prozess, die von den Programmierern besondere Kenntnisse und Erfahrung verlangt. Die Spezifikationen, die für eine erfolgreiche Zertifizierung eines Treibers zu erfüllen sind, sind teilweise sehr umfangreich, und müssen bereits während der Programmierung beachtet werden. Daher kann es insbesondere für kleine Hardware- oder Treiberhersteller, die nicht über das nötige Wissen verfügen, sinnvoll sein, mit spezialisierten

externen Beratern oder Programmierern bei der Treiberentwicklung zusammenzuarbeiten. Ein Unternehmen, das die Entwicklung von Gerätetreiber anbietet, findet sich unter [Mer08].
Die Treiberzertifizierung erfordert ebenfalls besondere Fachkenntnisse, die oftmals beim Hardware- oder Treiberhersteller nicht ausreichend vorhanden sind. Die Durchführung der Tests ist zeitaufwändig, einige der Tests dauern mehrere Stunden oder Tage, wobei jeder Test für jede unterstützte Version eines Betriebssystems durchgeführt werden muss. Außerdem besteht umfangreicher Hardwarebedarf für die Treibertests, beispielsweise verschiedenste USB-Geräte für die Kompatibilitätstests eines beliebigen USB-Geräts.
Aufgrund dieser Punkte ist es für Treiberhersteller oftmals aus Zeit- oder Kostengründen vorteilhaft, den Zertifizierungsprozess komplett auszulagern. Die Zertifizierungsunternehmen besitzen die benötigte Infrastruktur, entsprechend ausgebildetes Personal und Erfahrung in der Treiberzertifizierung und sind bereits als Mitglied bei Microsoft WHQL angemeldet, so dass neben den Tests und der Lieferung der Testergebnisee auch administrative Aufgaben wie die Beantragung der digitalen Signatur für den Treiberhersteller und die Kommunikation mit Microsoft bei aufgetretenen Fehlern entfallen. Die Kosten für die Zertifizierung eines Treibers der Geräteklasse „Unclassified“ durch ein Zertifizierungsunternehmen liegen bei ungefähr 800 Euro pro Betriebssystem, was allein schon durch die Kosten für die Verisign Class 3-Signatur in Höhe von 500 Dollar und die Kosten in Höhe von 250 Dollar für jeden an Microsoft eingereichten Test aufgewogen wird. Beispiele für Zertifizierungsunternehmen finden sich unter [HCL08], [Cra08], [SWA08] und [NTS08].

6 Zusammenfassung

Die Möglichkeit, Gerätetreiber durch Microsoft zertifizieren zu lassen, besteht für Hardwarehersteller bereits seit der Einführung von Windows 2000 im Jahr 1996. Da die Zertifizierung für die Hersteller bisher nicht zwingend vorgeschrieben ist, wird diese Möglichkeit aus Zeit- und Kostengründen selten genutzt, so dass Anwender in der Regel unzertifizierte Treiber verwenden müssen. Die irreführenden Warnhinweise, die bei der Installation eines unzertifizierten Treibers angezeigt werden, führen dabei gerade bei unerfahrenen oder computertechnisch unwissenden Anwendern zu Unsicherheit. Der größere Nachteil ist aber die Sicherheitsproblematik, die durch die Treiberzertifizierung eigentlich verbessert werden soll. Gerätetreiber haben durch ihre Funktionsweise Zugriff auf sicherheitskritische Bereiche eines Computersystems, was bei fehlerhaften Treibern zu Problemen wie Instabilität des Betriebssystems bis hin zu schweren Schäden an der Hardware führen kann. Erst mit Einführung der 64-bit Version von Windows Vista ist die Treiberzertifizierung zwingend vorgeschrieben, so dass zu hoffen ist, dass die Stabilität der Windows-Betriebssysteme in diesem Bereich zunimmt. Hierzu hat Microsoft auch die Fehlerbehandlung von Treibern verbessert. Die Behandlung von I/O-Operationen blockierter oder blockierender Treiber in Windows Vista wurde derart geändert, dass ein Abbruch der Operation und die Wiederherstel-

lung des Treibers einfacher und sicherer zu implementieren ist ([Tul07, S. 1294]). Der Prozess der Treiberzertifizierung ist komplex und kostspielig. Dies ist insbesondere für kleine Hardwarehersteller, die das nötige Fachwissen und die erforderliche Hardware nicht zur Verfügung haben, problematisch. Daher bieten einige Unternehmen die Treibertests und auch die komplette Durchführung der Zertifizierung als Dienstleistung an. Somit ist es auch für kleinere Hersteller möglich, Gerätetreiber zertifizieren zu lassen. Da die Zertifizierung von Treibern in Zukunft für sämtliche Microsoft-Betriebssysteme zur Pflicht werden könnte, ist der Markt für diese Dienstleister sicherlich noch im Wachstum.
Die Zertifizierung von Treibern ist für die Erhöhung der Sicherheit von Betriebssystemen ein wichtiger Schritt. Ob jedoch ein wirklicher Sicherheitsgewinn durch die Treiberzertifizierung vorliegt, hängt von der Qualität der Tests ab, die zur Zertifizierung durchgeführt werden. Wie in [CHI03] am Beispiel von unsicheren Netzwerkkarten-Treibern für Windows Server 2003 beschrieben ist, können auch schwerwiegende Fehler in Gerätetreibern durch nicht ausreichende oder schlecht umgesetzte Tests im Zertifizierungsprozess unentdeckt bleiben. Es hängt also von der zukünftigen Entwicklung der zur Treiberzertifizierung geforderten Tests und der Durchsetzung der Treiberzertifizierung als zwingenden Prozess in der Treiberentwicklung ab, ob die Ziele, die durch die Treiberzertifizierung erreicht werden sollen, tatsächlich erreicht werden.

Literatur

Can08. Canon drucker hat den windows logo test nicht bestanden - was nun?, 2008. http://www.spotlight.de/zforen/pca/m/pca-1202563457-21752.html.

CHI03. News - microsoft zertifiziert noch immer unsichere treiber - chip online, 2003. http://www.chip.de/news/Microsoft-zertifiziert-noch-immer-unsichere-Treiber_13684662.html.

Cra08. Craiton testing - outsource your whql, dtm and hct designed for windows logo testing, 2008. http://www.craiton.com/.

Dem06. Klaus Dembowski. *Das Addison-Wesley Handbuch der Hardwareprogrammierung*. Addison-Wesley, 2006.

DN99. Edward N. Dekker and Joseph M. Newcomer. *Developing Windows NT Device Drivers*. Addison-Wesley, 1999.

HCL08. Hcl-lab gmbh - ready for certification, 2008. http://www.hcl-lab.de/.

Hol07. Holtek Semiconductor. WHQL Test Program - Test Report for Audio Device. http://www.holtek.com.tw/english/literature/Holtek_HT82A821R_Pretest%202_Pass_Report.pdf, 2007.

Mer08. Custom windows driver development, 2008. http://www.mercdev.com/technologies/device_driver_development/.

MSD08. Msdn home page, 2008. http://msdn2.microsoft.com/en-us/default.aspx.

NTS08. Windows whql/dtm testing>nts computer and electronic products, 2008. http://www.ntscorp.com/services/computerelectronicproducts/CertificationServices/ComputerHardware/WHQLDTMTesting.

One03. Walter Oney. *Programming the Microsoft Windows Driver Model*. Microsoft Press, 2. edition edition, 2003.

SWA08. Whql precertification testing - quality test services, 2008. http://www.swatlab.com/qa_testing/whql_precert.html.

Tul07. Mitch Tulloch. *Windows Vista Resource Kit*. Microsoft Press, 2007.

USB08. Usb.org - tools, 2008. http://www.usb.org/developers/tools/.

WHC08. Products designed for microsoft windows - windows catalog and hcl, 2008. http://www.microsoft.com/whdc/hcl/default.mspx.

WHQ08. Windows quality online services, 2008. https://winqual.microsoft.com/.

WLP08. Windows logo program: Overview, 2008. http://www.microsoft.com/whdc/winlogo/default.mspx.

WMP08. Windows marketplace: Software and hardware for the windows operating system, 2008.

WSC08. Windows server catalog of tested products, 2008. http://www.windowsservercatalog.com/.

Safety-Normen

cand. inf. Alexander M. Turek

Betreuerin: Anne Martens

Zusammenfassung Die Betriebssicherheit — engl. *safety* — ist ein wichtiger Aspekt bei der Entwicklung neuer Softwaresysteme in sicherheitskritischen Bereichen. Diese Ausarbeitung wird einige Normen vorstellen, die das Vorgehen bei der Entwicklung solcher Systeme festlegen. Hierzu zählt insbesondere die generische Norm IEC 61508, die für die gesamte Systementwicklungs- und -betriebsdauer einen Lebenszyklus definiert. Als Anwendungsbereich werden Leitsysteme in Kernkraftwerken genauer betrachtet.

1 Einführung

Wenn Software in einem sicherheitskritischen Umfeld eingesetzt werden soll, genügen die üblichen Anforderungen an die Qualität von Software in der Regel nicht. Denn ein Fehler in der Software oder gar der Ausfall eines Softwaresystems kostet hierbei nicht nur viel Geld, sondern kann auch die Gesundheit oder gar das Leben von Menschen gefährden.

Die Frage, die sich hier stellt, ist: Wann ist ein System eigentlich sicher?

1.1 Sicherheit — Safety — Security

Sicherheit ist im allgemeinen zu verstehen als „Zustand, der frei von unvertretbaren Risiken der Beeinträchtigung ist oder als gefahrenfrei angesehen wird“ [Wiki07]. Für das deutsche Wort Sicherheit gibt es im Kontext von Software im Englischen 2 Entsprechungen, die klar gegeneinander abgegrenzt sind: Security und Safety.

Security ist als Schutz eines Softwaresystems vor seiner Umwelt zu verstehen. Es geht darum vorsätzliche Angriffe auf das System abzuwenden und die Integrität des Systems zu bewahren. Man kann daher auch von Angriffssicherheit oder Immunität sprechen.

Safety beschreibt den entgegengesetzten Fall. Die Umwelt, insbesondere das Leben von Menschen, wird vor den Schäden, den das System anrichten könnte geschützt. Die Ereignisse, vor denen geschützt werden soll, sind hierbei typischerweise nicht vorsätzlicher sondern vielmehr zufälliger Natur, wie beispielsweise das Fehlverhalten einer Komponente, ein technischer Defekt der zu steuernden Anlage oder ein Bedienfehler des Anwenders [Tzsc06]. Man spricht auch von Betriebssicherheit oder Isolation.

In dieser Ausarbeitung wird hauptsächlich der zweite Aspekt, die Safety behandelt.

1.2 Motivation

Wie kann man nun entscheiden, ob ein Softwaresystem sicher ist, bevor man es überhaupt einsetzt? Und woher kann man wissen, ob eine Softwarefirma dazu in der Lage ist, „sichere Software“ zu entwickeln, bevor man diese beauftragt?

Um dies möglichst objektiv zu beurteilen, bedarf es allgemein anerkannter Kriterien, auf deren Grundlage eine Softwarefirma, ihre Softwareprodukte und ihr Entwicklungsprozess geprüft und bewertet werden kann. Eine Sammlung solcher Kriterien ist eine Norm, eine Bewertung auf der Grundlage dieser Norm durch qualifizierte Prüfer nennt man Zertifizierung.

Diese Ausarbeitung wird exemplarisch einige Sicherheitsnormen der International Electrotechnical Commission (IEC) vorstellen. Kapitel 2 stellt die IEC 61508 vor, die als generische Norm in verschiedenen Domänen direkt eingesetzt oder als Grundlange für eine Sektornorm — also eine Norm, die sich auf eine bestimmt Domäne konzetriert — verwendet werden kann. Sie wurde ausgewählt, weil sie einen abstrahierten und gesamtheitlichen Ansatz verfolgt, um in möglichst vielen Domänen angewendet werden zu können. Anschließend befasst sich Kapitel 3 mit den Normen, die die 61508 für Kernkraftwerke umsetzen. Diese sind insbesondere die IEC 61513 und die 60880. Letztere befasst sich mit den Anforderungen an Software für diejenigen Systeme, die für Kernkraftwerke als besonders sicherheitskritisch eingestuft werden.

2 IEC 61508

Die IEC 61508 trägt den Titel *Funktionale Sicherheit sicherheitsbezogener elektrischer/elektronischer/programmierbarer elektronischer Systeme* und versteht sich — wie ihr Name schon sagt — als generische Norm für sicherheitsbezogene elektrische/elektronische/programmierbare elektronische (E/E/PE) Systeme. Sie geht aus der Arbeit einer 1985 von der *International Electrotechnical Commission (IEC)* gegründeten Arbeitsgruppe hervor. Sie besteht aus 4 normativen (IEC 61508-1 bis 4) und 3 informativen (5 bis 7) Teilen, die zwischen 1998 und 2000 veröffentlicht wurden. 2005 wurde sie um den Fachbericht IEC TR 61508-0 erweitert. Wie schon der Titel vermuten lässt, kann die Norm auf ein komplettes Softwaresystem einschließlich der Hardware angewandt werden. Die (für dieses Seminar interessanteren) Anforderungen an die Software werden im dritten Teil der Norm festgelegt. Der zweite Teil befasst sich mit der Hardware; der erste legt allgemeine Anforderungen an das gesamte System fest. Der vierte Teil besteht ausschließlich aus Begriffsdefinitionen und einem Abkürzungsverzeichnis [Bell06,Tzsc06].

In Deutschland ist sie als DIN EN 61508 bzw. VDE 0803 gültig; Ihre Anwendung ist jedoch freiwillig. Sie kann sowohl als eigenständige Norm als auch als Grundlage für Normen spezifischer Anwendungsbereiche verwandt werden. Die Idee dahinter ist, eine hohe Konsistenz sowohl innerhalb eines Anwendungsbereichs als auch zwischen Anwendungsbereichen zu schaffen — u.a. in Hinblick auf Terminologie und Prinzipien [Bell06,Tzsc06].

Die IEC 61508 wird derzeit einer Überarbeitung unterzogen, die im Mai 2008 abgeschlossen sein soll. Ein deutschsprachiger Entwurf davon liegt derzeit als DIN IEC 61508 vor [fNor06a,fNor06b,fNor06c,fNor06d] und aus Gründen der Aktualität werde ich in dieser Ausarbeitung mit diesem Entwurf arbeiten. Sofern erforderlich, werde ich auf Abweichungen zur derzeit noch gültigen Fassung hinweisen.

2.1 Grundlegende Konzepte der Norm

Es wird desöfteren der Begriff der *EUC (equipment under control)* fallen. Dies ist ein Sammelbegriff für „alle Einrichtungen, Maschinen, Geräte oder Anlagen, die Gefährdungen verursachen können und für die sicherheitsbezogene Systeme erforderlich sind“ [dTec07]. Hiervon zu unterscheiden ist das sogenannte *EUC-Leit- oder -Steuerungssystem (EUC control system)*. Dieses reagiert auf „Eingangssignale des Prozessen und/oder eines Bedieners“ und erzeugt Ausgangssignale, „welche die EUC in der gewünschten Art arbeiten lassen“ [fNor06d].

In der Norm wird eine Sicherheitsfunktion (safety function) definiert als eine Funktion, die „dazu vorgesehen ist, unter Berücksichtigung eines festgelegten gefährlichen Vorfalls [...] einen sicheren Zustand für die EUC zu erreichen oder aufrechtzuerhalten“ [fNor06d]. Systeme, die Sicherheitsfunktionen ausführen, gelten als *sicherheitsbezogene Systeme* im Sinne der Norm. Diese können entweder *sicherheitsbezogene E/E/PE-Systeme* sein — womit für sie die Anforderungen der Norm gelten — oder *andere risikoreduzierende Maßnahmen*. Hierzu können „körperliche Gebilde [...] (zum Beispiel ein Abflussystem, eine Brandschutzmauer oder ein Deich)“ zählen, aber auch Systeme, die auf einer „anderen Technologie als Elektrik/Elektronik/programmierbare Elektronik“ basieren (z.B. Hydraulik, Pneumatik)[1]. Das EUCCS selbst kann — muss aber nicht — ein sicherheitsbezogenes System im Sinne der Norm sein.

Im Hinblick auf Sicherheitsfunktionen unterteilt die Norm in drei *Betriebsarten (modes of operation)*. Wird eine Sicherheitsfunktion *mit niedriger Anforderungsrate (low demand mode)* betrieben, bedeutet dies, dass sie „nur auf Anforderung ausgeführt wird, um die EUC in einen festgelegten sicheren Zustand zu überführen“. Die Anforderungsrate beträgt hierbei „nicht mehr als einmal pro Jahr“ [fNor06d]. Die Notabschaltungssequenz eines Kernkraftwerkes wäre ein Beispiel hierfür. Dementsprechend spricht man von *einer Betriebsart mit hoher Anforderungsrate (high demand mode)*, wenn selbige mehr als einmal pro Jahr beträgt. Hält die Sicherheitsfunktion die EUC „als Teil des normalen Betriebs“ in einem sicheren Zustand, spricht man von der *Betriebsart mit kontinuierlicher Anforderung (continuous mode)*.

[1] Die geltende Fassung der IEC 61508 unterscheidet hierbei noch zwischen *Sicherheitsbezogenen Systemen anderer Technologie* und *externen Einrichtungen zur Risikominimierung* [Bell06]. Da aber beide außerhalb des Anwendungsbereiches der Norm liegen, wurden diese Konzepte im aktuellen Entwurf unter der Bezeichung *andere risikoreduzierende Maßnahmen* zusammengefasst.

Sicherheits-Integritätslevel	**mittlere Wahrscheinlichkeit eines gefahrbringenden Ausfalls bei Anforderung der Sicherheitsfunktion**
4	$\geq 10^{-5}$ bis $< 10^{-4}$
3	$\geq 10^{-4}$ bis $< 10^{-3}$
2	$\geq 10^{-3}$ bis $< 10^{-2}$
1	$\geq 10^{-2}$ bis $< 10^{-1}$

Tabelle 1. Sicherheits-Integritätslevel: Ausfallgrenzwerte für eine Sicherheitsfunktion, die in der Betriebsart mit niedriger Anforderungsrate betrieben wird, gemäß IEC 61508-1 [fNor06a].

Sicherheits-Integritätslevel	**Rate gefahrbringender Ausfälle der Sicherheitsfunktion pro Stunde**
4	$\geq 10^{-9}$ bis $< 10^{-8}$
3	$\geq 10^{-8}$ bis $< 10^{-7}$
2	$\geq 10^{-7}$ bis $< 10^{-6}$
1	$\geq 10^{-6}$ bis $< 10^{-5}$

Tabelle 2. Sicherheits-Integritätslevel: Ausfallgrenzwerte für eine Sicherheitsfunktion, die in der Betriebsart mit hoher Anforderungsrate oder kontinuierlicher Anforderung betrieben wird, gemäß IEC 61508-1 [fNor06a].

Sicherheitsfunktionen werden meist redundant realisiert, sodass ein Ausfall eines sicherheitsbezogenen Systemes nicht direkt zum totalen Verlust der entsprechenden Sicherheitsfunktion führen muss. Eine anderes sicherheitsbezogenes System sollte in diesem Fall den Ausfall kompensieren. Diese Fähigkeit nennt man Sicherheitsintegrität (safety integrity). Je nachdem wie effektiv ein solcher Ausfall kompensiert werden kann, kann man das Maß der Sicherheitsintegrität in mehrere Stufen unterteilen, was die IEC 61508 durch die Einteilung in 4 SIL (safety integrity levels) auch tut [Tzsc06]. Die Tabellen 1 und 2 zeigen die genaue Einteilung der SIL abhängig von der Betriebsart einer Sicherheistfunktionen.

2.2 Sicherheitslebenszyklus

Um ihre Anforderungen zu strukturieren, führt die IEC 61508 den sogenannten Sicherheitslebenszyklus ein. Eine Grundvoraussetzung für ein IEC 61508 zertifiziertes Softwaresystem ist der Einsatz eines ähnlichen Lebenszyklus für Planung, Entwicklung und Betrieb. Dieser muss nicht zwingend so aussehen, wie die aus

der Norm[2] entnommene Abbildung 1 — „vorausgesetzt, die Ziele und Anforderungen jedes Abschnitts [der] Norm werden erfüllt“ [fNor06a].

Der Grundgedanke ist, dass alle Aktivitäten, die mit der funktionalen Sicherheit in Zusammenhang stehen, systematisch verwaltet werden, wobei jede Phase des Zyklus wohldefinierte Ziele, Eingaben und Ergebnisse haben. Auf diese Weise lässt sich ein Verifikationsprozess realisieren, in welchem nach Abschluss jeder Phase überprüft wird, dass die zuvor festgelegten Ergebnisse wie geplant erzeugt wurden [Brow00].

Ich werde im Folgenden kurz auf die einzelnen Phasen des gesamten Sicherheitslebenszyklus auf der Grundlage der Norm [fNor06c] eingehen.

Konzept In dieser ersten Phase sollen gründliche Kenntnisse über die EUC und ihre „physikalische Umgebung“ erworben werden. Insbesondere mögliche Gefährdungsquellen und -situationen sollen ermittelt werden. Auch müssen Informationen über die aktuellen nationale und internationalen Sicherheitsvorschriften eingeholt werden. Alle so „gewonnenen Ergebnisse und Informationen müssen dokumentiert werden“.

Quellen für das zu sammelnde Wissen können u.a. auch Erfahrungsberichte über ähnliche Anlagen sein — insbesondere solche, die von gefährlichen Vorfällen handeln. Diese können später in der Gefährdungs- und Risikoanalyse verwendet werden, um mögliche Gefährdungen einschätzen zu können.

Definition des gesamten Anwendungsbereiches Es wird nun die Grenze der EUC/EUCCS festgelegt, „um alle Einrichtungen und Systeme zu berücksichtigen [...], die mit den relevanten Gefährdungen und Gefährdungssituationen in Verbindung stehen“. Dies schließt explizit auch Menschen ein. Ebenfalls in dieser Phase werden auch durch mögliche Unfälle ausgelöste Ereignisse festgelegt — sofern sie betrachtet werden müssen. Darüberhinaus werden sämtliche externen Ereignisse, die in der folgenden Gefährdungs- und Risikoanalyse betrachtet werden müssen, gesammelt.

Wie auch in der Phase zuvor, verlangt die Norm, dass gewonnene Informationen und Ergebnisse dokumentiert werden.

Gefährdungs- und Risikoanalyse Die Gefährdungs- und Risikoanalyse bildet die Grundlage für alle zukünftigen Entscheidungen. Aus diesem Grund ist es immens wichtig, dass ihre Ergebnisse bis zur Außerbetriebnahme des Systems festgehalten werden. Sollten zu einem späteren Zeitpunkt Entscheidungen getroffen werden, die diese Grundlage verändern, muss eine erneute Gefährdungs- und Risikoanalyse durchgeführt werden. Auch kann es vorkommen, dass sich durch

[2] Abgebildet ist der Lebenszyklus nach dem Entwurf vom Juli 2006 der DIN IEC 61508. In der derzeit noch gültigen Fassung DIN EN 61508 fehlt der Kasten 9, Kasten 10 trägt die Nummer 9 und neben Kasten 11 wird in der alten Fassung noch ein ebenfalls gestrichelter Kasten 10 dargestellt, der für die Realisierung „sicherheitsbezogener Systeme anderer Technologie“ steht [Bell06].

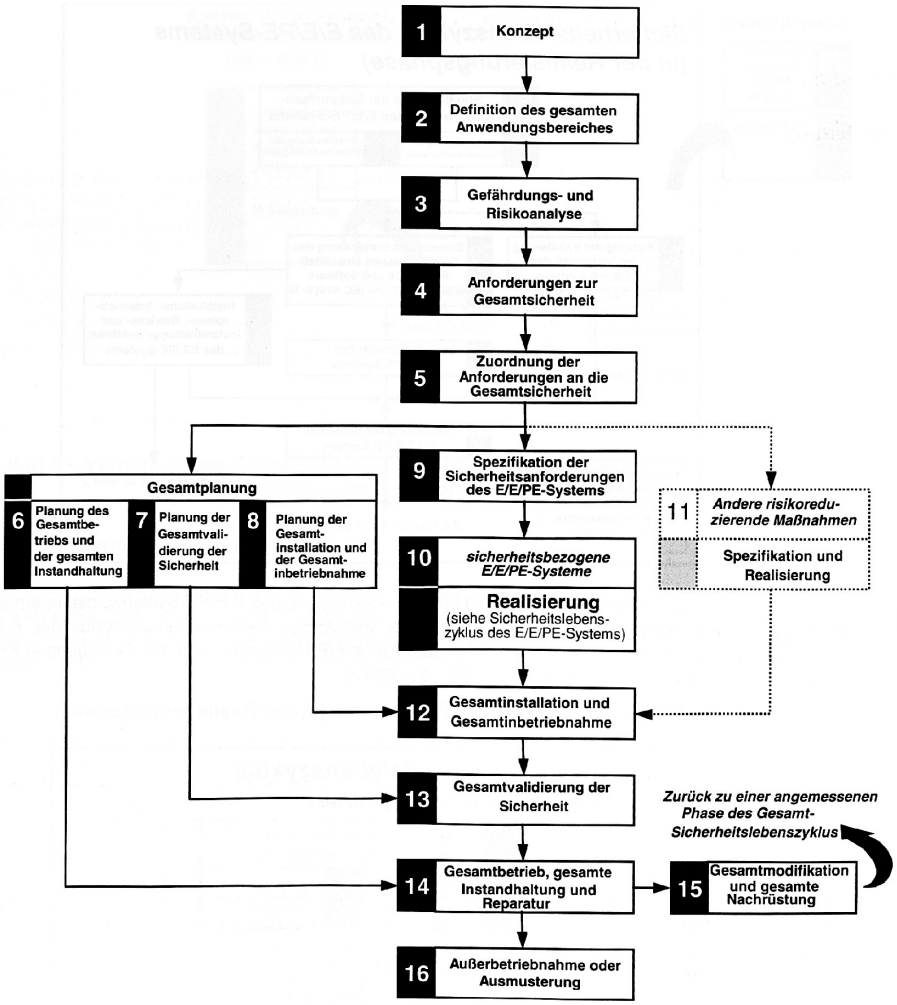


Abbildung 1. Der gesamte Sicherheitslebenszyklus nach IEC 61508-1 [fNor06a]. Gestrichelte Phasen liegen außerhalb des Anwendungsbereiches der Norm.

die Analyse der in der vorherigen Phase festgelegten Gefährdungen die Grenzen des zu konstruierenden Systems verschieben. Die Norm sieht in diesem Fall vor, dass die beiden Phasen in mehreren Iterationen durchlaufen werden.

Wie der Name schon sagt, handelt es sich um keine reine Risikoanalyse. Zuallererst wird die Gefährdung selbst begutachtet und ihre Beseitigung bzw. Minderung in Erwägung gezogen. Wie das genau aussehen soll, erläutert die Norm nicht, sie merkt lediglich an, dass es von „primärer Bedeutung“ sei, „dass erkannte Gefährdungen [...] an der Quelle beseitigt werden, zum Beispiel durch die Anwendung von inhärent sicheren Prinzipien und [...] bewährter Praktiken“.

Zur Analyse des Risikos müssen ausgehend von den in der vorangegangen Phase ermittelten Gefährdungen alle Abläufe von Ereignissen, die zu gefährlichen Vorfällen führen können, ermittelt werden. Zudem wird für alle gefährlichen Vorfälle eine Wahrscheinlichkeit festgelegt. Diese Werte sind natürlich hypothetisch, werden jedoch für die folgenden Schritte benötigt. Um so wichtiger ist es also, dass in der Konzeptphase gründlich gearbeitet wurde.

Anforderungen zur Gesamtsicherheit Es wird nun basierend auf den gefährlichen Vorfällen aus 2.2 „ein Satz aller notwendigen Gesamt-Sicherheitsfunktionen [...] ermittelt“. Diese werden zu diesem Zeitpunkt noch unabhängig von der später zugrundeliegenden Technologie formuliert. Als Beispiel für so eine Gesamt-Sicherheitsfunktion nennt die Norm u.a. „die Verhinderung des Übersteigens der Temperatur in Behälter X über 250°C“. Zudem muss nun festgelegt werden, welche Anforderungen an die Gesamt-Sicherheitsinegrität gestellt werden müssen, damit das Risiko ein tolerierbares Niveau erreicht. Dies kann durch Festlegung entweder der erforderlichen Risikominderung oder einer tolerierbaren Rate gefährlicher Vorfälle geschehen.

Ebenfalls muss nun das EUCCS genauer betrachtet werden: Ergibt die Beurteilung des EUC-Risikos, dass die Rate gefahrbringender Ausfälle des EUCCS kleiner als 10^{-5} sein muss, so ist dieses als sicherheitsbezogenes System zu sehen und dementsprechend einzustufen — darüber mehr in Abschnitt 2.2. Auch wenn das EUCCS nicht als sicherheitsbezogenes System gilt, kann eine angemessene Reaktion auf den Ausfall des EUCCS eine Gesamt-Sicherheitsfunktion darstellen.

Als Ergebnis dieser Phase sollte eine sog. *Spezifikation der Anforderungen zur Gesamtsicherheit* vorliegen, die die Anforderungen an die Sicherheitsfunktionen und an ihre Sicherheitsintegrität beschreibt.

Zuordnung der Anforderungen an die Gesamtsicherheit Erst jetzt werden die vorgesehenen sicherheitsbezogenen Systeme festgelegt und den eben ermittelten Sicherheitsfunktionen zugeordnet, damit das tolerierbare Risiko für die jeweilige Sicherheitsfunktionen erreicht wird. Hierbei müssen für alle sicherheitsbezogenen Systeme bezüglich der von ihnen ausgeführten Sicherheitsfunktionen Ausfallgrenzwerte festgelegt werden. Auch werden die zu realisierenden sicherheitsbezogenen E/E/PE-Systeme nach SIL eingestuft, basierend auf der/den Sicherheitsfunktion(en), die sie ausführen. Führt ein solches System mehr als eine

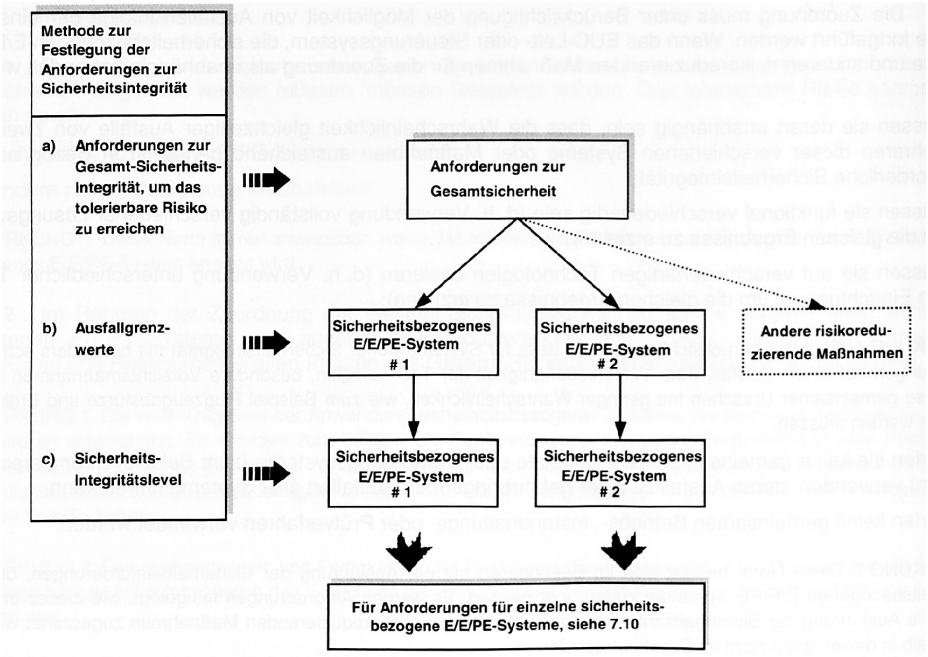


Abbildung 2. Methode zur Festlegung der Anforderungen zur Sicherheitsintegrität nach IEC 61508-1 [fNor06a].

Sicherheitsfunktion aus, so wird es in der Regel[3] nach dem höchsten SIL behandelt.

Diese beschriebene Zuordnung ist iterativ, d.h., „wenn sich herausstellt, dass das tolerierbare Risiko nicht erreicht werden kann, dann müssen die Spezifikationen [der sicherheitsbezogenen Systeme] geändert und die Zuordnung wiederholt werden“.

Abbildung 2 soll die Kernpunkte dieser Phase noch einmal verdeutlichen. Es folgt nun die Gesamtplanung (Abschnitte 2.2, 2.2 und 2.2), die sich mit der späteren Inbetriebnahme und dem Betrieb befasst, sowie die Realisierung der sicherheitsbezogenen Systeme (Abschnitte 2.2, 2.2 und 2.2).

Planung des Gesamtbetriebs und der gesamten Instandhaltung Dieser erste Teil der Gesamtplanung widmet sich ganz dem späteren Betrieb. Hierzu wird ein Plan erstellt, der ganz genau festlegt, welche Handlungen „routinemäßig durchgeführt werden müssen, um die funktionale Sicherheit der sicherheitsbezogenen E/E/PE-Systeme aufrechtzuerhalten“.

[3] Die Norm beschreibt Ausnahmen, auf die ich jedoch nicht im Detail eingehen möchte. Der interessierte Leser sei auf [fNor06a, Kapitel 7.6] verwiesen.

Es werden zudem Einschränkungen des Betriebs der EUC festgelegt, die in bestimmten Fällen[4] „notwendig sind, um einen unsicheren Zustand zu verhindern“. Außerdem ist festzulegen, wann diese Einschränkungen zurückgenommen werden dürfen, wie die Rückkehr zum Normalbetrieb auszusehen hat und wie festgestellt wird, dass der Normalbetrieb wieder erreicht worden ist.

Im Abschnitt 2.2 wird gefordert, dass eine „chronologische Dokumentation des Betriebs, der Reparatur und der Instandhaltung“ geführt wird. Wie diese Dokumentation auszusehen hat, muss ebenfalls bereits jetzt bestimmt werden.

Die Norm betont an dieser Stelle, dass der erstellte Plan mit denjenigen abzustimmen ist, „die verantwortlich sind für den zukünftigen Betrieb und die Instandhaltung“ — nicht nur der sicherheitsbezogenen Systeme, sondern auch solcher „nichtsicherheitsbezogener Systeme, welche die Möglichkeit haben, Anforderungen an die [sicherheitsbezogenen Systeme] zu stellen“.

Planung der Gesamtvalidierung der Sicherheit Im Sicherheitslebenszyklus findet nach der Installation und Inbetriebnahme die sog. Gesamtvalidierung der Sicherheit statt; Der genaue Ablauf wird bereits jetzt geplant. Der zu entwickelnde Plan beschreibt, wann die Validierung von wem unter welchen Umgebungsbedingungen durchgeführt wird. Hierzu sind insbesondere eine „technische Strategie für die Validierung (zum Beispiel analytische Methoden, statistische Tests usw.)“ und „Kriterien für Bestehen und Nichtbestehen“ zu erarbeiten.

Planung der Gesamtinstallation und Gesamtinbetriebnahme Ebenfalls zum Gesamtplanungs-Block gehört die Planung der Installation und Inbetriebnahme der zu realisierenden Systeme. Hierbei wird jeweils ein Plan für die Installation und die Inbetriebnahme entwickelt, der einen Zeitplan und die Vorgehensweise darlegt. Es werden darüberhinaus die Personen festgelegt, die für die verschiedenen Teile der Installation und der Inbetriebnahme verwantwortlich sind.

Ähnlich wie schon bei der Validierung müssen Kriterien bestimmt werden, nach denen bestimmt werden kann, wann die Installation einzelner Systeme sowie die Installationstätigkeiten insgesamt abgeschlossen sind.

Spezifikation der Sicherheitsanforderungen des E/E/PE-Systems Bevor es an die konkrete Realisierung eines E/E/PE-Systems geht, werden seine Sicherheitsanforderungen hinsichtlich den Anforderungen zu seiner Sicherheitsfunktion und Sicherheitsintegrität definiert. Diese zu erstellende Spezifikation „muss in einer Art und Weise ausgedrückt und strukturiert werden, so dass sie klar, genau, unzweideutig, nachprüfbar, testbar, pflegbar und ausführbar ist“. Sie hat sich zudem an diejenigen zu richten, „die wahrscheinlich die Informationen an irgendeiner Stufe des Sicherheitslebenszyklus [...] verwenden“. Enthalten muss sie eine Beschreibung aller Sicherheitsfunktionen, Schnittstellen und Bedienerschnittstellen des geplanten Systems.

[4] Fehlverhalten eines Systems, Instandhaltungsarbeiten

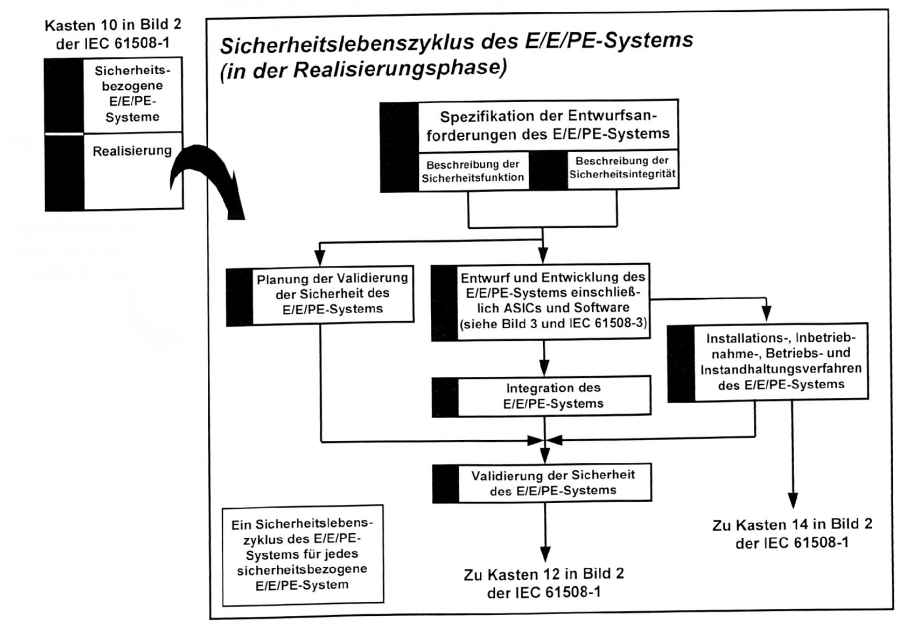


Abbildung 3. Sicherheitslebenszyklus des E/E/PE-Systems (in der Realisierungsphase) nach IEC 61508-2 [fNor06b].

Sicherheitsbezogene E/E/PE-Systeme: Realisierung Nun geht es an die konkrete Realisierung der zuvor spezifizierten E/E/PE-Systeme. Der Kasten 10 aus dem Gesamt-Sicherheitslebenszyklus wird hierfür noch einmal etwas detaillierter beschrieben, wie Abbildung 3 zeigt. Anders als in den übrigen Phasen des Sicherheitslebenszyklus, betrachtet man hier jedes zu realisierende System für sich.

Es wird eine *Spezifikation der Entwurfsanforderungen* erstellt, die die einzelnen Hard- und Softwareelemente identifiziert und die Anforderungen an selbige sowie an ihre Integration festlegt. Ähnlich wie beim Gesamtsystem findet parallel zur eigentlichen Entwicklung bereits die *Planung der Validierung der Sicherheit* statt (vgl. auch 2.2). Mit der eigentlichen Realisierung der Software auf PE-Systemen werde ich mich in Abschnitt 2.3 gesondert befassen. Die Realisierung der Hardware werde ich auslassen, da dies den Rahmen dieses Seminars sprengen würde.

Andere risikoreduzierende Maßnahmen: Spezifikation und Realisierung Wie schon in Abschnitt 2.1 gesagt, fallen andere risikoreduzierende Maßnahmen nicht in den Anwendungsbereich der IEC 61508. Dieser Punkt wurde jedoch der Vollständigkeit halber in den Sicherheitslebenszyklus aufgenommen, da solche Maßnahmen einen Teil des Gesamtsystems darstellen. Analog zu den

E/E/PE-Systemen in den Kästen 9 und 10 des Sicherheitslebenszyklus, sollen nun die geplanten risikoreduzierenden Maßnahmen realisiert werden — und zwar so, dass sie den zuvor festgelegten Anforderungen an die Sicherheitsfunktion/Sicherheitsintegrität genügen.

Gesamtinstallation und Gesamtinbetriebnahme Hier laufen die verschiedenen Entwicklungen wieder zusammen. Die realisierten Systeme werden nun zuerst installiert und anschließend in Betrieb genommen. Wie beides abzulaufen hat, wurde bereits detailliert geplant (vgl. 2.2); Demnach ist es nicht verwunderlich, dass die Norm an dieser Stelle fordert, dass nach dem erstellten Plan vorzugehen ist. Alle Tätigkeiten sind zu dokumentieren — insbesondere die Behebung von Ausfällen und Inkompatibilitäten.

Gesamtvalidierung der Sicherheit Das frisch in Betrieb genommene System wird nun auf Herz und Nieren geprüft, denn es gilt zu überprüfen, ob es tatsächlich die zuvor festgelegten Sicherheitsanforderungen erfüllt. Wie das vonstatten gehen soll — insbesondere, welche Tests durchgeführt werden sollen und wann diese als Bestanden gelten —, sollte bereits geplant worden sein (vgl. 2.2). Auch die Schritte der Validierung müssen dokumentiert werden. Fallen Teile der Validierung negativ aus, müssen die betroffenen Systeme modifiziert werden und es wird zu einem entsprechenden früheren Punkt im Lebenszyklus zurückgekehrt.

Gesamtbetrieb, gesamte Instandhaltung und Reparatur Nachdem das System nun in Betrieb genommen und sicherheitstechnisch validiert wurde, beginnt der normale Betrieb. Oberstes Ziel ist es hier natürlich, dass die funktionale Sicherheit aufrecht erhalten wird. Hierzu ist nachdem bereits im Vorfeld erstellten Plan vorzugehen (vgl. 2.2).

Die Systemdokumentation ist während des gesamten Betriebs zu pflegen und denjenigen zur Verfügung zur stellen, die für den Betrieb und Instandhaltung der Teilsysteme verantwortlich sind. In regelmäßigen Abständen sind Audits der funktionalen Sicherheit durchzuführen und ihre Ergebnisse zu dokumentieren. Ebenso dokumentiert werden müssen alle Fehler, die bei routinemäßigen Instandhaltungsarbeiten oder im tatsächlichen Betrieb aufgetreten sind. Wie mit solchen Fehlern umgegangen werden kann, zeigt das aus [fNor06a] entnommene Beispiel auf Abbildungen 9 und 10 in Anhang 5.2.

Gesamtmodifikation und gesamte Nachrüstung Wie bei jedem anderen Softwaresystem auch, müssen von Zeit zu Zeit Modifikationen und Nachrüstungen durchgeführt werden.

Hierzu wird zuvor eine *Modifikationsanforderung* erstellt, die genau beschreibt, welche festgestellten Gefährdungen betroffen sein könnten, welche Änderungen genau vorgesehen sind und aus welchen Gründen diese durchgeführt werden sollen. Daraufhin muss eine *Einflussanalyse* durchgeführt werden, welche auch eine

erneute Gefährdungs- und Risikoanalyse einschließt. Hierbei wird beurteilt, welchen Einfluss die geplanten Modifikationen auf die Arbeitsweise und die funktionale Sicherheit des Gesamtsystems — sowohl während als auch nach dem geplanten Eingriff — haben wird. Werden parallel weitere Modifikationsarbeiten durchgeführt, so müssen auch die mögliches Einflüsse untereinander auch betrachtet werden. Werden Einflüsse auf ein beliebiges sicherheitsbezogenes E/E/-PE-System festgestellt, so wird zu einer angemessenen Phase des Sicherheitslebenszyklus zurückgekehrt.

Wie ein solches Modifikationsverfahren aussehen könnte, zeigt beispielhaft die aus der Norm entnommene Abbildung 11 in Anhang 5.2.

2.3 Realisierung der Software

Das Herzstück des im vorigen Abschnitt vorgestellten Sicherheitslebenszyklus ist die Realisierung der einzelnen Systeme[5]. Analog zur Realisierung des eines E/E/PE-Systems, die in Abbildung 3 auf Seite 148 dargestellt ist, schlägt die Norm einen Lebenszyklus für die Software eines PE-Systems vor, der sich paralell zu dem der Hardware durchführen lässt und deswegen auch einen ganz ähnlichen Aufbau hat.

Sicherheitslebenszyklus für Software Abbildung 4 zeigt diesen Lebenszyklus. Aus der Nummerierung der Phasen wird ersichtlich, dass man sich im Kasten 10 des Gesamt-Sicherheitslebenszyklusses[6] befindet.

Ähnlich wie schon für das Gesamtsystem, werden im ersten Schritt die Sicherheitsanforderungen festgelegt. Diese ergeben sich zum größten Teil aus den Anforderungen, die bereits in Kasten 9 des Lebenszyklusses[7] besprochen wurde, festgelegt wurden, und den Entwurfsanforderungen an die Hardware. Die Norm empfiehlt an dieser Stelle „eine enge Zusammenarbeit zwischen dem Hardware- und Softwareentwickler“ [fNor06c]. Die so festgelegten Anforderungen müssen nach der Entwicklung und Integration validiert werden, die Validierung selbst wird vorher geplant. Der Anforderungen an beide Phasen unterscheiden sich nicht wesentlich von denen zu den Kästen 7 und 13 des Gesamtsystems[8].

Anforderungen an die Softwareentwicklung Wie nach der Norm die eigentliche Softwareentwicklung aussehen sollte, wird in Abbildung 5 dargestellt. Offensichtlich hat man sich sehr an dem klassischen V-Modell orientiert.

Softwarearchitektur Der Entwurf der Software soll „auf der Aufteilung in Elemente/Teilsysteme“ beruhen. Auf diese Weise soll die Komplexität der Software begrenzt und somit ihre Verständlichkeit erhöht werden. Darüberhinaus gilt es,

[5] vgl. Abschnitt 2.2.
[6] vgl. Abbildung 1 auf Seite 144.
[7] vgl. Abschnitt 2.2.
[8] vgl. Abschnitte 2.2 und 2.2.

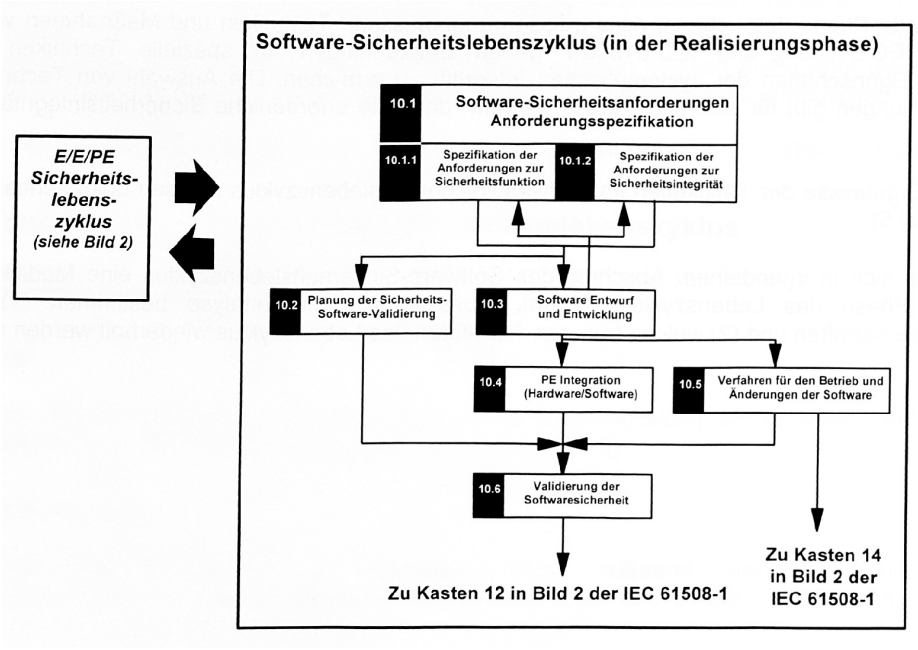


Abbildung 4. Der Sicherheitslebenszyklus für Software nach IEC 61508-3 [fNor06c].

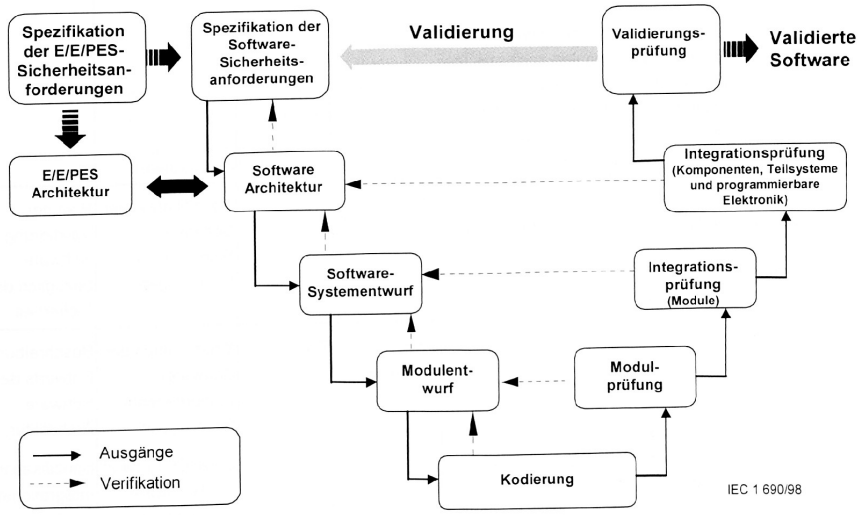


Abbildung 5. Der Entwicklungsprozess für Software nach IEC 61508-3 [fNor06c].

sicherheitsbezogene Komponenten der Software zu separieren. Ziel ist es, die späte Modifikation der Software zu erleichtern und das Einschleppen von neuen Softwarefehlern zu vermeiden. Als wichtige Entwurfsmerkmale werden „Modularität“, „verbergen von Informationen“ und „Einkapselungen“[9] genannt [fNor06c].

Hilfswerkzeuge und Programmiersprachen Die eingesetzte Programmiersprache sicherheitsbezogener Software muss diversen Anforderungen genügen. So schreibt die Norm beispielsweise vor, dass eine streng typisierte Programmiersprache verwendet werden soll, für die es einen Übersetzer gibt, „der entweder nach einer anerkannten nationalen oder internationalen Norm zertifiziert ist oder dessen Eignung für den Einsatz beurteilt worden ist“. Für die eingesetzte Programmiersprache müssen Programmierrichtlinien aufgestellt werden, die „gute Programmiertechniken beschreiben, unsichere Sprachmerkmale verbieten [...] und die Verfahren für die Dokumentation des Quellcodes beschreiben“. Die Programmierrichtlinien sind konsequent auf den Programmcode anzuwenden - ausgenommen derjenigen Teile, die von einem Entwicklungswerkzeug automatisch generiert worden sind [fNor06c].

Solche Werkzeuge werden allerdings ebenso wie Übersetzer als sicherheitsrelevant eingestuft und müssen daher ähnliche Kriterien erfüllen; Gleiches gilt beispielsweise auch für Versionskontrollsysteme [fNor06b].

[9] modularity, information hiding, encapsulation

Abbildung 6. Ein Kernkraftwerk [Wiki08b].

2.4 Sektornormen

Auf der IEC 61508 können wie bereits erwähnt auch Normen aufsetzen, die nur für eine bestimmt Domäne gelten, sogenannte *Sektornormen*. Diese berücksichtigen auch in dieser Domäne bewährte Praktiken, die durchaus weniger komplexe Anforderungen stellen können. Darüberhinaus sind sie in der Terminologie der Domäne geschrieben, was ihre Umsetzung durch einen Domänenexperten erleichtert. Die 61508 selbst muss in diesem Fall nicht mehr betrachtet werden, sie wird durch die Sektornorm für diese Domäne ersetzt [Bell06].

Solche Sektornormen existieren u.a. auch für den Bereich *Kernkraftwerke*, auf den im folgenden Abschnitt eingegangen wird.

3 Kernkraftwerke — ein Anwendungsgebiet von Safety-Normen

Kernkraftwerke wandeln Wärme in Strom um. Wasser wird erhitzt und der so entstandene Wasserdampf treibt eine Dampfturbine an. In diesem Punkt unterscheiden sie sich im Prinzip nicht von anderen Dampfkraftwerken, die die benötigte Wärme z.B. aus Kohle oder Erdöl gewinnen. Bei Kernkraftwerken wird

die Wärme durch Freisetzen von Kernenergie erzeugt. Als Brennmaterial wird hierbei i.d.R. angereichertes Uran verwendet [Wiki08b].

Diese Technologie birgt jedoch auch hohe Risiken. So kann es z.B. durch eine Fehlfunktion des Kühlsystems zu einer sogenannten Kernschmelze kommen, wodurch radioaktives Material unkontrolliert an die Umgebung abgegeben werden kann [Wiki08c]. Solche Unfälle gilt es zu verhindern, weshalb hohe sicherheitstechnische Anforderungen an die Steuerungssysteme von Kernkraftwerken gestellt werden.

In diesem 3. Abschnitt werden einige Normen aus diesem Bereich vorgestellt — insbesondere jene, die die oben erläuterte IEC 61508 für Kernkraftwerke umsetzen sollen.

3.1 IEC 61513

Die IEC 61513 trägt den Titel „Kernkraftwerke — Leittechnik für Systeme mit sicherheitskritischer Bedeutung — Allgemeine Systemanforderungen“ und gilt in Deutschland aktuell in der übersetzte, aber inhaltlich unveränderte Fassung DIN IEC 61513 bzw. VDE 0491-2 vom Oktober 2002 [fNor02].

Sie ist eine sogennante Sektorimplementation der bereits besprochenen IEC 61508 [Bell06], ist also eine Übertragung der generischen Prinzipien der 61508 auf den Industriesektor Kernkraftwerke. Als solche beschreibt sie ebenfalls einen *gesamten Sicherheitslebenszyklus*, der dem der 61508 ähnlich sieht [LLCM+06]. An die Stelle des E/E/PES[10] tritt hier das *leittechnische System*[11] — ein E/E/PES, das sogenannte *leittechnische Funktionen* ausführt, um jeweils „einen bestimmten Teilprozess zu betreiben, zu steuern und/oder zu überwachen“ [fNor02].

Kategorisierung leittechnischer Systeme Die leittechnischen Funktionen werden von der Norm IEC 61226 entsprechend ihrer sicherheitstechnischen Bedeutung in die Kategorien A, B und C eingeteilt [fNor93]:

Kategorie A umfasst all jene Funktionen, die „erforderlich sind, um nicht tolerable Auswirkungen der Störfälle zu verhindern“ [Reak96]. Hierunter fallen z.B. die Kernnotkühlung oder die Notstromversorgung [fdSd05].

Kategorie B umfasst Funktionen, „die erforderlich sind, um die Ausweitung einer Störung zu einem Störfall zu verhindern“. „Bei Ausfall dieser Leittechnik-Funktionen sind nur geringe Schadensauswirkungen zu erwarten“ [Reak96]. Beispiele hierfür wären die „Überwachung der radioaktiven Abgaben“ oder die „Steuerung von Brennelementhandhabungseinrichtungen“ [fdSd05].

Kategorie C enthält dementsprechend alle Funktionen, die weder Kategorie A noch B zugeordnet werden mussten.

Entsprechend dieser Kategorisierung teilt die 61513 alle leittechnischen Systeme in 3 Klassen ein. So führen Systeme der Klasse 1 hauptsächlich Funktionen

[10] siehe Abschnitt 2.1.

[11] instrumentation and control system, I&C system

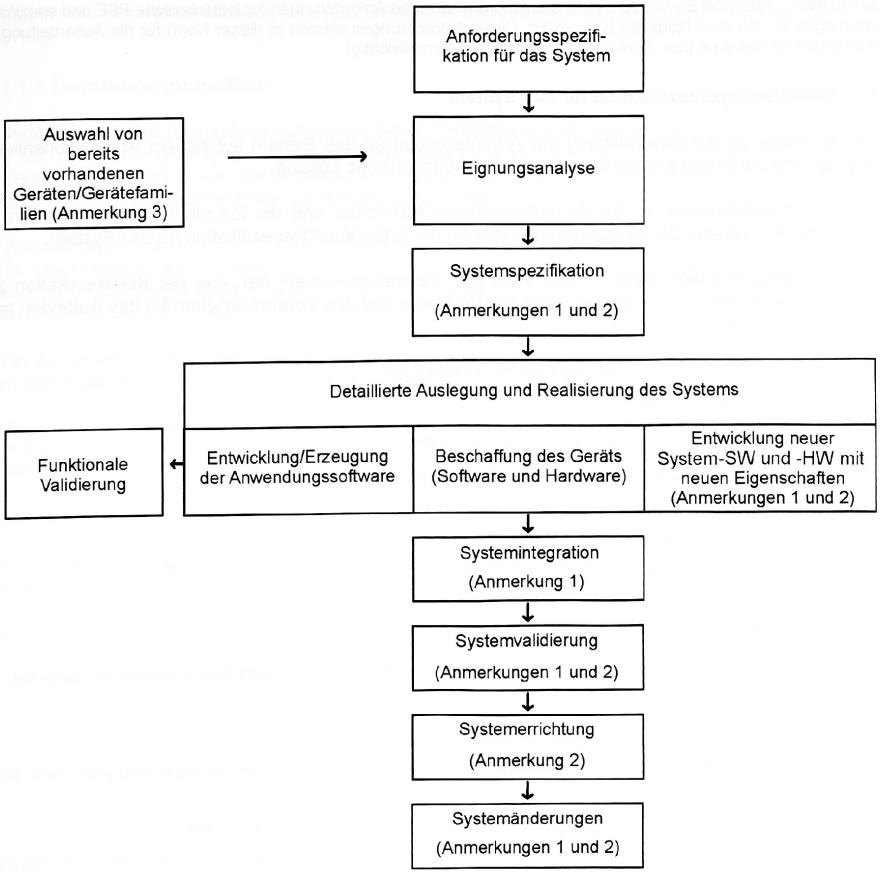


Abbildung 7. Der System-Sicherheitslebenszyklus nach IEC 61513 [fNor02].

der Kategorie A aus, allerdings dürfen sie auch Funktionen der Kategorien B und C ausführen [fNor02]. Diesem Schema folgend führen Systeme der Klasse 2 hauptsächlich Funktionen der Kategorie B aus, dürfen jedoch auch solche der Kategorie C, keinesfalls aber solche der Kategorie A ausführen. Systeme der Klasse 3 schließlich führen ausschließlich Funktionen der Kategorie C aus.

Die RSK-Leitlinien für Druckwasserreaktoren (auf die in Abschnitt 3.3 noch genauer eingegangen wird) definieren diese Kategorien ebenfalls, allerdings verwirrenderweise als Kategorien 1, 2 und 3 [Reak96].

System-Sicherheitslebenszyklus Der System-Sicherheitslebenszyklus ist die konkrete Umsetzung des Sicherheitslebenszylus eines E/E/PE-Systems in der

Realisierungsphase der IEC 61508-2 bzw IEC 61508-3. Er wird in Abbildung 7 dargestellt.

Hauptabweichungen der IEC 61513 von der 61508 Im Anhang D der Norm werden die Hauptunterschiede zwischen IEC 61513 und IEC 61508-1, 61508-2 sowie 61508-4 beschrieben. Die Autoren haben offensichtlich versucht, erfolgreich praktizierte Methoden nicht gewaltsam durch die Konzepte der generischen 61508 zu ersetzen.

So gehen die eben vorgestellten (deterministisch bestimmbaren) Kategorien auf die Sicherheitsrichtlinien der internationalen Atomenergiebehörde IAEA zurück und werden von der nuklearen Industrie traditionell verwendet. Aus diesem Grund treten sie in der IEC 61513 an die Stelle der (auf der probabilistischen Gefährdungs- und Risikoanalyse basierenden) SIL[12]. Auch behandelt die Norm weder die Sicherheitsanalyse der Kernkraftanlage, noch legt sie die Mittel fest, „mit denen die Eignung der Leistungsfähigkeit und die Zuverlässigkeitsanforderungen aufgrund der Ergebnisse dieser Analyse festgestellt werden“. Als Grund hierfür wird angeführt, dass es Praxis sei, hierfür nach den Prinzipien der IAEA sowie anderen IEC-Normen oder nationalen Regelungen vorzugehen [fNor02].

Dem aufmerksamen Leser wird zudem aufgefallen sein, dass in der obigen Aufzählung der 3. Teil der IEC 61508 fehlt, welcher für die Realisierung der Software zuständig ist. In der Tat macht die Norm — wenn überhaupt — nur sehr vage Angaben zu Software. Dies rührt daher, dass es bereits lange vor der 61513 eine IEC-Norm gab, die sich mit sicherheitskritischer Software in Kernkraftwerken befasste: IEC (60)880.

3.2 IEC 60880

IEC Publication 880 wurde im Jahr 1986 herausgegeben, mit dem Ziel, die „bisher bei Hardware-Systemen angewendeten grundsätzlichen Sicherheitsprinzipien für die Nutzung digitaler Systeme [...] in Sicherheitssystemen von Kernkraftwerken“ umzusetzen. 1997 benannte die IEC alle älteren Normen um[13] — aus der 880 wurde die IEC 60880. Später dann wurde ein ergänzender zweiter Teil der Norm herausgegeben, der sich „Softwareaspekten zu Vermeidung von Ausfällen gemeinsamer Ursache, Verwendung von Softwarewerkzeugen und vorgefertigter Software beschäftigt [fNor07]. Im August 2007 ist eine überarbeitete Fassung der 60880 in Kraft getreten, die beide Teile in sich vereint und die noch bis Ende 2008 gültigen Fassungen von 1986 bzw. 2000 ablösen soll. In Deutschland gilt sie unter der Bezeichnung DIN IEC 60880 bzw. VDE 0491-3-2.

Wie ihr Titel *Kernkraftwerke — Leittechnik für Systeme mit sicherheitskritischer Bedeutung — Softwareaspekte für rechnerbasierte Systeme zur Realisierung von Funktionen der Kategorie A* schon verrät, befasst sich die Norm ausschließlich mit Software in Klasse-1-Systemen, also solchen, die leittechnische

[12] vgl. Abschnitt 2.1

[13] Es wurde jeweils 60000 hinzuaddiert. IEC-Normen haben seitdem stets eine Nummer zwischen 60000 und 79999 [Wiki08a].

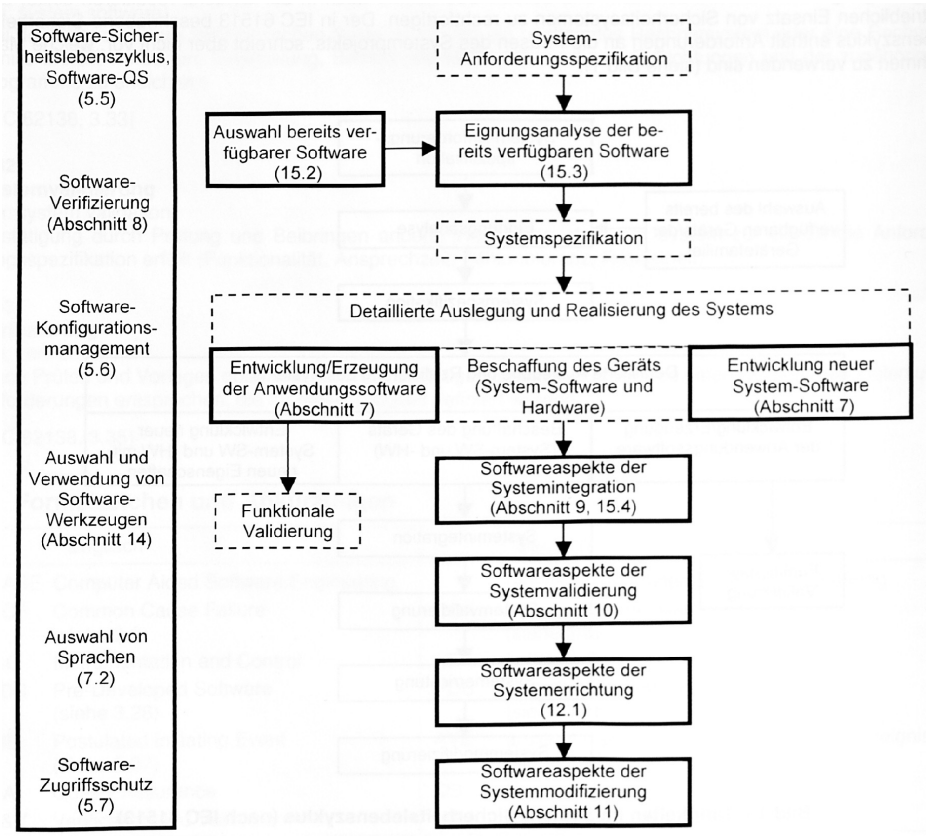


Abbildung 8. Softwarebezogene Tätigkeiten nach IEC 60880 im System-Sicherheitslebenszyklus [fNor02].

Funktionen der Kategorie A ausführen[14]. Diese neue Fassung setzt den System-Sicherheitslebenszyklus der IEC 61513 um, wie Abbildung 8 zeigt. Die Hardware des Systems ist parallel zur Software nach 61513 zu entwickeln. Ähnlich wie auch ihr generisches Pendant, die IEC 61508-3, wird sich am V-Modell orientiert [fNor07]. Ich werde im Folgenden auf einige Aspekte der Norm eingehen.

Projektplanung Die Norm stellt an die Planung des Softwareprojekts einige Anforderungen. So soll das Projekt in mehrere Phase strukturiert werden. Jede Phase ist jeweils in „gut definierte Tätigkeiten unterteilt". Jede Phase soll eine „in gewissem Maß" in sich geschlossene Einheit bilden, darf aber von anderen Phasen abhängen bzw. andere Phasen beeinflussen [fNor07]. Der Ablauf der Phasen darf auch iterativ sein; Eine Phase darf auch schon beginnen, bevor die vorherige abgeschlossen wurde — jedoch darf sie erst abgeschlossen werden, wenn es die vorherige ist [fNor02].

Darüberhinaus muss ein Qualitätssicherungsplan existieren oder in einer frühen Phase der Entwicklung aufgestellt werden, der bestimmte Forderungen an die Entwicklung stellen muss. So muss beispielsweise „gefordert werden, dass die Aktivitäten im Rahmen der Phasen kompetenten Personen zugeordnet werden, die über geeignete Ressourcen verfügen" [fNor07].

Zudem wird die Etablierung eines Konfigurationsmanagements gefordert. Hierbei werden die in der 61513 bereits festgelegten grundlegenden Anforderungen um einige softwarespezifischen Aspekte ergänzt. Hierzu gehört u.a., dass in der Entwicklung befindliche Software deutlich von bereits freigegebener getrennt wird.

Zugriffsschutz Während der Planung und Entwicklung muss stets der Aspekt des Zugriffsschutzes berücksichtigt werden. Darunter versteht die Norm den „Schutz von Software und Daten, sodass nicht autorisierte Personen und Systeme sie nicht lesen oder ändern können". Die hierfür notwendige Zugriffsschutzanalyse sollte zu allererst evaluieren, ob der nötige Zugriffsschutz nicht schon durch Gegenmaßnahmen auf der Ebene des Gesamtsystems oder des Leitsystems zu erreichen ist. Entsprechende Richtlinien definiert die 61513. Sind diese Maßnahmen nicht effektiv genug, müssen Gegenmaßnahmen auf Softwareebene ergriffen werden. Hierzu gehört u.a. eine Benutzerauthentifikation [fNor07].

Merkwürdig mag einem auf den ersten Blick vorkommen, dass eine Safety-Norm auch das Thema Zugriffsschutz behandelt, wo dies doch nach den Definitionen im Abschnitt 1.1 ein Security-Aspekt ist. Wie man aber an anderen Aspekten der Norm sieht, versucht man die Betriebssicherheit u.a. durch eine hohe Softwareproduktqualität zu erreichen. Security ist hier also vielmehr als Teilaspekt der Softwarequalität zu sehen — vgl. hierzu [GrCa87] oder ISO 9126.

Anforderungen Die Norm fordert, dass die Anforderungen an die Software in einer *Anforderungsspezifikation* festgelegt werden. Als Grundsatz für die Anfor-

[14] vgl. Abschnitt 3.1

derungen wird angeführt, dass die Software-Anforderungen beschreiben müssen, „was die Software zu tun hat und nicht, wie sie es tun muss“ [fNor07]. Neben einer Auflistung von Aspekten, die in den Anforderungen abzudecken sind, stellt die Norm auch eigene Anforderungen an das Softwareprodukt, z.B. die Selbstüberwachung.

Umfang der Anforderungen In ihrem Abschnitt 6.1.3 enthält die Norm eine Auflistung darüber, was Software-Anforderungen spezifizieren müssen, u.a. [fNor07]:

- bereitzustellende Anwendungsfunktionen,
- Schnittstellen und Wechselwirkungen der Software mit ihrem Umfeld,
- Parameter der Software, die während des Betriebs geändert werden können,
- die geforderte Leistungsfähigkeit der Software (insbes. Antwortverhalten) sowie
- durch die Software in Hinblick auf ihr Umfeld getroffene Annahmen (externe Abhängigkeiten).

Selbstüberwachung Das Konzept der Selbstüberwachung soll die Zuverlässigkeit des Systems erhöhen. Die zu erstellende Software soll während des Betriebs die Hardware in spezifizierten Zeitabständen sowie das Verhalten der Software selbst überwachen. Ziel ist es, ein Fehlverhalten oder einen Ausfall zu erkennen und eine anschließende Analyse zu ermöglichen. Erreicht werden kann dies u.a. durch „zwischenzeitliche Plausibilitätsprüfungen“ sowie Redundanz und Diversität [fNor07].

Auslegung Die *Auslegung* der Anforderungen ist eigentlich nichts weiter als ein modularisierter Entwurf des Softwaresystems auf der Grundlage der aufgestellten Anforderungen. Die Norm empfiehlt hierbei ein Top-Down-Verfahren. Die Module sollen „klar und verständlich sein“ und ausschließlich über vollständig dokumentierte Schnittstellen angesprochen werden. Diese Schnittstellen sollen „so einfach wie möglich“ gehalten und die Anzahl ihrer Parameter minimiert werden. Rekursive Strukturen gilt es zu vermeiden.

Hierbei soll insbesondere darauf geachtet werden, dass die Software später leicht zu modifizieren ist. Hierzu sollen bereits jetzt Stellen im Programm identifiziert werden, die möglicherweise später geändert werden müssen. Diese Stellen sollen beim Entwurf in möglichst unabhängige Module ausgelagert werden. Darüberhinaus macht die Norm weitere Vorschläge zu Funktionalitäten, die als eigene Module vom Anwendungsprogramm getrennt werden sollen.

Die „Zwischenprodukte“ der Auslegung, also die verschiedenen Ebenen der Modulstruktur sollen hinsichtlich ihrer Vollständigkeit und Konsistenz durch Inspektionen verifiziert werden.

Ergebnis dieser Phase soll eine *Software-Auslegungsspezifikation* sein, die gegen die Anforderungsspezifikation validiert wird und als Basis für die anschließende Implementierung dient [fNor07].

Implementierung Die Software soll nach der Norm in einer Sprache realisiert werden, die „strikten (oder gut definierten) semantischen und Syntaxregeln“ folgt. Für sie sollten ein gründlich getesteter Übersetzer sowie Werkzeuge für automatisches Prüfen vorhanden sein.

Wie schon in der 61508-3[15] wird auch hier das Aufstellen von Programmierrichtlinien gefordert. Ziel ist hier eine einheitliche „Modulgestaltung“ zur Förderung der Verständlichkeit der Modulstruktur. Für die konkrete Codierung enthält die Norm einen umfangreichen Anhang mit Richtlinien, die hauptsächlich dazu dienen, den Kontrollfluss — z.B. durch Vermeidung von rückwärtsgerichteten Sprungbefehlen oder Sprüngen in eine Schleife hinein — und den Datenfluss — Subroutinen „sollten mit ihrem Umfeld ausschließlich über ihre Parameter kommunizieren“ — zu vereinfachen, bzw. analysierbar und testbar zu machen.

Die Implementation wird anschließend verifiziert, wobei hier — im Gegensatz zum Entwurf — eine Bottom-Up-Strategie zu verfolgen ist: Es werden also zuerst die grundlegendsten Module verifiziert, anschließend sukzessive die darauf aufbauenden Module. Das Verifizierungsteam, das übrigens nicht das Entwicklerteam sein darf, hat dabei einen detaillierten *Software-Prüfbericht* zu erstellen, der u.a. folgende Punkte abdecken muss [fNor07]:

- Die für die Prüfung verwendete Hardwarekonfiguration,
- geprüfte Eingangswerte,
- erwartete und erzielte Ausgangswerte,
- Übereinstimmung mit den Akzeptanzkriterien der zuvor erstellten *Prüfspezifikation* und
- Aufzeichnung über die Zwischenfälle mit Fehlern.

Vergleich mit IEC 61508-3 Da die 60880 eine Sektorimplementation der 61508-3 darstellt, liegt ein Vergleich der beiden Normen nahe. Beide verfolgen das Ziel, Programmierfehler durch strukturierten modularisierten wohldokumentierten Entwurf, Verringerung der Komplexität und weitere das Programmverständnis fördernde Maßnahmen zu vermeiden. Über zusätzliche qualitätssichernde Maßnahmen, wie z.B. die Verifikation aller Zwischenprodukte durch unabhängige Testteams sollen die Zuverlässigkeit erhöhen.

In einigen Punkten — insbesondere bei den Entwurfs- und Programmierrichtlinien — wird die 60880 deutlich konkreter. Allerdings erschließt sich nicht ganz, inwiefern diese Unterschiede auf das spezielle Anwendungsgebiet der 60880 zurückzuführen ist. Das eigentlich generell für sicherheitskritische Systeme interessante Konzept der Selbstüberwachung sucht man in der 61508-3 zudem vergebens.

3.3 Weitere Sicherheitsormen für Kernkraftwerke

IEC 62138 Aufgrund ihrer Ähnlichkeit zur IEC 60880 sei die IEC 62138 (*Kernkraftwerke — Leittechnik für Systeme mit sicherheitstechnischer Bedeutung —*

[15] vgl. Abschnitt 2.3.

Softwareaspekte für rechnerbasierte Systeme zur Realisierung von Funktionen der Kategorien B oder C) nur der Vollständigkeit halber genannt.

Sie bildet das Gegenstück zur 60880, indem sie sich mit Systemen der Klassen[16] 2 und 3 befasst. Sie übernimmt den in Abbildung 8 auf Seite 157 dargestellten Lebenszyklus sowie viele Konzepte der 60880 (wie z.B. den Zugriffsschutz), lässt aber z.B. die Selbstüberwachung weg. Insgesamt ist sie weitaus kompakter und weniger restriktiv als die 60880, wobei für Klasse-3-Systeme noch weniger Restriktionen gelten als für Systeme der Klasse 2.

Die in Deutschland derzeit gültige Fassung ist die DIN IEC 62138 (VDE 0491-3-3) vom September 2004 [fNor04].

RSK-Leitlinien für Druckwasserreaktoren Die *Reaktorsicherheitskommission (RSK)* ist ein Expertengremium, das den Bundesumweltminister in Fragen der Sicherheit von Kernkraftwerken berät. Mit den *RSK-Leitlinien für Druckwasserreaktoren* hat sie eine Empfehlung herausgegeben, die zur Beurteilung eines Kernkraftwerkes herangezogen werden kann [Wiki08d].

In Kapitel 7 der Leitlinien werden einige Anforderungen an die Softwareentwicklung aufgestellt. Wie auch bei den IEC-Normen wird hier eine Abstufung vorgeommen, wobei die Anforderungen bei Klasse-1-Systemen wieder am restriktivsten sind. So fordert sie für den Entwurf und die Implementierung von Klasse-1-Systemen den Einsatz von „formalen [...] Konstruktions- und Prüfmethoden“, einen einfachen Aufbau und einen Funktionsumfang, der „auf das notwendige Maß begrenzt“ ist. Zum Vergleich: Bei Klasse-2-Systemen genügen hier schon „rechnergestützte Beschreibungen und Testverfahren, die den Nachweis der korrekten Arbeitsweise unterstützen“ und Klasse-3-Systeme seien lediglich „nach anerkannten Methoden der Softwaretechnik zu qualifizieren“. Die selbstüberwachende Auslegung der Programme taucht hier im Übrigen nicht nur bei Klasse-1 sonder auch bei Klasse-2-Systemen auf [Reak96].

4 Rückblick

Es wurden nun die generische Norm IEC 61508 und ihre Sektorimplementationen für Kernkraftwerke (61513, 60880 und 62138) mit dem Schwerpunkt Software-Safety betrachtet. Es wurde jeweils ein Lebenszyklus definiert, der das System von der Konzeption bis zur Abschaltung begleiten soll. Zur Realisierung wird das Gesamtsystem in mehrere Teilsysteme mit eigenem Lebenszyklus zerlegt. Die Betriebssicherheit der Teilsysteme soll hierbei durch eine Reihe von Vorgaben und Maßnahmen erreicht werden. Hierzu gehört u.a. eine Risikoanalyse und Konzentration von QS-Maßnahmen wie Verifikation und Test auf besonders kritische Module.

[16] Zur Klassifizierung der leittechnischen Systeme, vgl. Abschnitt 3.1

5 Anhang

5.1 Glossar

Ich werde im Folgenden einige Begriffe erläutern, die des Öfteren fallen, wenn über Safety gesprochen wird.

Schaden (harm) Ein Schaden ist generell ein nicht wünschenswerter Effekt auf die Umwelt des Systems. Ihn gilt es durch Safety-Maßnahmen zu vermeiden. Die [fNor06d] versteht darunter allgemein die „physische Verletzung oder Schädigung der Gesundheit von Menschen oder Schäden von Gütern oder der Umwelt“.

Gefährdung (hazard) Eine Gefährung ist „eine potentielle Schadensquelle“ [fNor06d], also eine potentielle Ursache für einen gefählichen Vorfall.

Gefährlicher Vorfall (hazardous event) Ein gefährlicher Vorfall ist ein Ereignis, das „zu einem Schaden führen kann“ [fNor06d]. Safety-Anforderungen könnten sein, das Risiko des Eintretens sowie das Ausmaß und die Härte der Konsequenzen klein zu halten [Tzsc06].

Risiko (risk), tolerierbares Risiko (tolerable risk) Das Risiko bezeichnet die „Kombination aus der Wahrscheinlichkeit, mit der ein Schaden auftritt, und dem Ausmaß dieses Schadens“. Tolerierbar ist ein Risiko dann, wenn es „basierend auf den aktuellen gesellschaftlichen Wertvorstellungen in einem gegebenen Zusammenhang tragbar ist“ [fNor06d].

Versagen gemeinsamer Ursache (common cause failure, CCF) Das Versagen zweier oder mehrerer Systeme oder Komponenten „infolge eines einzelnen spezifizierten Ereignisses oder Grundes“ wird als CCF bezeichnet [fNor07]. Ursache eines CCF ist häufig, dass bei der Entwicklung einer Komponente nicht mit dem Fehlverhalten einer anderen Komponente gerechnet wurde.

Redundanz (redundancy) Unter Redundanz versteht man das Vorhandensein mehrerer (identischer oder verschiedener) Systeme oder Komponenten, sodass „ jedes, unabhängig von Betriebszustand oder Versagen irgendeines anderen, die geforderte Funktion ausüben kann“ [fNor07].

Diversität (diversity) Diversität oder diversitäre Redundanz bezeichnet das „Vorhandensein von zwei oder mehreren unterschiedlichen Verfahren oder Mitteln, um ein bestimmtes Ziel zu erreichen“. Insbesondere wird diese Maßnahme gegen Versagen gemeinsamer Ursache eingesetzt.

5.2 IEC 61508-1: Beispiele

Dieser Anhang enthält einige Beispielmodelle zur IEC 61508-1, die aus Gründen der Übersichtlichkeit nicht in den Haupttext integriert wurden. Die Abbildungen 9 und 10 gehören hierbei zu Abschnitt 2.2 und Abbildung 11 zu Abschnitt 2.2.

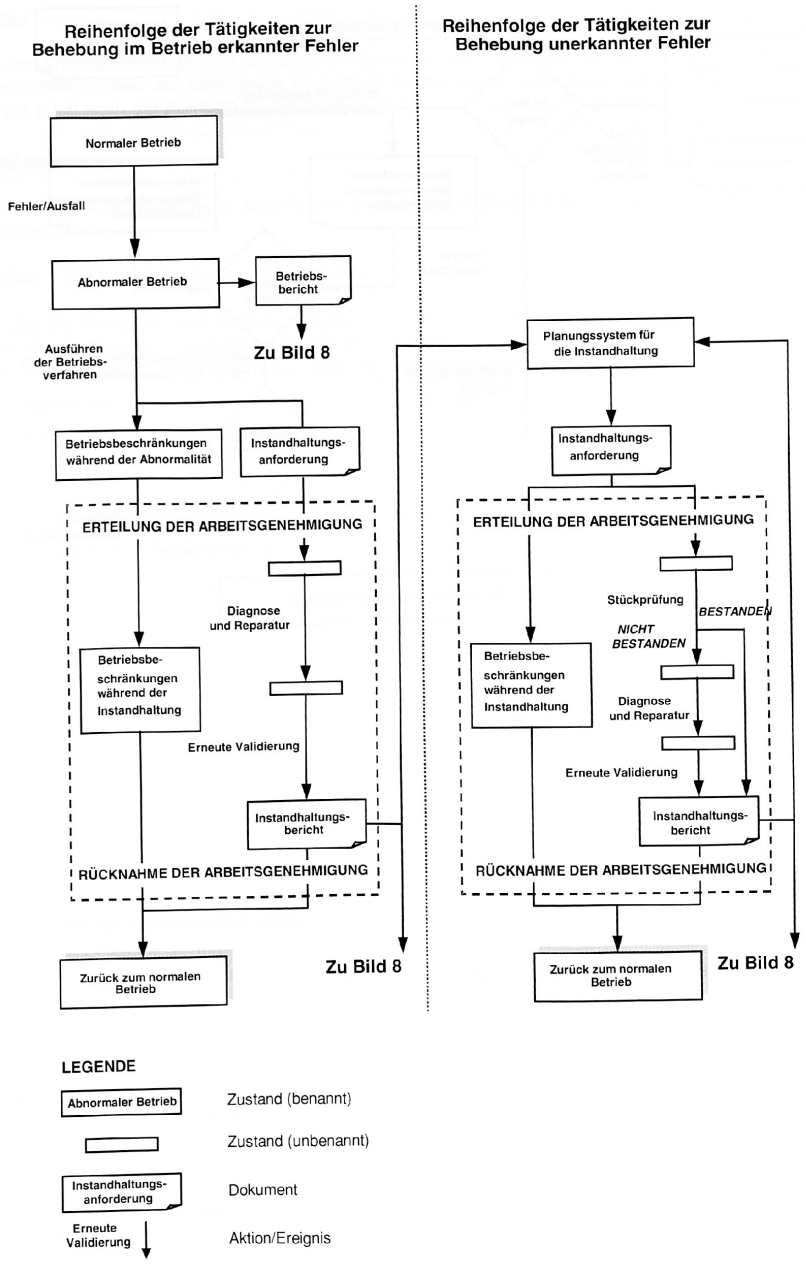


Abbildung 9. Beispiel eines Modells der Betriebs- und Instandhaltungstätigkeiten nach IEC 61508-1 [fNor06a].

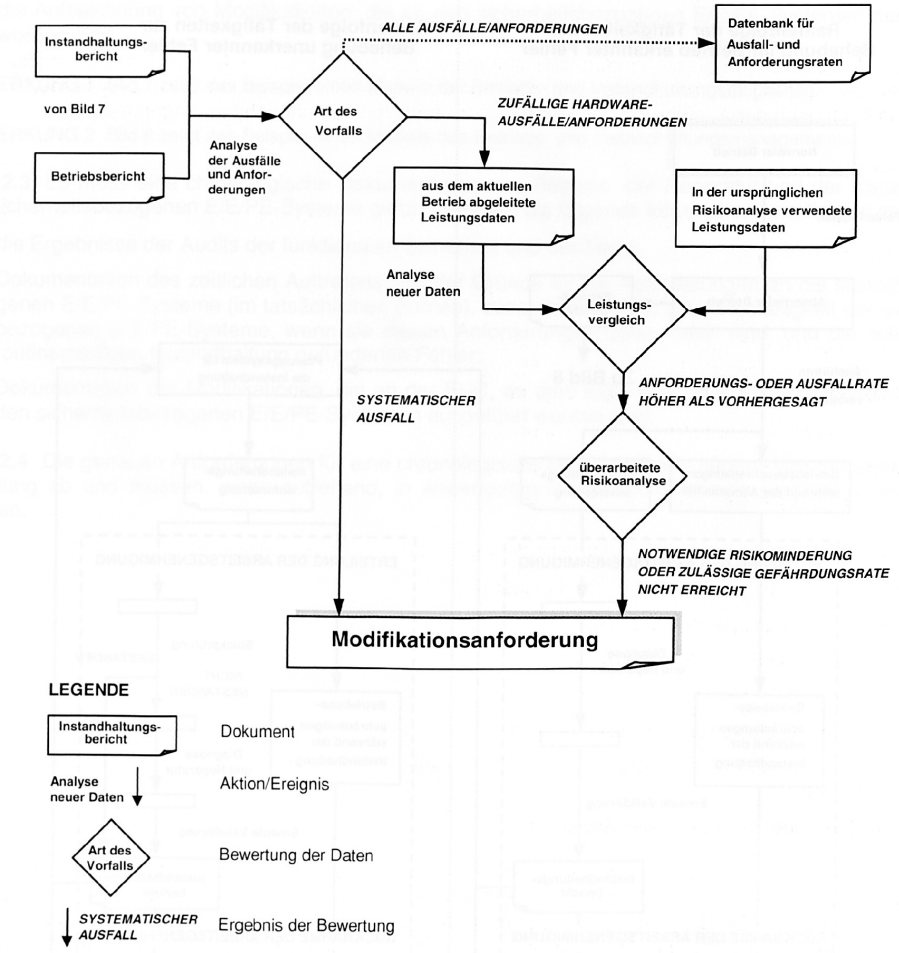


Abbildung 10. Beispiel eines Modells des Betriebs- und Instandhaltungsmanagements nach IEC 61508-1 [fNor06a].

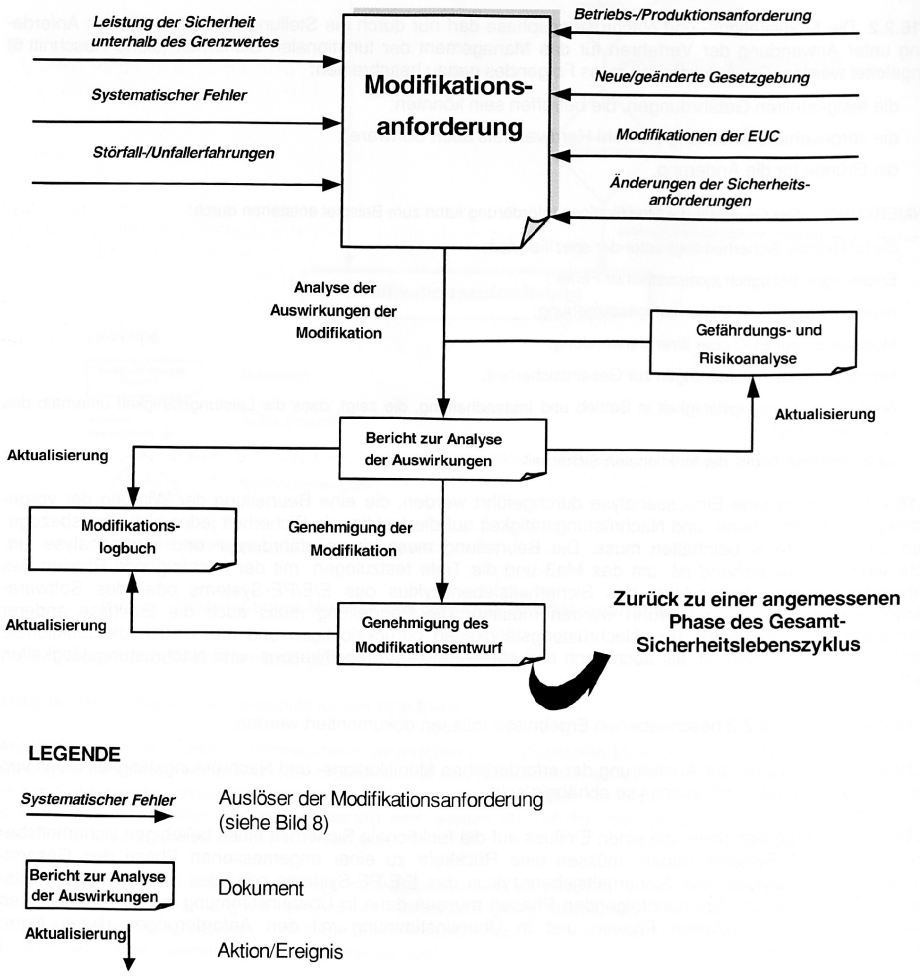


Abbildung 11. Beispiel eines Modells eines Modifikationsverfahrens nach IEC 61508-1 [fNor06a].

Literatur

Bell06. Ron Bell. Introduction to IEC 61508. In *SCS '05: Proceedings of the 10th Australian workshop on Safety critical systems and software*, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc., S. 3–12.

Brow00. Simon Brown. Overview of IEC 61508. Design of electrical/electronic/-programmable electronic safety-related systems. *Computing & Control Engineering Journal*, 11(1), 2 2000, S. 6–12.

dTec07. Haus der Technik. Funktionale Sicherheit SIL — Sicherheitstechnische Systeme nach IEC 61508, IEC 61511, 2007. Webseite des Hauses der Technik in Essen am 1.12.2007, 14:35.

fdSd05. Schweizer Hauptabteilung für die Sicherheit der Kernanlagen. Anforderungen für die Anwendung von sicherheitsrelevanter rechnerbasierter Leittechnik in Kernkraftwerken, 4 2005. HSK R-046.

fNor93. Deutsches Intitut für Normung. Kernkraftwerke — Sicherheitsleittechnik — Kategorisierung, 1993. DIN IEC 61226.

fNor02. Deutsches Intitut für Normung. Kernkraftwerke – Leittechnik für Systeme mit sicherheitskritischer Bedeutung – Allgemeine Systemanforderungen, 10 2002. DIN IEC 61513.

fNor04. Deutsches Intitut für Normung. Kernkraftwerke – Leittechnik für Systeme mit sicherheitstechnischer Bedeutung – Softwareaspekte für rechnerbasierte Systeme zur Realisierung von Funktionen der Kategorien B oder C, 9 2004. DIN IEC 62138.

fNor06a. Deutsches Intitut für Normung. Funktionale Sicherheit sicherheitsbezogener elektrischer/elektronischer/programmierbarer elektronischer Systeme — Teil 1: Allgemeine Anforderungen, 7 2006. DIN IEC 61508-1.

fNor06b. Deutsches Intitut für Normung. Funktionale Sicherheit sicherheitsbezogener elektrischer/elektronischer/programmierbarer elektronischer Systeme — Teil 2: Anforderungen an sicherheitsbezogene elektrische/elektronische/programmierbare elektronische Systeme, 7 2006. DIN IEC 61508-2.

fNor06c. Deutsches Intitut für Normung. Funktionale Sicherheit sicherheitsbezogener elektrischer/elektronischer/programmierbarer elektronischer Systeme — Teil 3: Anforderungen an Software, 7 2006. DIN IEC 61508-3.

fNor06d. Deutsches Intitut für Normung. Funktionale Sicherheit sicherheitsbezogener elektrischer/elektronischer/programmierbarer elektronischer Systeme — Teil 4: Begriffe und Abkürzungen, 7 2006. DIN IEC 61508-4.

fNor07. Deutsches Institut für Normung. Kernkraftwerke – Leittechnik für Systeme mit sicherheitstechnischer Bedeutung – Softwareaspekte für rechnerbasierte Systeme zur Realisierung von Funktionen der Kategorie A, 8 2007. DIN IEC 60880.

GrCa87. Robert B. Grady und Deborah L. Caswell. *Software metrics: establishing a company-wide program*. Prentice-Hall. 1987.

LLCM+06. Jang-Soo Lee, Arndt Lindner, Jong-Gyun Choi, Horst Miedl und Kee-Choon-Kwon. Software Safety Lifecycles and the Methods of a Programmable Electronic Safety System for a Nuclear Power Plant. In Janusz Górski (Hrsg.), *Computer Safety, Reliability and Security — 25th international Conference, SAFECOMP 2006, Gdansk, Poland*, 2006, S. 85 – 98.

Reak96. Reaktorsicherheitskommision. RSK-Leitlinien für Druckwasserraktoren, 11 1996.

Tzsc06. Benjamin Tzschoppe. Safety-Einführung. In *Seminar Safety und Security Engineering im Sommersemester 2006 an der Universität Duisburg-Essen*, 7 2006.

Wiki07. Deutsche Wikipedia. Sicherheit, 2007. Stand vom 27.11.2007, 17:00.

Wiki08a. Deutsche Wikipedia. International Electrotechnical Commission, 2008. Stand vom 26.1.2008, 14:08.

Wiki08b. Deutsche Wikipedia. Kernkraftwerk, 2008. Stand vom 9.2.2008, 14:30.

Wiki08c. Deutsche Wikipedia. Kernschmelze, 2008. Stand vom 10.2.2008, 12:20.

Wiki08d. Deutsche Wikipedia. Reaktorsicherheitskommision, 2008. Stand vom 28.1.2008, 6:26.

Java EE Benchmarks

Jens Küttel

Betreuer: Michael Kuperberg

Abstract Java EE is a widely used platform for programming server-side component-based applications in the Java programming language. It is implemented in many different application servers which differ in performance and other characteristics. To compare the application servers with respect to performance, different competing benchmarks have been established. However, problems like optimisation of results or different approaches to benchmarking require a closer look on these benchmarks. In this contribution, we first describe the basic structure of Java EE, define the criteria for comparing Java EE implementations. Then, we discuss several Java EE benchmarks and compare them to each other.

1 Introduction

Today, Web applications become more and more important. Enterprises want deliver there customer for example the ability to get in contact with the enterprise, get information about the company, buy something online, etc. Many processes which were costly, inefficient or just impossible (24 hours shopping for example) are realised and optimised through Web applications. Java EE was developed to provide an infrastructure for Web applications. Through the usage of this infrastructure one gets more performant applications and avoids the design of the architecture every time. Java EE is a widespread component-oriented framework that helps to implement and run Java applications. There are many products, which are implemented to Java EE technology. Because there are many different vendors there are differences in the performance of the products. So there is a need of comparing and benchmarking of the performance of these products. This is the central topic of this contribution and will be explained and illustrated in detail in the following. Therefore, a basic knowledge for Java EE itself and afterwards corresponding benchmarks will be discussed and compared.

2 Fundamentals

To understand what is measured in Java EE benchmarking and to get a fundamental knowledge of the topic, a basic overview of the architecture itself will be given.

2.1 Architecture

As mentioned in [SunM07, p.50] the Java EE architecture consists of three components (also called containers) which are written in Java programming language.

- Application Client/Applet Container (client-side)
- Web Container (server-side)
- EJB Container (server-side)

Application Client/Applet Container Applets are embedded in a web page and are executed with the Java virtual machine (JVM). They allow client-side implementation of business logic. Application Clients run on a client system and can interact directly with EJBs over HTTP for example.

Web Container The Web Container consists of Java Server Pages (JSP) which are executed as Servlets but allow the developer to implement in a more web page like manner and Servlets which deliver direct responses to requests of clients.

EJB Container This container holds and executes the Enterprise Java Beans (EJB) which are elements of the needed business logic.

Conclusion The Java EE Framework delivers an architecture which can be adapted to the individual needs of a company. No redundant implementation of recurring code has to be done any more. "Java Platform, Enterprise Edition (Java EE) is the industry standard for developing portable, robust, scalable and secure server-side Java applications. Building on the solid foundation of Java SE, Java EE provides web services, component model, management, and communications APIs that make it the industry standard for implementing enterprise class service-oriented architecture (SOA) and Web 2.0 applications" [SunM07].

2.2 The Spring Framework

As described in [John05], today the Spring framework is the most popular framework for building enterprise Java applications. It has a modular architecture which means one don't have to use everything of the Spring framework but only parts one needs, for example parts for database usage. This leads to a very lightweight infrastructure and simplifies the implementation of Java EE applications. These aspects and many others helped Spring to become an accepted and often used approach for building web applications.

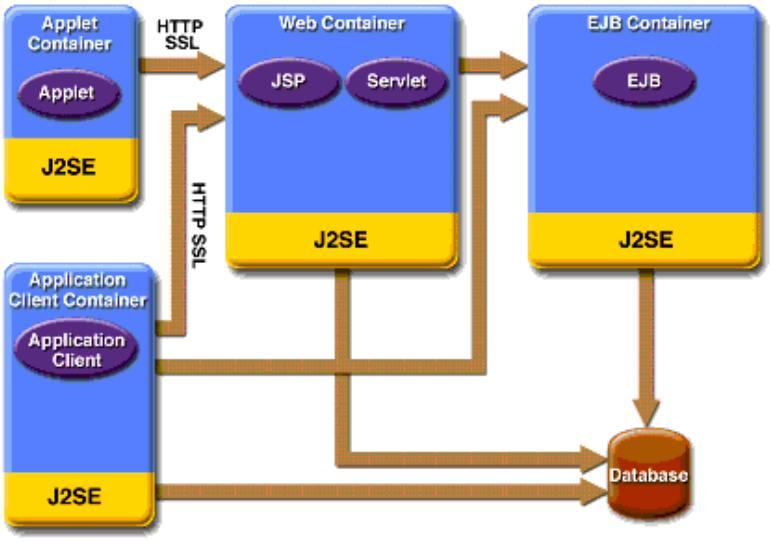


Figure 1. Java EE Architecture [SUN05]

2.3 Implementations of Java EE

There are many implementations of Java EE (open-source or proprietary) available today. A Google search delivers some of these server implementations:

Open Source Server

- Apache Geronimo (uses optionally Servers like Apache Tomcat or Jetty)
- JBoss Application Server (Apache Tomcat)
- JOnAS (uses Apache Tomcat)
- GlassFish (uses Apache Tomcat)
- Enhydra Enterprise (uses Apache Tomcat)
- Sun Java System Application Server

Proprietary Server

- ATG Dynamo Application Server (DAS)
- BEA WebLogic
- IBM WebSphere
- SAP Web Application Server (SAP NetWeaver Application Server)
- Oracle Application Server
- Orion Application Server

The most wide-spread server implementations are BEA WebLogic, IBM WebSphere and JBoss Application Server.

3 Criteria for Benchmarking

To compare different benchmarks, criteria have to be defined. According to [Wolf02, p.40] aspects which can be benchmarked and compared against each other are the following ones...

- **Response time** (time to complete a transaction)
- **Throughput** (transactions per time unit)
- **Resource utilisation** (utilisation of RAM and CPU for example)

But how and where are these factors determined? The parts of a Java EE implementation which have to be considered are

- **Transaction management**
- **Naming and directory services**
- **Communication between the Java EE components**
- **Management of the components itself**
- **Single components or the whole architecture**

Furthermore [Wolf02, p.41] specifies in more detail how a comparison of systems can be achieved. First of all the mentioned criteria can be further refined with minimal, average and maximal values. For example the average response time is defined as sum of all response time divided by n (n the amount of response time values). Minimal and maximal values are self-explanatory. So to compare systems [Wolf02, p.41] defines the following.

$$\frac{execution\_time_a}{execution\_time_b} = r \tag{1}$$

$$r = \frac{execution\_time_a}{execution\_time_b} = \frac{\frac{1}{performance_b}}{\frac{1}{performance_a}} = \frac{performance_a}{performance_b} \tag{2}$$

Additionally aspects like the following are interesting and will be considered in the following parts.

- Who measures?
- For whom one measures?
- Costs for the benchmark? Free?
- License?

3.1 Benchmarking of Java EE applications

In [Wolf02, p.42], it is explained how the principal benchmarking of Client/Server applications is done. So basic components are:

- **Driver** and
- **Application on Server**.

The Driver generates workload for the Server with many requests at the same time. Benchmarking is done through reporting of the results that means how good the performance is of the server.

3.2 Challenges

As described in [Petr01, p.1] there have to be met some prerequisites. These are necessary to compare results and to abstract of individual characteristics of the benchmarks. [Petr01, p.1,2] gives the following prerequisites:

- **Test data** which describes real user behaviour
- **User-Profile**, the characteristics of users
- **Reproducible baseline** which means the benchmark can be repeated in the same way
- **Definition of SUT (System under test)** which means to determine what have to be tested
- **Load-test-server** for controlling of client
- **Realistic environment** which means a basic load in network
- **Test drivers and scripts**

As the prerequisites show the complexity benchmarking must not be under estimated. It is a task which needs skill and planning to have useful results.

4 Benchmarking

As mentioned before there are many implementations of the same architecture Java EE. Many vendors deliver implementations and claim to have the most performant and feature-rich realization. But which one should one choose, which one is the most performant for an application(s). This leads to a comparison of the different available products. In the following different benchmarking products will be beheld and evaluated.

4.1 IBM's Performance Benchmark Sample (Trade2)

[Yan 03, p.1,2] IBM's Performance Benchmark includes Web Primitives and a trade application. With the help of Web Primitives one can benchmark elementary parts of the server like the JSP engine, the Servlet engine, EJB entities, Session Beans or the HTTP support itself. The trade application on the other side allows a more complex testing of the server. It consists of Servlets, EJBs and many more constructs to realize more and heavier workload in a more realistic and user-specific way.

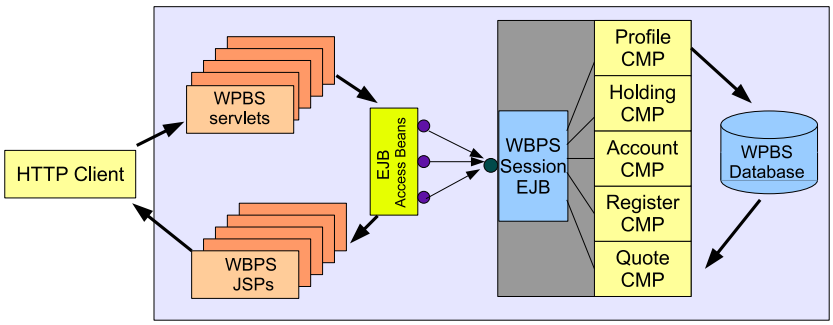


Figure 2. Trade2 architecture [Yan 03, p.2]

Benchmark Fig. 2 shows the Trade2 architecture and the user available operations. [Yan 03, p.2] summarizes the possible operations a user can execute via web browser as follows.

- **Register**: A User can create a Profile, an ID + password and a initial account balance at Trade2
- **Login**: After setting up an Account the User would like to Login and use the Account
- **Browse**: The User wants to browse the actual stock price for example and many other trade-specific options
- **Update**: Credit Limitations are possible and changeable
- **Buy**: Buying of an amount of stocks
- **Sell**: Selling of an amount of stocks
- **Logout**: The User wants to leave at the end of his transactions

To evaluate a server's performance [Yan 03, p.2] uses the Microsoft Web Stress Tool to emulate user web page hits and operation usage and Trade2 for more realistic and detailed workload which comes through the orchestration of operations. It's being distinguished between lighter and heavier database usage, EJB database access or direct database access through JDBC.

Reporting As you can see in Figure 3 lighter database usage leads in this case to a better throughput whereas more database usage decreases the amount of fulfilled requests per seconds. TTFB (time to first byte) and TTLB (time to last byte) decreases in DB-Light mode whereas they increase in high database usage. The other modes like the Standard mode simulates a standard workload, the uniform mode a workload in which every operation is equal times used and the No-Register mode without user registrations.

In case of direct access of the database the amount of requests fulfilled per second is about the same as with EJB access. The EJB indirection adds an offset but there are no differences visible between the different kind of modes.

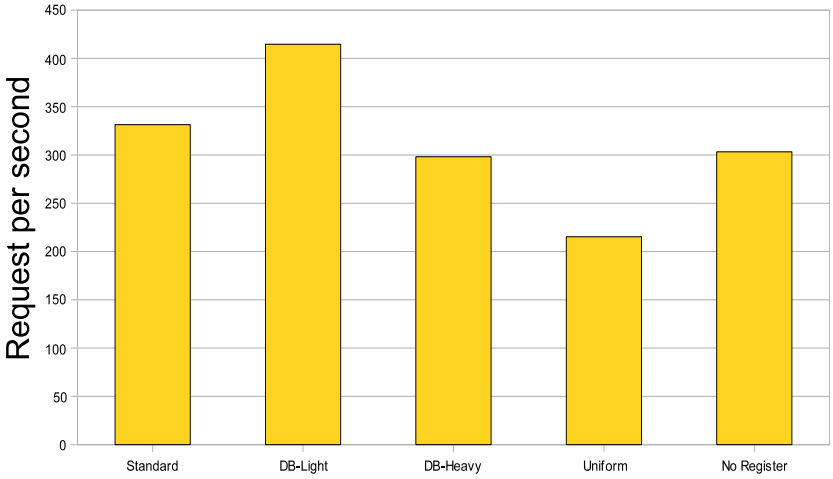


Figure 3. EJB-DB access mode [Yan 03, p.4]

4.2 Sun's ECperf

[Yan 03, p.2] Sun Microsystems and the Java community also developed a benchmark for performance and scalability of servers.

Benchmark The aim of this benchmarking concept is to simulate enterprise processes and the workload which derives from them. Figure 5 shows the involved parties (customer, corporation, manufacturing and supplier). The driver and supplier component are used to simulate producer-consumer relationship activities, relationship activities inside the ECperf application and relationship activities to the outside. They are not part of system under test (SUT). Reporting of results is done by the driver.

[Yan 03, p.3] identifies the following workloads:

Workload through corporation

- **AddCustomer**
- **HasSufficientCredit**
- ...

Workload through customer (Figure 6, 7)

- **NewOrder**
- **ChangeOrder**
- **GetOrderStatus**
- ...

Workload through manufacturing (Figure 8, 9)

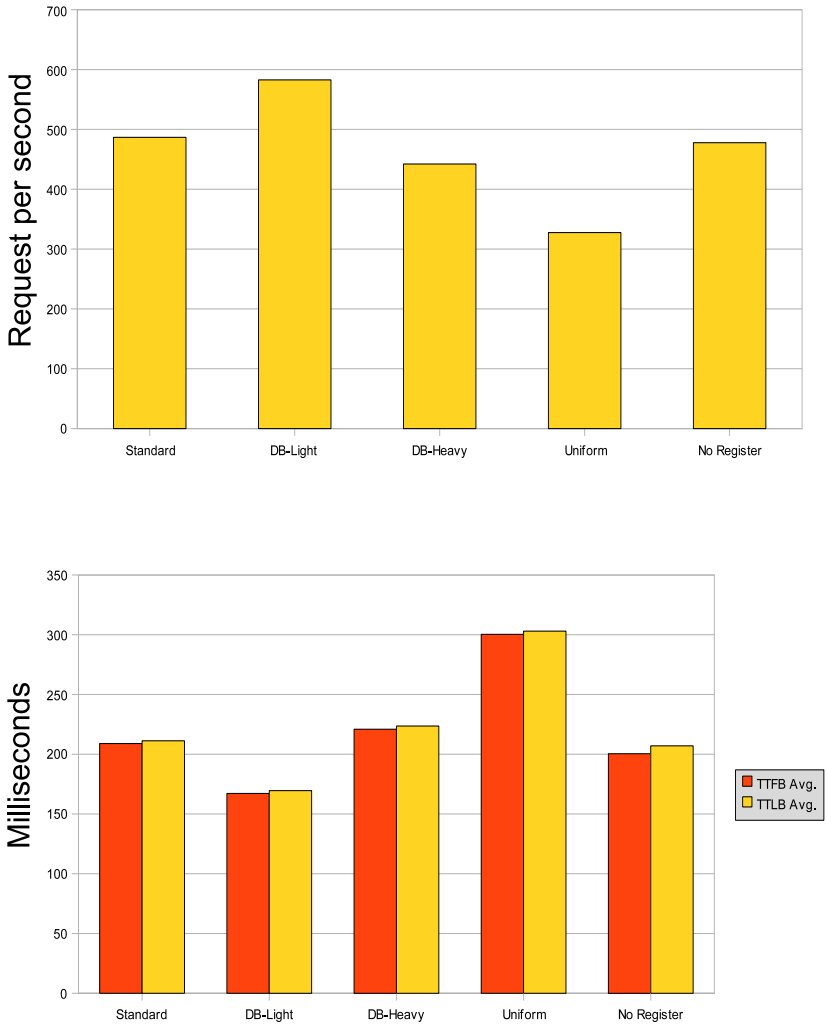


Figure 4. JDBC-DB access mode [Yan 03, p.4]

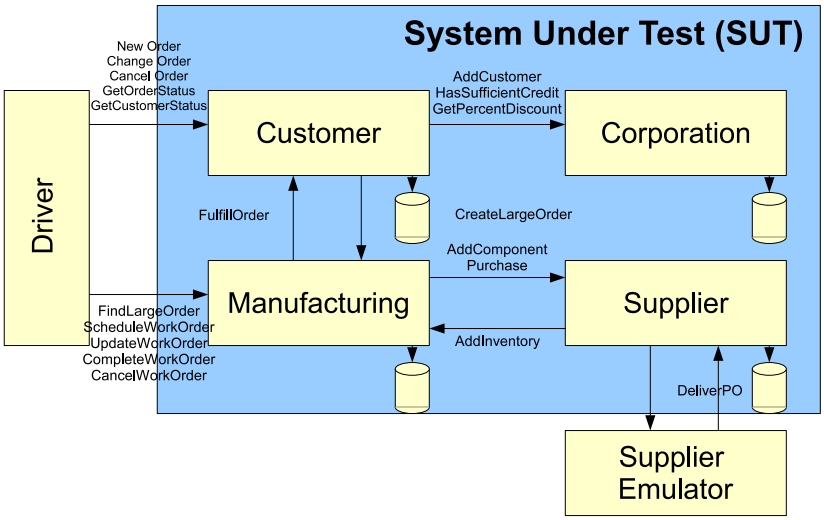


Figure 5. ECperf architecture [Yan 03, p.2]

- **ScheduleWorkOrder**
- **UpdateWorkOrder**
- **CompleteWorkOrder**
- ...

Workload through supplier

- **AddComponent**
- **Purchase**
- **DeliverPO**
- ...

Reporting The following diagrams shows the frequency distribution of different performance aspects. As one can see in figures 6 and 7, new orders and work orders have low response times for a high amount of transactions because of caching effects for example. Figure 9 shows that the exemplary test system is able to process 800 til 900 work orders constantly.

The time-range for the response times is between milliseconds and about two seconds.

Comparison Trade2 vs. ECperf Both concepts are open and so usable to test almost every Java EE implementation. There are differences in how the workload is generated. Trade2 needs an extra tool (in this case the Microsoft web stress tool) whereas ECperf has its own generator (driver, supplier). But to make it possible to compare ECperf-tested systems the driver has to be described in detail. Problems occur in the interpretation of the results. Trade2 depends

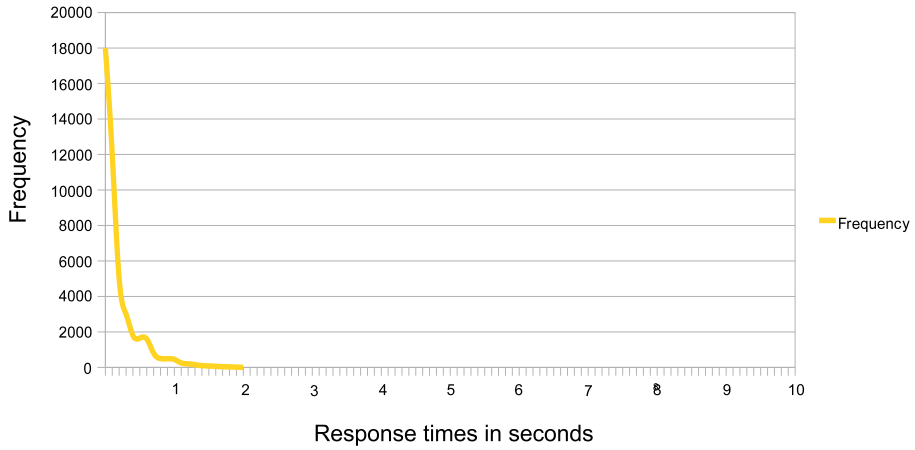


Figure 6. Frequency Distribution (New Orders) [Yan 03, p.5]

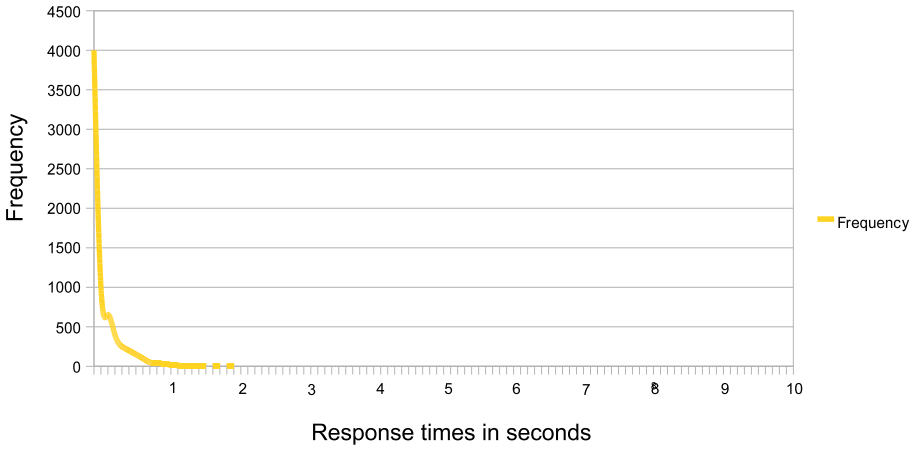


Figure 7. Frequency Distribution (Customer Status) [Yan 03, p.5]

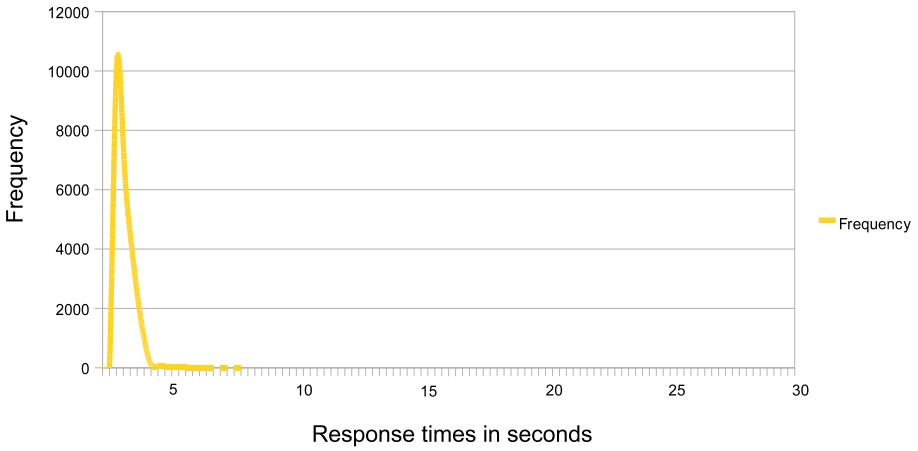


Figure 8. Frequency Distribution (Work Orders) [Yan 03, p.6]

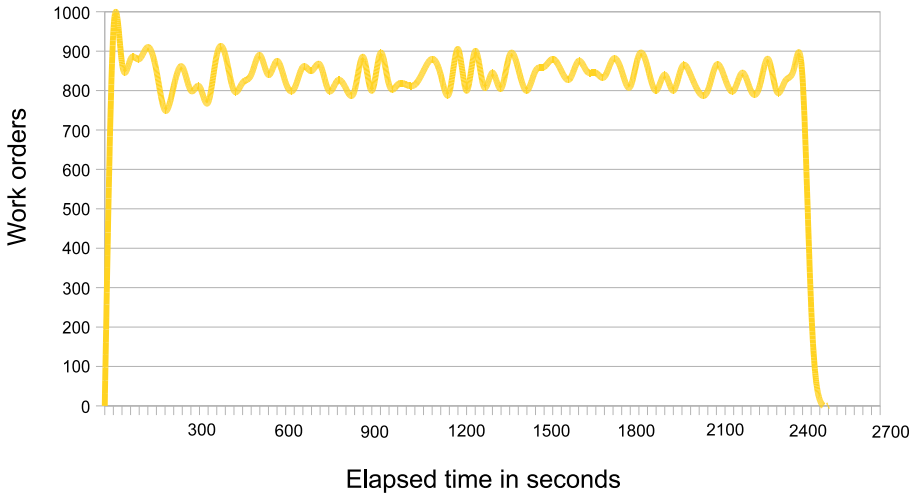


Figure 9. Manufacturing Workorder Throughput [Yan 03, p.6]

on the web test load generator and the runtime mode. Because there are no guidelines for the result reporting misinterpretation can occur. ECperf on the other side defines such guidelines and prevent the misinterpretation of benchmark results. These two benchmarks are the most popular ones.

4.3 SPECjAppServer2004

The SPECjAppServer2004 [Corp] benchmark is a further benchmark established by the Standard Performance Evaluation Corporation.

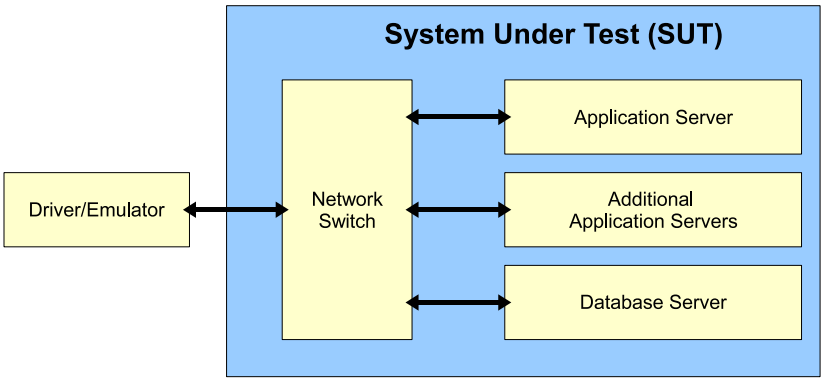


Figure 10. SPECjAppServer2004 Illustration [Corp06]

Terms [Corp] defines some terms which are used in the following.

- **SPECjAppServer2004 JOPS** - Metric for comparison
- **EAStress2004** - test workload
- **SPECjAppServer2004 Kit** - complete benchmarking kit
- **Driver** - benchmark driving force
- and many more

Benchmark SPECjAppServer2004 includes a business model which represents a common business environment and common domains.

As you can see in Figure 11 domains can be classified. The Dealer Domain give a web user the opportunity to interact via web interface. This is the starting point of workload generation. The Customer Domain represents intuitively the customer administration. The Supplier Domain tries to handle the given suppliers whereas the Corporate Domain handles information about customer, supplier and products. Finally the Manufacturing Domain which stands for performing of operations which have to be done ”just-in-time”. Each of these Domains is

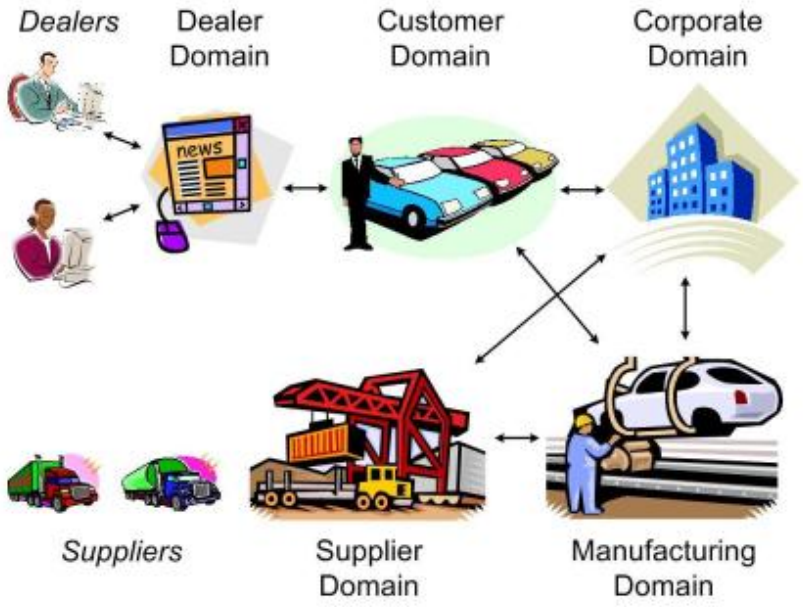


Figure 11. SPECjAppServer2004 Business Model [Corp06]

implemented through Java components except of the Supplier Domain, which is realized through an Supplier emulator. Like in the ECperf benchmark test a Driver called SPECjAppServer Driver is used. Three basic operations can be figured out. The Manufacturing stands for a vehicle production company and so the operations are the following [Koun05]:

- **Purchase** - purchasing vehicles
- **Manage** - manage customer needs
- **Browse** - look up cars

SPECjAppServer2004 needs a driver configuration to test the performance and stability of the target system. Drivers can be divided in two categories. The Master machine is the driving force and generates workload. The Agent machines on the other side are the machines which executes the workload. [Corp] defines different applications, different Agents for application execution and a controller which runs the Master machine. Central part of the benchmarking environment is the Driver. Its job is to extract the test properties which are defined in advance and sets up the independent Agents for the work they have to do. Finally the Driver collects and aggregates the execution-reports from the Agents.

Reporting As described in [Corp] reporting of results consists of three steps. First of all you have to generate a submission file. This submission file starts

with the description of the system-under-test (SUT).
The SPECjAppServer2004 reporter generates a document **result.props** which includes the final results. These results are the second part of the submission file. The second step is to generate a report with help of SPECjAppServer2004 reporter. Finally this report can be sent to the SPEC OSG Java subcommittee for supporting there effort to collect, aggregate and evaluate benchmark results.

SPECjAppServer2004 uses many Client Agents with Java Virtual Machines installed so not only performance but the scalability can be tested, too. Similar to ECperf a Driver component is part of the benchmark. Tuning of the parameters offered by the Driver allows testing the server for bottlenecks and compare against other implementations.

The SPECjAppServer2004 benchmark can be used to stress intensively the JMS and MDB infrastructure. It is only free for SPEC members [Corp].

Tester	System	JOPS	J2EE Server Nodes	J2EE Server Instances	J2EE Server CPU Description	DB Server Instances
1	BEA Systems, Inc	BEA WebLogic Server 9.0 on HP DL380 Cluster	1374,11	5	10 Cores, 10 Chips	1
2	BEA Systems, Inc	BEA WebLogic Server 9.0 on HP DL380 Cluster	1664,34	6	12 Cores, 12 Chips	1
3	Fujitsu Limited	BEA WebLogic Server 9.2 on Fujitsu SPARC Enterprise T2000	801,7	1	8 Cores, 1 Chip	1
4	Fujitsu Limited	Oracle Application Server 10g Release 10.1.3.3 – Java Edition on Fujitsu SPARC Enterprise T5220	2000,92	1	8 Cores, 1 Chip	1
5	HP	BEA WebLogic Server 9.0 on HP-UX rx4640	471,28	1	4 Cores, 4 Chips	1

Figure 12. SPECjAppServer2004 Benchmark results [Koun05]

4.4 TPC-C

TThe TPC (Transaction Processing Performance Council) is a non-profit corporation founded to define transaction processing and database benchmarks and to disseminate objective, verifiable TPC performance data to the industry' [TPC07]. [Fran07] gives an overview about TPC-C. So it is the third generation for benchmarking "[...]performance and price/performance of transaction processing systems".

Benchmark [Naum07, p.17-20] describes that the TPC-C benchmark models a warehouse with orders in which five transactions can be executed by terminal operators.

- **New-order** (through customer)
- **Payment** (actualize customer account)
- **Delivery** (of orders)
- **Order-status** (search latest status of order)
- **Stock-level** (observing of inventory)

Reporting The performance is measured in transactions per seconds and the Price/Performance ratio in Dollar/Transaction. Diagram 13 shows the comparison of different systems. TPC-C is free for anyone.

4.5 TPC-W

As described in [Smit01, p.1,2] TPC-W simulates like most of the benchmarking tools an E-Commerce environment in which transactions can be done by simulated customers. Figure 14 shows the scenario.

Benchmark A RBE (Remote Browser Emulator) as mentioned in [Smit01, p.1,2] simulates a real customer. There are 14 possible websites which can be included in the benchmark. So there are no major differences to real E-Commerce websites. There is a Home page an pages which lists many products and product offers. New users have to register first and after that they can be active in the system. To do a benchmark [Wolf02, p.43] gives the following metrics.

- **WIPS** (Web interactions per second)
- $**WIPS** (amount of dollars per WIPS)
- **WIPSO** (Web interactions per second ordering)
- **WIPSB** (Web interactions per second browsing)

For WIPS the ratio between browsing interactions and ordering interactions is 4:1. This is the assumed surfing behaviour of a standard customer.
$\WIPS is the amount of dollars of the whole project divided by WIPS. This is the ratio of price and result. WIPSO includes a 1:1 ratio of browsing and ordering interactions and WIPSB includes only 10% ordering activities.

Rank	Company	System	tpmc	Price/tpmc	Database
1	HP	HP Integrity Superdome-Itanium2/ 1.6Ghz/24MB iL3	4092799	2,93 US $	Oracle Database 10g R2 Enterprise Edition w/Partitioning
2	IBM	IBM System p5 595	4033378	2.97 US $	IBM DB2 9
3	IBM	IBM eServer p5 595	3210540	5,07 US $	IBM DB2 UDB 8.2
4	IBM	IBM System p 570	1616162	3,54 US $	IBM DB2 Enterprise 9
5	IBM	IBM eServer p5 595	1601784	5,05 US $	Oracle Database 10g Enterprise Edition
6	Fujitsu	PRIMEQUEST 540 16p/32c	1238579	3,94 US $	Oracle Database 10g Enterprise Edition
7	HP	HP Integrity Superdome-Itanium2/ 1.6GHz-64g/64c	1231433	4,82 US $	Microsoft SQL Server 2005 Enterprise Edition SP1
8	HP	HP Integrity rx5670 Cluster-Itanium2/1.5 GHz – 64p/6	1186893	5,52 US $	Oracle Database 10g Enterprise Edition
9	IBM	IBM eServer pSeries 690 Model 7040-681	1025486	5,43 US $	IBM DB2 UDB 8.1
10	IBM	IBM System p5 570 Model 9117-570	1025169	4,42 US $	IBM DB2 UDB 8.2

Figure 13. TPC-C System comparison [Naum07, p.28]

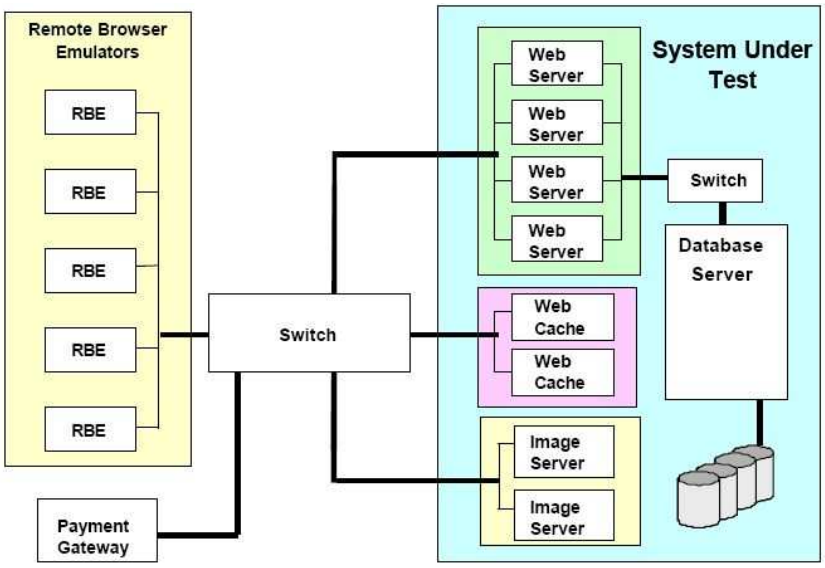


Figure 14. TPC-W System [Smit01, p.5]

Reporting As described by [Wolf02, p.43] with WIPSB and WIPSO are results for an extreme load distribution and so are not the most important results. More important results are $\WIPS and WIPS. They emulate real user behaviour and so they includes the most useful information. TPC-W is free for anyone.

4.6 SPECweb2005

How described in [SPEC05] SPECweb2005 is another 3-workload benchmark tool to benchmark the application server performance. To make results comparable exactly defined environment attributes have to be kept (Further details to attributes see [SPEC05]).

Filesets Filesets are representations of different workloads which can be used to benchmark the system. Example filesets:

- **Banking fileset** (all requests use HTTPS (SSL))
- **E-commerce fileset** (HTTP and HTTPS requests)
- **Support site fileset** (only HTTP requests)

Benchmark SPECweb2005 has two metrics the primary metric for the overall performance and the workload submetrics for the workload specific measurement. [SPEC05] defines the primary metric the following way "The primary metric, SPECweb2005, is the geometric mean of the ratio of each of the workload submetric scores to their respective workload reference scores, with that result multiplied by 100". Workload submetrics are measured in amount of simultaneous sessions which can be handled keeping the benchmark quality-of-service (QOS) requirements.

SPECweb2005 can be purchased on CD-ROM from SPEC at $1,200 for new licensees, $600 for upgrades, and $300 for eligible non-profit organizations.

4.7 Empirix Bean-Test

Empirix Bean-Test is a benchmark for BEA WebLogic, IBM WebSphere, Bluestone's Total-e-Server application servers. It allows to benchmark Java Beans as well as the servers on which the application is hosted on.

Benchmark & Reporting First a project has to be generated. After that one has to specify the servers on which the Beans have to be tested. The Java Beans are found automatically. If the project is generated correctly client code for test cases is available. To run a test case one has to set up clients with the accordant code. Reporting is done by test results which can be compared assumed the same tests are chosen before. [John01] for more details.

4.8 pBOB

As described in [IBM] TPC-C is the originator of pBOB (portable Business Object Benchmark) but without being a TPC-C benchmark. The benchmark is a Java implementation with a Java emulation of the database which is accessed during the execution. With pBOB one can benchmark server-side business applications.

4.9 jBOB

jBOB has the intention to benchmark the speed of an implementation and prescind of the throughput. It's origin is also TPC-C and the difference to pBOB is how mentioned in [Wolf02, p.45] the focus on database systems and transactional controlled implementations. Literature for pBOB and jBOB is very rare.

4.10 RUBiS

[Wolf02, p.46,47] adduces RUBiS (Rice University Bidding System) as a further benchmark example. It is a benchmark which modelise an auction house like Ebay and is implemented in three ways (PHP, Java Servlets, EJBs).

Benchmark Most important activities which are modeled are the following ones:

- **Selling**
- **Navigation**
- **Bidding**

[Wolf02, p.46] mentions seven tables on which transactions are executed.

- **Users**
- **Items**
- **Categories**
- **Regions**
- **Bids**
- **Buy_now**
- **Comments**

Reporting Performance is measured by throughput in requests/min. Fig. 15 shows exemplarily the results for the transaction **browsing**.

RUBiS is licensed under the GNU Lesser General Public License.

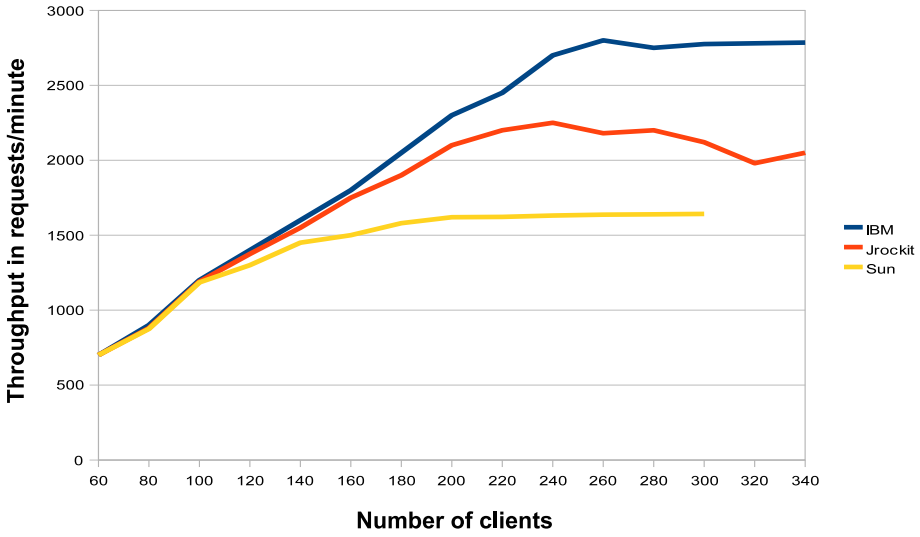


Figure 15. RUBiS **browsing** [Obje04]

5 Conclusion

As mentioned Java EE delivers companies many opportunities to support their business and make the technical realisations of their business processes more efficient. Therefore fast and reliable Java EE software solutions are needed. These characteristics have to be benchmarked. Many benchmarks have been mentioned and similarities can be seen. Most of the benchmark tools try to simulate an E-Commerce environment with interactions like ordering, browsing, registering and many others. Each of them try a mix of this interactions and map the measured behaviour in key indicators which should allow a comparison. The different mixes of interactions makes it hard to compare the results. As mentioned in the beginning, benchmarks are done mostly by the company which realized the corresponding Java EE product. So they try to show the results in a way which supports selling of the product. The mix of interactions has a major influence of the benchmark results and gives opportunity to increase advantages of the product and to veil disadvantages. The question which benchmark to choose is case dependent. For the comparison of warehouse-like architectures ECperf, Trade2, TPC-C and TPC-W are the appropriate benchmarks. jBOB and pBOB can be used for applications with additional heavy database usage whereas RUBiS, Empirix Bean-Test, SPECjAppServer2004 and SPECweb2005 are benchmarking more common domains.

References

Corp. The Standard Performance Evaluation Corporation. SPEC Organizational Information. http://www.spec.org/.

Corp06. Standard Performance Evaluation Corporation. *SPECjAppServer2004 User's Guide*. Standard Performance Evaluation Corporation, November 2006.

Fran07. Amitabh Shah Francois Raab, Walt Kohler. *Overview of the TPC Benchmark C: The Order-Entry Benchmark*. TPC, 2007.

IBM. IBM. Portable Business Object Benchmark (pBOB). http://www.research.ibm.com/jasper/PBOB.html.

John01. Cedrick W. Johnson. Product Review: Bean-Test 3.1. *JAVA DEVELOPER'S JOURNAL*, 2001, S. 1.

John05. Rod Johnson. Introduction to the Spring Framework 2.5. *ServerSide.com*, Band 1, 2005, S. 1.

Koun05. Samuel Kounev. SPECjAppServer2004: The New Way to Evaluate J2EE Performance. *Dev2Dev*, 2005, S. 1.

Naum07. Felix Naumann. Datenbanksysteme II Benchmarks, June 2007.

Obje04. ObjectWeb-Consortium. JVM evaluation results with RUBiS. http://jmob.objectweb.org/jvm/RUBiS.html, 2004.

Petr01. Roland Petrasch. Empfehlungen zum Thema Performance-Tests. Technischer Bericht, FH Nordakademie, February 2001.

Smit01. Wayne D. Smith. TPC-W*: Benchmarking an E-Commerce Solution. Technischer Bericht, Intel Corporation, October 2001.

SPEC05. SPEC. SPECweb2005. Technischer Bericht, SPEC, 2005.

SUN05. SUN. J2EE Architecture. Technischer Bericht, SUN, December 2005.

SunM07. Inc. SunMicrosystems. *The Java EE Tutorial - For Sun Java System Application Server 9.1.* SunMicrosystems, Inc., 4150 Network Circle Santa Clara, CA 95054 U.S.A., September 2007.

TPC07. TPC. About the TPC. http://www.tpc.org/information/about/abouttpc.asp, 2007.

Wolf02. Thomas Wolf. Benchmark für EJB-Transaction und Message-Service. Technischer Bericht, Universität Oldenburg, 2002.

Yan 03. Wei Qu Yan Zhang, Anna Liu. Comparing Industry Benchmarks for J2EE Application Server: IBM's Trade2 vs. Sun's ECperf. Technischer Bericht, Australian Computer Society, Inc., 2003.

Zertifizierung von Softwarebenchmarks

Jakob Blomer

Betreuer: Michael Kuperberg

Zusammenfassung Durch Softwarebenchmarks soll sich für den Anwender die Leistung einer Software bestimmen lassen. Da Benchmarks eine Vielzahl von Parametern und Annahmen enthalten, besteht die Gefahr der Fehlinterpretation der Meßergebnisse. Benchmarks, die nicht vom Anwender selbst erstellt werden, müssen daher vertrauenswürdig und dokumentiert sein, um aussagekräftig zu bleiben. Die Benchmarkmethodik ist durch die Normen DIN 66273 und der darauf aufbauenden ISO/IEC 14756 standardisiert. Anstatt eines Akkreditierungs- und Zertifizierungsprozesses nach diesen Standards haben sich als Quasi-Zertifikate für Softwarebenchmarks die Benchmarksuiten von Herstellerkonsortien etabliert. Darunter betrachtet die Arbeit exemplarisch die SPEC- und TPC-Benchmarksuiten.

1 Einführung

Benchmarks sind ein Instrument zur Leistungsbewertung eines Softwaresystems. Sie messen unter den nicht-funktionalen Eigenschaften einer Software ihre Leistung, d.h. wie stark die Ressourcen der Ausführungsumgebung bei der Erledigung einer Aufgabe durch die Software beansprucht werden. Bestenfalls läßt sich durch Benchmarkergebnisse der Wert einer Software bestimmen. Wenn z.B. ein Wetterdienst ein Rechenzentrum anmietet und Software $A++$ einen doppelt so schnellen Algorithmus zum Lösen von Differentialgleichungen verwendet wie Software A, dann kann die durch die verkürzte Laufzeit gesparte Miete direkt mit dem Preis der Softwarepakete verrechnet werden.

Bei komplexer oder nicht frei erhältlicher Software ist es für mögliche Anwender aufwendig, Benchmarks selbst durchführen. Er ist daher auf vertrauenswürdige Benchmarkergebnisse Dritter angewiesen. Da Benchmarkergebnisse hier als Kaufkriterium verwendet werden, besteht ein natürliches Interesse der Hersteller an guten Werten der eigenen Software (und schlechten Werten der Konkurrenzsoftware), das sich beispielsweise im Zurechtschneiden („Tunen“) der Software auf einen Standardbenchmark äußern kann. Es besteht also Bedarf an standardisierten Benchmarkmethoden und danach zertifizierten Benchmarks.

Ein Benchmark liefert als Ergebnis eine oder mehrere Zahlen, die vom Anwender zu interpretieren sind. Diese Interpretation ist i. a. einzelfallabhängig und unabhängig von der Zertifizierung des Benchmarks. Dennoch sollten die Werte eines guten Benchmarks nicht der intuitiven Interpretation widersprechen. Wenn also der SPEC-CPU-Wert für einen Übersetzer deutlich höher ist als der der

Konkurrenz, sollte der im Praxisbetrieb produzierte Code tatsächlich „ein gutes Stück“ schneller sein.

Nach einem Überblick über Benchmarks als Teilgebiet der Leistungsevaluation und den einschlägigen DIN/ISO-Normen betrachtet Abschnitt 4 praktische Benchmark-Regelwerke. Im Bereich der Server- und Systemsoftware haben sich insbesondere die SPEC- und TPC-Suiten durchgesetzt [Idea00].

2 Benchmarkumfeld

„The word benchmark is taken from the marks showing measures like feet or inches on the work bench of former craftsman“[Dirl06, 1.1]. Beim Benchmarking handelt es sich also um das Handwerk des vergleichenden Messens, wobei wir im speziellen *Leistungs*messungen meinen. Diese Arbeit betrachtet „Benchmarking“ nicht im Sinne der Leistungsmessung von Entwicklungsprozessen – diese Sicht findet sich beispielsweise bei Ebert und Dumke [EbDu07, Kapitel 11] und bei der International Software Benchmarking Standards Group (ISBSG). Wir betrachten die Leistung der Software selbst. Ein Softwarebenchmark im Sinne dieser Arbeit läßt sich jedoch auf natürliche Weise als Spezialfall eines *software measurement system* auffassen, wie es bei Ebert und Dumke definiert ist [EbDu07, 3.2].

Nach Ferrari [Ferr78] zählt Benchmarking neben Simulation und Modellierung zur Leistungsevaluation, genauer zu den Meßmethoden der Leistungsevaluation. Zwar gelten für Softwarebenchmarks besondere Bedingungen, ein Softwarebenchmark läßt sich aber nicht von der ausführenden Hardware trennen. Softwarebenchmarks messen also den Verbund aus Hardware- und Softwaresystem.

Definition 1. *Als* Softwarebenchmark *bezeichnen wir die Leistungsmessung eines Softwaresystems bezüglich festgelegter* Indizes.

Definition 2. *Ein* Leistungsindex *beschreibt den Verbrauch einer Ressource bei der Ausführung der zu messenden Software.*

Insbesondere für Softwarebenchmarks schlägt Ferrari diese Form der am Ressourcenverbrauch orientierten Leistungsdefinition vor [Ferr78, 9.2.1]. Eine Kostenfunktion $c : (I_1, \ldots, I_n) \ni (v_1, \ldots, v_n) \mapsto r \in \mathbb{R}$, die einer Menge von Verbrauchswerten v_i zu festgelegten Indizes I_i einen reellen Wert zuweist, stellt dann das Bindeglied zwischen den technischen Leistungsindizes und einem ökonomischen Index dar. Tabelle 1 stellt gängige Meßindizes zusammen.

Bei Standardsoftware liegt der Fokus auf der Geschwindigkeit oder Indizes, die sich in Geschwindigkeit umrechnen lassen. Zeitintervalle haben den Vorteil, daß *externe* Ereignisse gemessen werden können (Zeit), also Ereignisse, deren Quelle nicht im System selbst liegt. Sie sind daher gegen Verfälschungen unempfindlich. Diese Unterscheidung hat praktische Relevanz; so läuft beispielsweise die Systemuhr in einigen virtuellen Maschinen langsamer als die Echtzeit [Blom07]. Der Index Auslastung zählt bei Ferrari zu den sekundären Indizes, d.h. Indizes

Index	Einheit	Bedeutung
Geschwindigkeit	Zeit	Dauer, bis eine Dateneingabe verarbeitet wurde
Verbrauch	ressourcenabh.	Absoluter Verbrauch von Ressourcen, z.B. Energie, Speicher, ...
Durchsatz	Zeit^{-1}	Anzahl Dateineingaben, die in einer Zeiteinheit verarbeitet werden können
Kapazität	Zeit^{-1}	Maximal möglicher Durchsatz
Latenz	Zeit	Dauer zwischen Dateneingabe und Beginn der Verarbeitung
Turnaround	Zeit	Minimale Dauer zwischen zwei Dateneingaben
Antwortzeit	Zeit	Dauer zwischen Dateineingabe und einer zugehörigen Ausgabe (die Eingabe muß nicht vollst. verarbeitet sein)
Auslastung	%	Quotient aus der Gesamtmenge einer Ressource und dem der Software zugeordneten Anteil

Tabelle 1. Gängige Leistungsindizes (nach Ferrari [Ferr78, 1.4])

die nicht direkt die Leistung angeben, sondern Hinweise auf Leistungspotentiale geben können [Ferr78, 1.4]. Aus diesem Grund werden sie in der ISO-Norm nicht mehr zu den Leistungsindizes gezählt [Dirl06, 1.2].

Methodisch lassen sich Benchmarks klassifizieren nach

1. Der betrachteten Komponente, also Prozessor, Speicher, Software, ...
2. Der Art der Eingabedaten: hier können neben Real-Daten auch synthetische Daten, die bestimmte Anwendungsmuster repräsentieren sollen, verwendet werden.
3. Der Granularität der Messung, wir unterscheiden zwischen Microbenchmark und Anwendungsbenchmark

Anwendungsbenchmarks haben gegenüber Microbenchmarks den Vorteil, daß die Gesamtleistung nicht aus den Leistungsdaten der einzelnen Komponenten zusammengesetzt wird; die Kumulation von Einzelleistungen ist für die Gesamtleistung ein schwacher Schätzer [Dirl06, 1.2]. Im Gegensatz zum Monitoring, das zur Überwachung produktiver Systeme verwendet wird, wird ein Benchmark an einem Testsystem und üblicherweise als vorbereitende Maßnahme durchgeführt.[1] Die Korrektheit des Systems setzen wir voraus, d.h. Test und Verifizierung sind keine Bestandteile eines Benchmarks. Im Umfeld der Leistungsevaluation zeichnet sich die Benchmarktechnik durch Messung am System selbst aus, während Simulationstechniken und analytische Methoden mit Modellen des Systems arbeiten.

[1] Benchmarks können auch als Entscheidungsunterstützung während des Softwareentwurfs oder zur Leistungsverbesserung bestehender Systeme verwendet werden. Diese Anwendungen sind aber im Rahmen von Benchmark*zertifizierung* irrelevant.

Die eigentliche Messung wird durch eine Reihe von Parametern festgelegt [Ferr78, 1.2]. Zunächst bestimmen die *Systemparameter* das zu messende Softwaresystem, z.B. Plattform und funktionale Eigenschaften. Die Konfiguration der Systeme wird durch die *Installationsparameter* festgehalten. Installationsparameter sollen „kleine“ Änderungen erfassen, obwohl prinzipiell jede Konfigurationsänderung ein neues System produziert. Die nicht system-inhärenten Parameter bestehen aus den Meßindizes, der Wahl der Meßwerkzeuge und den Eingabedaten.

Die Meßwerkzeuge zeichnen sich bestenfalls durch hohe Präzision und geringe Interferenz mit dem zu messenden System aus. Insbesondere verfälscht eine grobkörnige (System-)Uhr die Messung schneller Operationen. Meßwerkzeuge können von Instrumentierungs-Infrastruktur unterstützt werden, beispielsweise speziellen Prozessorregistern, die ticks, page-faults, usw. speichern. Messungen für nicht-zeitliche Ressourcen können durch einen Interpretierer ersetzt werden, der den Verbrauch jeder Instruktion errechnet [Ferr78, 9.3.1].

Die Eingabedaten bestehen aus einer kleinen, repräsentativen Menge der Arbeitslast. Die Arbeitslast eines Softwaresystems besteht i. a. aus allen möglichen Eingabedaten. Die Auswahl der repräsentativen Menge birgt die Gefahr zu weitreichender Annahmen. In manchen Fällen ist es möglich, Dateneingaben aus der Praxis aufzuzeichnen. Zum Beispiel können die Anfragen an eine Datenbank in einer Bank über eine Woche mitgeschnitten werden. Wir sprechen in diesem Fall von einem (Realdaten-)*Schnappschuß*. Ansonsten sind wir auf synthetische Daten angewiesen, z.B. eine Auswahl an Standardalgorithmen für Prozessor- und Übersetzerbenchmarks. Formal muß ein Lastmodell erstellt werden, das

- repräsentativ,
- einfach zu konstruieren,
- kompakt,
- system-unabhängig und
- reproduzierbar

ist [Ferr78, 5.1]. Aus diesem Modell werden Äquivalenzklassen gebildet, die verschiedenen Anwendungsfällen (bzw. Benutzergruppen) entsprechen [Ferr78, 5.2.2, 2.1] [Dirl06, 2.1]. Zu beachten ist, daß die Systemleistung nicht proportional mit ansteigender Last abfallen muß, sondern vielmehr bis nahe der Lastgrenze unterproportional und darüber hinaus rapide und unkontrolliert abfallen kann [Ferr78, 14.4].

2.1 Interpretation

Nach Ferrari gilt „very little can be said in general about the interpretation of results. It is mostly an art, which can be learned by practicing it and observing others practice it. Successful interpretation relies heavily on both intuition and deep knowledge of system structure and organization.“[Ferr78, 2.7.1] Schwierig ist insbesondere die Umsetzung der Benchmarkwerte in „weiche“ Ergebnisse, d.h. aus Sicht des Anwenders Eigenschaften wie „schnell“, „akzeptabel“, „schlecht“.

Die ISO-Norm beschreibt ein Bewertungssystem, das aus den Benchmarkwerten die Diskriminante „acceptable“ bzw. „unacceptable“ ableitet (siehe Abschnitt 3.1)

Grundsätzlich läßt sich bei der Interpretation zwischen absoluten und relativen Benchmarks unterscheiden. Absolute Werte sind dann von Interesse, wenn bereits Vorgaben existieren, die nicht unterschritten werden sollen. Google definiert beispielsweise eine maximale Antwortzeit von einer halben Sekunde für Suchanfragen. Absolute Werte können jedoch auch durch die Vorspiegelung exakter (und vor allem großer) Zahlen die wirklichen Relationen verschleiern. Relative

Benchmarks hingegen stellen die Unterschiede verschiedender Softwareprodukte dar. Dabei rechnen sich bestimmte Fehler heraus, wenn sie bei allen Messungen mit demselben Faktor eingehen. Das kann beispielsweise ein Faktor sein, der durch Profiler zu jeder Instruktion multipliziert wird. Auch relative Angaben können die Aussagekraft eines Benchmarks verschleiern, wenn bereits die schwächsten Ergebnisse gut genug sind.

Ein Benchmarkergebnis ist generell nur aussagekräftig, wenn die Benchmarkparameter bekannt sind. Da die Benchmarkdokumentation aber unspektakulärer als die reinen Ergebnisse ist, ist sie zur Veröffentlichung ungeeignet. SPEC und TPC schreiben bestimmte Standards vor, wie Benchmarkergebnisse zu veröffentlichen sind (siehe Abschnitt 4).

Die Unschärfen bei der Benchmarkinterpretation ergeben sich also aus zwei Richtungen. Einerseits aus der Reduktion einer komplexen Messung mit verschiedenen Annahmen auf einzelne (wenige) Zahlen. Insbesondere wenn der Vektor der Meßergebnisse auf ein Skalar reduziert wird und damit eine Ordnung induziert wird, die natürlicherweise gar nicht vorhanden ist. Andererseits ergeben sich Unschärfen aus der Extrapolation dieser Zahlen auf die jeweiligen Systeme des Anwenders. Für Softwarebenchmarks betrifft das vor allem das implizite Herausrechnen der Hardware des gemessenen Systems. Die Präsenz von Benchmarkergebnissen verdrängt möglicherweise auch wertbestimmende nicht-funktionale Eigenschaften – wie Robustheit oder Sicherheit – mit weniger ausgefeilten Techniken der quantitativen Gütebestimmung.

2.2 Statistische Stabilität

Theoretisch ist durch die Wahl der Parameter der Zustand des zu testenden Systems eindeutig bestimmt. In der Praxis bleiben jedoch einige Objekte des Zustandsraums unkontrolliert, z.B. die Belegung der Caches oder die Aktivität von Hintergrundprozessen. Es ist daher nötig, über mehrere Messungen mit denselben Parametern zu mitteln. Wir gehen von addier- und dividierbaren Maßen aus, so daß wir ein arithmetisches Mittel bilden können. Das ist für die Indizes aus Tabelle 1 der Fall. Für andere Maße – beispielsweise subjektive Maße wie „gut“, „akzeptabel“ und „schlecht“ – kann auf den Median zurückgegriffen werden. Der Median bietet noch andere Vorteile: er ist unempfindlich gegen statistische Ausreißer und er liefert (bei entsprechender Definition als Unter- oder Obermedian) immer einen tatsächlich gemessenen Wert.

A priori ist unklar, wie groß die Stichprobe für einen verläßlichen Mittelwert sein muß. Auch sagt ein Mittelwert noch nichts über die Streuung der Meßwerte aus. Ein Mittelwert erhält seine Aussagekraft daher erst durch ein zugehöriges Konfidenzintervall. Durch ein vorgegebenes Konfidenzintervall läßt sich die Frage beantworten, wieviele Messungen mindestens nötig sind. Wir gehen dabei davon aus, daß die Präzision unseres Stichprobenmittels, also unseres Schätzers für den Erwartungswert, mit zunehmender Stichprobengröße besser wird (das Mittel ist *konsistent*). Eine Einführung zu Konfidenzintervallen findet sich bei Henze und Kadelka [HeKa00, Kapitel 18]; Ferrari geht auf die Besonderheiten bei Benchmarks ein [Ferr78, 2.6.4].

Definition 3. *Ein (zweiseitiges)* Konfidenzintervall *für* $\varepsilon > 0$ *mit* Konfidenzniveau α, $0 < \alpha < 1$, *zu einem arithmetischen Mittel* m *mehrerer unabhängiger Messungen besagt, daß der tatsächliche Wert des gemessenen Parameters mit einer Wahrscheinlichkeit von mindestens* $1 - \alpha$ *im Intervall* $[m - \varepsilon, m + \varepsilon]$ *liegt.*

Für $\alpha \leq 0,05$ gilt das Intervall als signifikant, für $\alpha \leq 0,01$ als hochsignifikant. Zur Bestimmung des Konfidenzintervalls benötigen wir zusätzlich zum Mittelwert die Varianz σ^2 und die Verteilungsfunktion des gemessenen Parameters. Meist werden voneinander unabhängige und normalverteilte Messungen angenommen. Wir gehen im folgenden detailliert auf diesen Fall ein. Die Meßwerte seien durch $x_1, \ldots, x_n$ mit Mittelwert m gegeben. Da die tatsächliche Varianz unbekannt ist (ebenso wie der tatsächliche Mittelwert), nähern wir mittels der Stichprobenvarianz

$$s^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - m)^2. \tag{1}$$

Aufgrund dieser Schätzung muß für kleine Stichproben mit der Studentschen t-Verteilung weitergerechnet werden. Ab $n \approx 30$ kann näherungsweise die Normalverteilung verwendet werden. Es gilt für das Konfidenzintervall

$$t\text{-verteilt: } \left[m - t(1 - \alpha/2, n-1) \frac{s}{\sqrt{n}}, m + t(1 - \alpha/2, n-1) \frac{s}{\sqrt{n}} \right] \tag{2}$$

$$\text{normalverteilt: } \left[m - \phi(1 - \alpha/2) \frac{s}{\sqrt{n}}, m + \phi(1 - \alpha/2) \frac{s}{\sqrt{n}} \right] \tag{3}$$

In diese Formeln gehen die Quantilfunktion der Standardnormalverteilung $\phi(\alpha)$ bzw. der t-Verteilung mit s Freiheitsgraden $t(\alpha, s)$ ein. Ihre Berechnung ist nicht trivial (mehrere tausend Codezeilen); Statistikpakete – beispielsweise R – verfügen über entsprechende Funktionen. Quantile für häufig benötigte Signifikanzgrenzen α lassen sich auch Tabellen entnehmen. Ab einem Stichprobenumfang von $n \approx 50$ gilt bei unbekannten Verteilungen Formel 3 näherungsweise.

Der benötigte Stichprobenumfang wächst etwa quadratisch mit der Verkleinerung des Konfidenzintervalls. Für ein doppelt so genaues Intervall sind also viermal so viele Messungen nötig.

Beim Median haben wir es einfacher. Hier geben wir als Maß für die Streuung zusätzlich noch die 0,05- und 0,95- bzw. 0,01- und 0,99-Quantile an. Mit größerem Aufwand sind auch Aussagen für unbekannte Verteilungen und für abhängige Messungen möglich [Ferr78, 2.6.4].

Für komplexe Software, die lange Laufzeit und wenige Lade- und Entladezyklen hat, ist es sinnvoll, Initialaufwände wie Speicherallokation und Laden dynamischer Bibliotheken nicht mitzumessen. In diesem Fall werden beispielsweise die ersten ersten k Meßergebnisse ignoriert. Der Startzustand gehört zur Parameterdokumentation.

3 Standardisierung und Zertifizierung

Habt Euch vorher wohl präpariert,
Paragraphos wohl einstudiert,
Damit Ihr nachher besser seht,
Daß er nichts sagt, als was im Buche steht;
MEPHISTO, FAUST I

Eine Zertifizierung besagt, daß das untersuchte Zertifizierungsobjekt dem entspricht, „was im Buche steht“. Das heißt eine Zertifizierung ist zunächst nur eine Konformitätsbestätigung einen Standard betreffend. Sie trifft keine Aussage darüber, ob der Standard gut oder schlecht oder überhaupt relevant ist. Der Sinn eines Zertifikats ist aber gerade, zu bestätigen, daß etwas „dem Stand der Kunst“ entspricht. Ein Zertifikat ist nicht nur eine Konformitätsgarantie, sondern auch eine Herkunftsgarantie. Daran sind implizit bestimmte Güteeigenschaften geknüpft. Die Güte eines Zertifikats wird also als Marken-Eigenschaft kolportiert. So wird Zertifikaten nach DIN- oder ISO-Standards ein hohes Vertrauen entgegengebracht, während herstellereigene Standards skeptischer betrachtet werden.

3.1 Standardisierte Verfahrensweisen (Best Practices)

Für Benchmarks von EDV-Systemen gibt es standardisierte Verfahrensweisen nach DIN 66273 [DIN02] und nach ISO 14756 [ISO05]. Diese Standards beschreiben generische Methoden und Prozesse, um Benchmarks durchzuführen. Sie unterscheiden sich daher von den „Zertifikaten“ aus Abschnitt 4, in denen einzelne konkrete Benchmarks standardisiert werden.

DIN 66273 Der Standard DIN 66273 existiert seit 1990. Er ist ergänzt um genormte Arbeitslastvorgaben für Rechenzentren [DIN02, Teile 2-4]. Im Gegensatz zum ressourcen-orientierten Ansatz Ferraris nimmt die DIN eine rein physikalische Sichtweise ein und definiert

$$\text{Leistung} = \frac{\text{DV-Arbeit}}{\text{Zeit}}.$$

Der DIN-Standard konzentriert sich auf

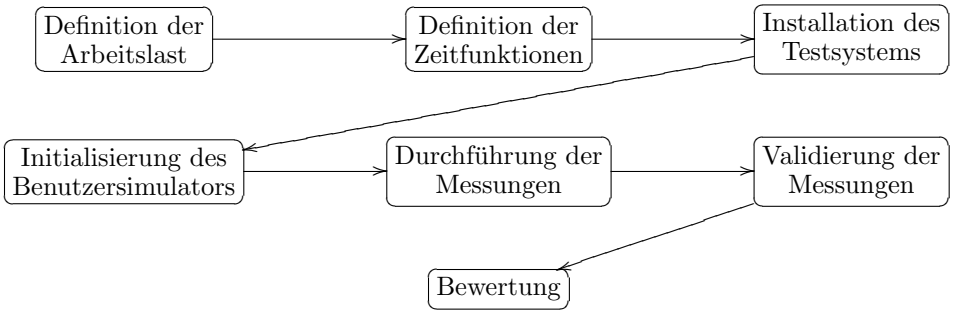
1. Eine Methode zur Erstellung feingranularer Arbeitslast-Modelle
2. Die sorgfältige Validierung der Meßergebnisse, inklusive statistischer Konfidenz
3. Die Bewertung der Messung.

Um die Arbeitslast zu bestimmen, folgt die DIN 66273 einem Mehrbenutzermodell. Das zu testende System kann bestimmte Aufgabentypen erledigen, z.B. einen Shell-Befehl ausführen, ein Programm laden, usw., die ihm von den Benutzern bzw. einem Simulator für mehrere Benutzer voneinander unabhängig während des Tests zugeteilt werden. Einzelne, konkrete Aufgaben werden zu Aufgabenketten komponiert, wobei zwei Aufgabenketten äquivalent sind, wenn sie die gleiche Anzahl Aufgaben vom gleichen Typ (in gleicher Reihenfolge) enthalten. Diesen Äquivalenzklassen wird je Benutzer eine Auftrittswahrscheinlichkeit zugeordnet. Das Testsystem führt also eine Reihe von Aufgabenketten aus, von denen jeweils gleichartige Ketten mit einer bestimmten Wahrscheinlichkeit vorkommen. Aufgaben können dabei auf einander warten müssen – was die Last eines interaktiven Systems beschreibt – oder gleichzeitig ans System gesendet werden – was einem Stapelverarbeitungssystem entspricht. Die Menge aller Aufgaben ist die *DV-Arbeit*.

Beim DIN-Standard gehört die Validierung der Korrektheit der Tests explizit dazu. Es werden also die Ausgaben des Systems mit Referenzausgaben verglichen. Das hat insbesondere den Sinn, festzustellen, ob eine Aufgabe tatsächlich vom System erledigt worden ist. Die statistische Konfidenz, also die Frage, ob die Stichprobe der Messung groß genug war, überprüft der Standard gemäß Formel (3).

Bestandteil jedes Aufgabentyps ist außerdem eine Zeitfunktion, die den Aufgabentyp in Zeitklassen einteilt. Sie gibt an, welcher Anteil der Aufgaben in welcher Zeit ausgeführt werden sollen, z.B. also 90% aller Suchanfragen in 0,5 Sekunden, 5% in 2 Sekunden und die restlichen 5% in nicht mehr als 10 Sekunden. Initialaufwände werden in einer Stabilisierungsphase durchgeführt und nicht mitgemessen. Die Zeitfunktion ist Bindeglied zur abschließenden Bewertung des Systems. Die Bewertung erfolgt entweder relativ zu den Meßwerten eines Vergleichssystems, oder als einfache Entscheidung, ob die Zeitvorgaben eingehalten wurden.

Insgesamt haben wir folgenden Benchmark-Prozeß:



ISO 14756 Der Standard ISO 14756 baut auf der DIN 66273 auf und erweitert sie um

- Einen Abschnitt zum Benchmark von Softwaresystemen [ISO05, 8.2]
- Sechs ausgearbeitete Arbeitslast-Modelle, die zusammen mit der Norm als konkrete Benchmarks verstanden werden können.

Zunächst stellt auch der ISO-Standard fest, daß sich die Leistung einer Software nur im Verbund mit der ausführenden Hardware ergibt. Der Standard schlägt daher eine relative Leistungsmessung für Software vor. Dabei wird das gängige Schichtenmodell aus Hardware, Mikroprogrammen, Betriebssystem, Übersetzer und Anwendungen zugrunde gelegt. Wenn jetzt die Komponente der Anwendungsschicht, d.h. die zu testende Software, unter Beibehaltung der Komponenten aller anderen Schichten ausgewechselt wird, ist eine relative Leistungsmessung möglich.[2] Konsequenterweise benötigt ein Softwarebenchmark daher immer ein Referenzsystem.

Weder der DIN- noch der ISO-Standard spezifiziert einen Benchmarkreport. Der Report, also die Dokumentation der Meßparameter und der Meßergebnisse, gehört jedoch selbstverständlich zum vollständigen Benchmark dazu. Dirlewanger schlägt eine Reportstruktur zur ISO-Norm vor [Dirl06, 9.3].

3.2 Zertifizierung

Eine ordentliche Zertifizierung besteht nicht nur aus der Konformitätsbestätigung selbst, sondern auch aus der Gewißheit, daß die zertifizierende Stelle überhaupt die Kompetenz besitzt, ein Zertifikat auszustellen. Es ist also gewissermaßen ein Zertifikat für die Zertifizierungsstelle nötig. Die Qualität aller beteiligten Institutionen wird durch eine Vertrauenskette sichergestellt. Grundlage ist ein möglichst weitläufig anerkannter Standard, also beispielsweise ein DIN-, ISO- oder IEEE-Standard. Außerdem muß eine „Master“-Institution öffentlichen Vertrauens ausgezeichnet sein: die *Akkreditierungsstelle*. Das kann beispielsweise eine Behörde sein. Die Akkreditierungsstelle spricht einer Zertifizierungsstelle die Kompetenz aus, Zertifikate bezüglich eines bestimmten Standards auszustellen.

[2] Streng genommen sind auch diese relativen Ergebnisse nur für genau die Komponenten der anderen Schichten gültig

Für (Software-)Benchmarks existiert zwar mit der ISO-Norm ein internationaler Standard, in der Zertifizierungspraxis haben sich aber alternative Ansätze etabliert. Es gibt eine Fülle von Institutionen, die Benchmarks veröffentlichen, darunter die Hersteller selbst, Fachzeitschriften, technische Institute, Anwender und Benchmark-Organisationen wie die SPEC. Für Softwarebenchmarks bezeichnen wir eine abgeschwächte Form des „Zertifikats“ als Versicherung der veröffentlichenden Institution, die Messungen nach einer bestimmten, nachvollziehbaren Verfahrensweise erstellt zu haben. Ist ein Benchmark nach den Regeln einer Benchmark-Organisation entstanden, gehört üblicherweise die Bestätigung der entsprechenden Organisation dazu.

Zwei Merkmale zeichnen die Güte eines solchen Zertifikats aus: das Renommee dessen, der veröffentlicht, und die Güte der Benchmarkmethode, insbesondere die vollständige Dokumentation. Eine aufmerksame Öffentlichkeit kann anhand der Dokumentation nachvollziehen, ob die aus den Benchmarks gezogenen Schlüsse sinnvoll sind. Falls Benchmark-Organisationen Herstellerkonsortien sind, kommen als weitere Kontrolle die gegenläufigen Interessen der konkurrierenden Hersteller dazu.

4 Standardsoftware-Benchmarks

Im folgenden betrachten wir die Standardsoftware-Benchmarks SPEC CPU, SPECweb und die TPC-Suiten. Diese Benchmarks stellen abgeschwächte Zertifikate nach Abschnitt 3.2 da. Im Gegensatz zum DIN- und ISO-Standard spezifizieren sie eine konkrete Arbeitslast, sie geben Meßwerkzeuge vor und sie legen Meßmethode und Dokumentations- und Veröffentlichungsregeln fest. Die Benchmarks werden üblicherweise vom Hersteller selbst erstellt und der Benchmarkorganisation zur Überprüfung und Veröffentlichung übergeben. Die Regeln der BAPCo-Benchmarks, die Leistung typischer Client-Software messen, sind nicht frei zugänglich. SPEC, TPC und BAPCo sind als Herstellerkonsortien organisiert.

Während die Dokumentationsregeln wie üblich der Reproduzierbarkeit dienen, sind die Veröffentlichungsregeln besondere Merkmale dieser Benchmarks. Die Organisationen sind sich bewußt (und es ist auch explizit so vorgesehen), daß die Benchmarkergebnisse zu Werbezwecken verwendet werden [SPEC06a, 1.1] [TPC07a, 8.2]. Es ist daher sorgfältig darauf geachtet, keine irreführenden Darstellungen zu ermöglichen. Im Gegensatz zu generischen Standards sind durch die klare Begrenzung auf einen bestimmten Softwaretyp detaillierte Vorgaben der Konfigurationsparameter möglich (beispielsweise Compilerflags, einzuhaltende RFCs, usw.). Trotz dieser Maßnahmen sind die Benchmarks selbstverständlich auf gutwillige Tester angewiesen [SPEC06a, 1.2.3].

Bei allen betrachteten Benchmarks sind die Arbeitslasten und die Meßwerkzeuge nicht frei zugänglich. Die Verwendung freier Software ist nur eingeschränkt vorgesehen [SPEC06b, 3.7.2].

4.1 SPEC CPU 2006 und SPECweb 2005

Die Standard Performance Evaluation Corporation (SPEC) wurde 1988 gegründet, mit dem Ziel dem vorherrschenden „marketing hype“ vergleichbare und sinnvolle Benchmarks entgegenzusetzen. Die SPEC-Benchmarks messen Hardware-Software-Verbundsysteme aus unterschiedlichen Kategorien, unter anderem Übersetzer und Prozessoren durch die SPEC CPU und Webserver durch die SPECweb. Leistungsindizes sind relativer Durchsatz (SPECweb), also gleichzeitig bedienbare Benutzer, bzw. absolute und relative Geschwindigkeit (SPEC CPU). Die SPEC gibt jeweils Referenzsysteme vor. Es wird ein schwaches Verfahren zur Sicherstellung statistischer Stabilität verwendet. Aus drei Durchläufen der Benchmark-Suiten wird der Median weiterverwendet. Die relative Geschwindigkeit (SPECrate) bestimmt sich aus dem geometrischen Mittel der relativen Geschwindigkeiten der Einzelbenchmarks. Folglich müssen die Suiten vollständig und unter Beibehaltung aller Konfigurationsparameter durchgeführt werden, wobei die SPEC CPU eine separate Messung der Ganzzahl- und der Gleitkommabenchmarks erlaubt. Die Arbeitslasten bei der SPEC CPU bestehen aus einer Reihe von rechenintensiven Realprogrammen (gcc, povray, ...), bei der SPECweb bestehen sie aus typischen statischen und dynamischen Webseiten aus den Bereichen Online Banking, E-Commerce und technischer Support. Die Arbeitslasten der SPECweb konnten von Dirlewanger mit kleinen Änderungen in ISO-konforme Arbeitslasten überführt werden [Dirl06, 14.8.3.4].

Zum Führen der SPEC-Benchmark-Zertifikate ist berechtigt, wer Benchmarks gemäß den „Run and Reporting Rules“ durchführt und veröffentlicht [SPEC06a,SPEC06b]. Eine Veröffentlichung auf der SPEC-Webseite ist nicht nötig, wird aber erbeten. Die SPEC erhält (auf Anfrage innerhalb von 10 Arbeitstagen) eine vollständige Kopie der Benchmarkunterlagen inkl. aller nötigen Benchmarkparameter, um den Test reproduzieren zu können. Für die SPEC CPU sind zwei Modi vorgesehen: „Peak Builds“ mit aggressiver Optimierung und „Base Builds“ mit sicherer Optimierung. Werden Peak Builds veröffentlicht müssen auch die zugehörigen Base Builds zugänglich gemacht werden. Die verwendete Hard- und Software muß Produktionsreife haben und von Kunden am Markt gekauft werden können. Veröffentlichungen, die nicht in allen Punkten den SPEC-Regeln entsprechen, sind erlaubt, wenn sie klar als „Estimates“ gekennzeichnet sind. Die SPEC behält sich im Fall von Regelverstößen „Maßnahmen“ („to take actions“) vor.

Von veröffentlichten Benchmarks wird erwartet, daß sie

- sinnvoll,
- vergleichbar und
- reproduzierbar

sind. Über allem gilt der Grundsatz des „fair use“. Das heißt insbesondere, daß Software nicht speziell für SPEC-Benchmarks optimiert sein darf. Die SPEC ist sich bewußt, daß der Grat zwischen allgemeiner Optimierung und benchmarkspezifischer Optimierung schmal ist. Als Richtlinie gilt, daß Software und Konfigurationsparameter auch für andere und größere Arbeitslasten als die der Benchmarks verwendet werden. Ein explizite Überprüfung der Ergebnisse ist nicht

spezifizert. Die SPEC vertraut auf ehrliche Tester und ein selbstverwaltendes Peer-Review durch ihre Mitglieder.

4.2 TPC

Das Transaction Processing Performance Council (TPC) wurde 1988 gegründet. Die TPC zertifiziert Benchmarks für Transaktionssysteme. Gemessen wird ein Verbund aus Hardware und Datenbanksoftware. Die aktuellen Benchmarks umfassen TPC-C für Datenbankmanagement-Systeme, TPC-E für Datenbanken im Börsenumfeld, TPC-H für Entscheidungsunterstützungs-Systeme (Business-Intelligence-Systeme) und die TPC-App für Applikations- und Webserver im B2B-Betrieb. Die einzelnen Benchmarks sind also szenarienbasiert, was den Aufgabentypen des DIN-Standard vergleichbar ist. Der TPC-C-Benchmark definiert außerdem Dienstgüten, die in etwa einer DIN-Zeitfunktion entsprechen: 95% der Antwortzeiten dürfen 2 Sekunden nicht überschreiten. Die Leistungsindizes der Benchmarksuiten sind durchsatzorientiert, also Anfragen bzw. Transaktionen pro Zeit. Bestandteil des Index ist der Preis des Systems, der gemäß der TPC Pricing Specification [TPC07b] angegeben werden muß. Das schließt ein, daß das System überhaupt am Markt erhältlich ist. Es gilt eine Art All-Inclusive-Grundsatz, der Preis muß dem entsprechen, was ein beliebiger Kunde am Markt für einen einmaligen Einzelkauf inkl. Garantie und dreijährigem 24x7-Support bezahlen würde. Die genauen Spezifikationen für Preisangaben sind benchmarkspezifisch. Als Rahmen für Meßungenauigkeiten akzeptiert die TPC bis zu 2% Abweichung für die Größe Preis/Leistung.

Die Regeln für gültige TPC-Benchmarks sind in den TPC Policies spezifiziert [TPC07a]. Ein Benchmark ist TPC-zertifiziert, falls und solange er auf der TCP-Result-List im Web verfügbar ist. Vor der Veröffentlichung findet eine explizite Review-Phase statt, in der die TPC-Mitglieder zu Eingaben aufgefordert sind. Die Benchmarkunterlagen müssen der TPC am Tag der Veröffentlichung durch den Hersteller eingereicht werden. Die zertifizierten Ergebnisse dürfen und sollen einer breiten Öffentlichkeit zugänglich gemacht werden, unter Verwendung des TPC-Logos. Die Veröffentlichungsregeln folgen der „TPC Fair Use Policy“, d.h. sie müssen

- präzise,
- aufrichtig,
- korrekt und
- lesbar und klar dargestellt

sein [TPC07a, 8.2.1, 8.2.2]. Die TPC ist dabei wesentlich penibler als die SPEC, bis hin zur Angabe von Minimalschriftgrößen. Neben einer vollständigen Benchmark-Dokumentation zur TPC-internen Überprüfung gehört zur Veröffentlichung die „Executive Summary“. Sie enthält neben dem Benchmarkergebnis die Einheit des Leistungsindex (die „Metrik“), die Anzahl der Prozessorkerne (bzw. Hyperthreading-Einheiten) und die Preisinformationen. Die TPC ermuntert die Medien, bei Verweisen auf TPC-zertifizierte Benchmarks dieselben Veröffentlichungs-Richtlinien anzuwenden.

5 Schlußbemerkungen

Gute Benchmarkergebnisse haben einen hohen Wert. Sie liefern griffige Argumente für Kaufentscheidungen, insbesondere im performance-fokussierten Umfeld der Standard-Software, und sie schaffen Vergleichbarkeit, selbst wenn sie gar nicht vorhanden ist. Die größte Unsicherheit sind die angenommenen Arbeitslasten – die Messung selbst ist mit einiger Sorgfalt hinreichend genau durchführbar. Wenn immer möglich, sollte ein Anwender daher mit seinen eigenen Daten messen. Es empfiehlt sich außerdem, zwischen dem eigentlichen Benchmark und den daraus gezogenen Schlüssen zu unterscheiden. Interessant ist insbesondere, von wem die Schlüsse gezogen werden. Im Fall der Mindcraft-Studie [Mind99] kam erwartungsgemäß Microsoft zu einer ganz anderen Bewertung, als die Open-Source-Szene. Auch umgekehrt gibt es Sorge vor falschen Eindrücken: der Hersteller VMWare untersagt in seinen Lizenzbedingungen die ungefragte Veröffentlichung von Benchmarks.

Um mißbräuchliche Darstellungen zu verhindern, ist einige Anstrengung nötig. Die SPEC CPU Run and Reporting Rules beispielsweise bestehen zu einem Gutteil daraus, die kleinen Tricks zu verbieten. Ein Vorteil von Zertifizierungen ist, daß sie ebenso griffig wie die Benchmarkergebnisse selbst sind: die Zahlen werden mit einem Logo versehen. Gute Zertifikate können die relevanten von den irrelevanten Benchmarks trennen. Andererseits sind Benchmarkzertifizierungen keine binären Checklisten-Entscheidungen, eine Zertifizierungsstelle müßte konkrete, quantitative Angaben bestätigen; und daran lassen sich leicht die Finger verbrennen. Anscheinend bewährt haben sich Zertifikate von Benchmarkorganisationen.

Literatur

Blom07. J. Blomer. Benchmarking HEP applications in virtual machines. Technischer Bericht, AliEnX Group, CERN, 2007.

DIN02. DIN. *DIN 66273 – Messung und Bewertung der Leistung von DV-Systemen.* Deutsches Institut für Normung. 2002.

Dirl06. Werner Dirlewanger. *Measurement and Rating of Computer Systems Performance and of Software Efficiency.* Kassel University Press. 2006.

EbDu07. Christof Ebert und Reiner Dumke. *Software measurement.* Springer. 2007.

Ferr78. Domenico Ferrari. *Computer Systems Performance Evaluation.* Prentice-Hall. 1978.

HeKa00. N. Henze und D. Kadelka. Wahrscheinlichkeitstheorie und Statistik. Vorlesungsskript, 2000.

Idea00. Ideas International. Übersicht zu ausgewählten Benchmarkergebnissen. `http://www.ideasinternational.com/benchmark/bench.html`, 2000.

ISO05. ISO. *ISO/IEC 14756:1999 – Measurement and rating of performance of computer-based software systems.* International Organization for Standardization. 2005.

Mind99. Mindcraft. Open Benchmark: Windows NT 4.0 and Linux. Technischer Bericht, Mindcraft, 1999.

SPEC06a. SPEC. SPEC CPU2006 Run and Reporting Rules. Technischer Bericht, Standard Performance Evaluation Cooperation, 2006.

SPEC06b. SPEC. SPECweb2005 Release 1.10 Run and Reporting Rules. Technischer Bericht, Standard Performance Evaluation Cooperation, 2006.

TPC07a. TPC. TPC Policies, Version 5.14. http://www.tpc.org/information/about/documentation/TPC_Policies_v5.14.htm, Transaction Processing Performance Council, 2007.

TPC07b. TPC. TPC Pricing Specification, Version 1.2.0. http://www.tpc.org/pricing/spec/Price_V1.2.0.pdf, Transaction Processing Performance Council, 2007.

SWT: Das Standard Widget Toolkit

Grischa Liebel

Betreuer: Dipl.-Inform. Klaus Krogmann

Zusammenfassung Ohne eine grafische Oberfläche kommt heute selten noch ein Programm aus. In den meisten höheren Programmiersprachen stehen gleich mehrere Programmierschnittstellen zur Verfügung, mit denen solche Oberflächen erstellt werden können. So gibt es auch unter Java drei verschiedene dieser Schnittstellen: AWT, Swing und SWT.
Diese Ausarbeitung soll einen kleinen Einblick in die Programmierschnittstelle SWT, das Standard Widget Toolkit, geben. Dabei wird sowohl auf die Programmierung mit SWT eingegangen, als auch ein Vergleich zu den anderen beiden Programmierschnittstellen, AWT und Swing, vorgenommen.

1 Einleitung

1.1 Grafische Benutzeroberflächen

„Grafische Benutzeroberflächen“, kurz GUIs, erlangten bereits in den 80er Jahren, spätestens aber mit der Veröffentlichung von Microsoft Windows 3.10 im Jahre 1990, einen Durchbruch. Heutzutage sind sie aus dem Bereich der Desktopcomputer nicht mehr wegzudenken [1]. Dementsprechend ist es auch für Programmierer von großer Bedeutung, über die Möglichkeiten der Programmierung solcher Oberflächen Bescheid zu wissen. Unter Java gibt es verschiedene Programmierschnittstellen, wegen der englischen Übersetzung APIs genannt, um grafische Benutzeroberflächen zu erstellen. Eine davon ist das SWT. Im Rahmen dieser Ausarbeitung wird zunächst eine kleine Einführung in den Themenbereich gegeben. Anschließend wird die Programmierung mit SWT, sowie die Erweiterung mit Hilfe von „JFace“, behandelt. Zum Schluss wird ein Vergleich zu den zwei anderen bekannten APIs zur Erstellung von GUIs in Java, AWT und Swing, vorgenommen.

1.2 Das SWT

Das SWT wurde 2001 von Eclipse.org veröffentlicht. Die bereits existierenden, von Sun veröffentlichten, APIs zur Erstellung von grafischen Oberflächen, AWT und Swing, hatten sich aufgrund von Defiziten, hauptsächlich im Aussehen und der Geschwindigkeit, nie für große, aufwendige Oberflächen geeignet. SWT sollte das Beste beider GUI-APIs vereinen und somit endlich die großen Unterschiede beseitigen, die zwischen einem grafischen Java-Programm und einem, extra

für die jeweilige Plattform entwickelten, Programm bis zu diesem Zeitpunkt bestanden hatten [2]. Obwohl SWT von den Entwicklern vordergründig für Eclipse gedacht war, ist es auch getrennt davon nutzbar.
Bereits zwei Jahre nach der Veröffentlichung wurde SWT durch den „Readers Choice Award“ des „Java Developer's Journal“ als „Best Java Component“ ausgezeichnet [3]. Heute nutzen viele größere Programme, darunter auch Eclipse selbst, das SWT.

1.3 Möglichkeiten von SWT

SWT bietet dem Programmierer eine Vielfalt an Möglichkeiten. Einige dieser Möglichkeiten werden hier genannt, für einen tiefergehenden Einblick sei auf [4] und [5] verwiesen, auf denen auch die hier dargestellten Informationen basieren.
Der meistgenannte Vorteil ist die Geschwindigkeit von SWT. Anstatt alle Komponenten, wie Buttons, Textfelder oder Fenster selbst zu zeichnen, benutzt es die nativen Methoden des Betriebssystems. Um die nativen Methoden aufzurufen, gibt es in SWT die sogenannten „Wrapper“-Klassen. Diese sorgen dafür, das die Java-Befehle zum Zeichnen von Komponenten innerhalb der SWT-Bibliotheken zu einem Aufruf der nativen Methoden des Betriebssystem führen. Zeichnet man beispielsweise einen Button in einem SWT-Programm, so wird dadurch indirekt nur die betriebssystemeigene Methode zum Zeichnen dieses Buttons aufgerufen und nicht speicherintensiv durch die „Java Virtual Machine“ emuliert. Da sich die Methoden von Betriebssystem zu Betriebssystem allerdings unterscheiden, muss auf jedem Betriebssystem auch eine andere Methode aufgerufen werden. Der Programmteil von SWT, der für den Aufruf der nativen Betriebssystemmethoden verantwortlich ist, musste somit von den Eclipse-Entwicklern mehrfach entwickelt werden. Weiterhin ist nicht auf jedem Betriebssystem jede Funktion eines anderen Betriebssystems vorhanden. Um dennoch überall die gleichen Möglichkeiten zu bieten, ohne auf Funktionen verzichten zu müssen, werden Methoden, für die es keine Entsprechung im Betriebssystem gibt, emuliert. Zur Schnelligkeit des SWT kommt somit eine große Funktionsvielfalt.
Das zweite wichtige Argument, das für SWT spricht, ist das sogenannte „Look and Feel“. Da SWT nur die Betriebssystemmethoden aufruft, werden die Komponenten auch so gezeichnet, wie es im Betriebssystem vom Benutzer eingestellt wurde. Das Programm sieht somit nicht mehr „fremd“ für den Benutzer aus, sondern so, als wäre es extra für sein Betriebssystem und seine Einstellungen kompiliert worden.
SWT arbeitet nach dem Prinzip des „Top-Down Programming“. Wird eine Komponente, also beispielsweise ein Button, erstellt, so muss direkt angegeben werden, in welcher übergeordneten Komponente, beispielsweise welchem Fenster, er platziert werden soll. So kann der Button nicht erst erstellt werden und später in ein Fenster, oder gar mehrere, eingefügt werden, sondern dies muss gleich beim Erstellen geschehen.
Eine Eigenheit von SWT ist das „manual disposal“. Im Gegensatz zu anderen Java Bibliotheken, wird der, durch Objekte reservierte, Speicher nicht automatisch wieder freigegeben, sondern muss vom Programmierer ausdrücklich freigegeben

werden. Zunächst scheint dies umständlich, ist allerdings im Bezug auf SWT sinnvoll. Da SWT direkt die Betriebssystemmethoden aufruft ist es wichtig, dass die Komponenten zum richtigen Zeitpunkt gelöscht werden, sowohl rechtzeitig als auch nicht zu früh. Eine späte Freigabe von Speicher würde zu einer unnötigen Verschwendung von Systemressourcen führen, die zu frühe Freigabe könnte sogar zu schwerwiegenden Fehlern führen. Die Aufgabe des „Aufräumens“ wird dadurch erleichtert, dass beim Entfernen einer Komponente alle untergeordneten Komponenten, also Komponenten die dieser Komponente zugewiesen wurden, mit entfernt werden.

2 Die SWT-API

Im Folgenden wird auf die Programmierung von Java-Programmen, die SWT nutzen, eingegangen. Dabei wird ein Beispielprojekt begonnen, in das nach und nach die erklärten Komponenten eingebunden werden und das schließlich ein funktionsfähiges Programm bildet, das eine Passwortabfrage realisieren soll.

2.1 Einbindung von SWT in ein Java-Projekt

Dadurch. dass SWT von Eclipse.org und nicht von Sun entwickelt wird, ist es nicht Teil der Java Runtime Environment (JRE). Daher müssen die SWT-Bibliotheken, zusätzlich zum eigentlichen Programm, in das Projekt eingebunden werden. Dies kann auf verschiedene Weisen geschehen. Wird Eclipse verwendet, so gestaltet sich die Einbindung einfach. Am einfachsten ist die Einbindung unter Eclipse mit Hilfe der heruntergeladenen Zip-Datei von der Eclipse-Webseite [6]. Nachdem die betriebssystemspezifische Zip-Datei heruntergeladen wurde, wird sie in Eclipse mit „File“, „Import“ als „Existing Project into Workspace” eingebunden. Soll ein neues Java-Projekt die SWT-API nutzen, so kann diesem Projekt im „Java Build Path“ die importierte SWT-API als Referenz übergeben werden. Dazu wird in den Projekteigenschaften unter „Java Build Path“, im Reiter „Projects“ das importierte SWT-Projekt ausgewählt und hinzugefügt. Von nun an steht die SWT-API im neuen Projekt zur Verfügung.

2.2 Grundgerüst eines SWT-Programms

Um mit SWT ein Fenster darzustellen, wird immer der gleiche Grundaufbau benötigt [7] [8]. Zuerst müssen die SWT-Pakete in das Programm importiert werden. Dabei reicht für den Anfang das allgemeine SWT-Paket und das „Widgets“-Paket.

```
import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
```

Zunächst wird ein neues Objekt vom Typ `Display` erzeugt. Es wird als oberste Komponente benötigt, um weitere Komponenten aufzunehmen und Benutzeraktionen wie Mausklicks zu verwalten. Das Display-Objekt ist für den Benutzer

unsichtbar. Um schließlich das grafische Programmfenster auf den Bildschirm zu zeichnen, wird ein Objekt vom Typ `Shell` erzeugt, dem das Display als übergeordnete Komponente zugewiesen wird. Anschließend wird mit `setSize()` die Fenstergröße festgelegt, wobei im Beispiel eine Größe von 400 auf 150 Pixel verwendet wird.

```
Display display = new Display();
Shell shell = new Shell(display);
shell.setSize(400,150);
```

Angezeigt wird das Shell-Objekt erst, nachdem die `open()`-Methode aufgerufen wird. Zum Schluss wird noch die Event-Schleife benötigt, die den Programmablauf regelt und Aktionen, wie einen Mausklick, aufnimmt. Dabei bleibt das Programm so lange in der Schleife hängen, wird also nicht beendet, so lange das Programmfenster nicht geschlossen wurde. Dies bedeutet natürlich in diesem Fall auch, dass das erzeugte `Shell`-Objekt das Hauptfenster ist, bei dessen Schließung das ganze Programm beendet wird. Innerhalb der Schleife wird festgelegt, dass das `Display`-Objekt in einen Wartezustand versetzt wird, bis eine zu verarbeitende Aktion, wie dem oben genannten Mausklick, erfolgt. Sämtlicher Programmcode zur Erzeugung von Grafikkomponenten oder Steuerung der grafischen Oberfläche muss vor dieser Schleife und stehen, da nachstehender Programmcode erst beim Schließen des Fensters ausgeführt wird. Nach der Schleife wird schließlich nur noch das `Display`-Objekt mit der `dispose()`-Methode wieder gelöscht. Alle anderen Komponenten werden, da sie in display oder darunterliegende Objekte eingebunden sind, mitzerstört.

```
shell.open();
while (!shell.isDisposed()) {
    if(!display.readAndDispatch()) {
        display.sleep();
    }
}
display.dispose();
```

Das Grundprogramm, in das in den weiteren Kapiteln zusätzlicher Programmcode eingefügt wird, ist damit fertig. Zunächst werden die Pakete importiert, dann ein Objekt vom Typ `Display` und eines vom Typ `Shell` erzeugt. Dann wird die Größe des `Shell`-Objektes mit `setSize()` festgelegt. Mit `open()` wird das Fenster auf dem Bildschirm gezeichnet und schließlich in der Event-Schleife auf das Schließen des Hauptfensters oder auf eine, zu verarbeitende, Aktion gewartet.

```
import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;

public class SWTProgram {
    public static void main (String args[]) {
```

```
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setSize(400,150);

        shell.open();

        while (!shell.isDisposed()) {
            if(!display.readAndDispatch()) {
                display.sleep();
            }
        }
        display.dispose();

    }
}
```

Grafisch sieht das Fenster folgendermaßen aus.

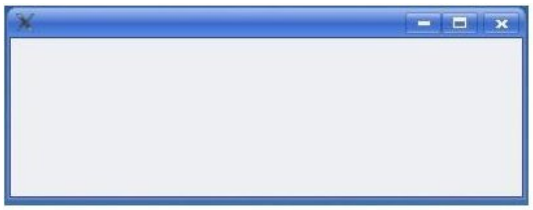


Abbildung 1. Grundgerüst unter Linux mit KDE 3.5

2.3 Widgets

Grundlagen zu Widgets Der englische Begriff „Widget“ bedeutet, ins Deutsche übersetzt, eigentlich „Dingsbums“. Im Bezug auf grafische Oberflächen bezeichnet es jedoch ein Steuerelement, also alles was auf dem Bildschirm angezeigt werden kann [4]. In SWT sind die Widgets alle im, oben bereits verwendeten, Paket „org.eclipse.swt.widgets“ enthalten. Im folgenden wird auf die Widgets `Label`, `Button`, `Text` und `Composite` eingegangen, wobei die Informationen zu großen Teilen auf [7] und [8] basieren.

Widgets am Beispiel Zunächst wird das Widget `Label` betrachtet. Das Label ist eine nicht auswählbare Komponente, die eine Zeichenkette oder ein Bild anzeigt. Eingesetzt wird es häufig zur Beschriftung von Textfeldern. Beim Erzeugen

eines Labels muss diesem im Konstruktor die übergeordnete Komponente, also im Beispielfall das Objekt „shell“, und ein Stil zugewiesen werden. Beim Widget `Label` kann der Stil zur horizontalen Anordnung des Labels dienen. Als Stilvariable muss grundsätzlich ein Integer übergeben werden. SWT liefert für diese Stile viele Konstanten, sodass hier nur eine passende ausgewählt werden muss. Da zunächst keine besondere Anordnung benötigt wird, wird der Stil `SWT.NONE` gewählt.

```
Label userL = new Label (shell, SWT.NONE);
```

Mit `setText()` wird anschließend der Inhalt des Labels festgelegt. Da im Beispielprojekt noch kein Layout vorhanden ist, welches die Widgets nach einem bestimmten Muster anordnet, muss die Position und die Größe des Labels genau festgelegt werden. Dies geschieht mit der Methode `setBounds()`, welcher die X- und die Y-Koordinate des Labels innerhalb der übergeordneten Komponente, sowie die Höhe und Länge des Labels als Parameter übergeben werden. Folgender Code weist dem Label den Text „Benutzername: “, die Position 10, 10 und die Länge 100, Höhe 20 zu.

```
userL.setText("Benutzername: ");
userL.setBounds(10, 10, 100, 20);
```

Das Label wird in das Shell-Objekt gezeichnet, sobald die `setBounds`-Methode aufgerufen wurde.
Da noch ein Passworteingabefeld benötigt wird, wird noch ein Label, allerdings mit dem Text „Passwort: “ und Position 10, 40 erzeugt.

```
Label passL = new Label (shell, SWT.NONE);
passL.setText("Passwort: ");
passL.setBounds(10, 40, 100, 20);
```

Der angegebene Code wird im Beispielprogramm vor der `open`-Methode des `Shell`-Objekts platziert und erzeugt zwei Zeichenketten „Benutzername: “ und „Passwort: “.
Als nächstes werden zwei Eingabefelder benötigt. Eines zur Eingabe des Benutzernamens und eines, mit Sternchen als Eingabezeichen, in welches das Passwort eingegeben wird. Ein Textfeld wird durch das Widget `Text` dargestellt. Im wesentlichen gleicht die Erzeugung dieses Widgets der Erzeugung des Labels, nur wird als Stil diesmal `SWT.BORDER` gewählt, um einen Rahmen um das Eingabefeld zu zeichnen.

```
Text userT = new Text (shell, SWT.BORDER);
```

Mit `setTextLimit()` kann die maximale Anzahl der Zeichen, die eingegeben werden können, festgelegt werden. Im Beispiel wird 20 als maximale Eingabelänge festgelegt. Mit `setBounds()` wird auch hier die Position und die Größe des Widgets festgelegt.

```
userT.setTextLimit(20);
userT.setBounds(150, 10, 150, 20);
```

Nun wird weiterhin noch das Passworteingebefeld benötigt, welches die besondere Eigenschaft hat, dass der eingegebene Text nur in Form von '*'-Symbolen angezeigt wird. Dazu dient die Methode `setEchoChar()`, der das Rückgabezeichen in Form eines Characters übergeben wird.

```
Text passT = new Text (shell, SWT.BORDER);
passT.setTextLimit(20);
passT.setEchoChar('*');
passT.setBounds(150, 40, 150, 20);
```

Insgesamt werden also zwei Textfelder mit maximaler Eingabelänge 20 erzeugt. Eines davon, das Passworteingabefeld, zeigt den eingegebenen Text in Form von Stern-Symbolen an. Zur Überprüfung der eingegebenen Zeichenketten wird im Laufe der späteren Kapitel noch eine Methode benötigt, die die eingegebenen Zeichenketten der Textfelder zurückgibt. Dazu dient die Methode `getText()` der Text-Widgets.
Nachdem der Benutzer nun die Möglichkeit hat seinen Benutzernamen und das zugehörige Passwort einzutippen, wird noch ein Widget benötigt, mit dem er seine Eingaben bestätigen kann. Dazu dient das Widget `Button`. Die Erzeugung ähnelt wieder derer von Label und Text, diesmal mit dem Stil `SWT.PUSH`. Dieser sorgt dafür, dass der Button als gewöhnliche Schaltfläche mit einer Zeichenkette angezeigt wird. Andere Möglichkeiten wären `SWT.ARROW` für einen Pfeil-Button oder `SWT.RADIO` für einen Radio-Button. Die Beschriftung des Buttons wird, wie bei Label, mit `setText()` vorgenommen. Im Beispielfall wird „Absenden“ verwendet. Schließlich werden, wie gewohnt, die Ausmaße des Widgets mit `setBounds()` gesetzt.

```
Button button = new Button (shell, SWT.PUSH);
button.setText("Absenden");
button.setBounds(10, 80, 90, 30);
```

Das gesamte Programm sieht nun folgendermaßen aus.

Abbildung 2. Beispielprogramm mit Widgets unter Linux mit KDE 3.5

Zuletzt wird nun noch auf das `Composite`-Widget eingegangen, das zwar nicht

für das Beispielprogramm benötigt wird, aber für das Verständnis von verschachtelten Layouts notwendig ist. Das Composite dient als „Container"-Objekt. Das bedeutet, dass es andere Widgets in sich aufnehmen kann. Wird ein Objekt vom Typ `Composite` erzeugt, so können diesem Objekt Widgets zugewiesen werden. Existiert beispielsweise ein Composite „comp1" und es wird ein Button erzeugt, so könnte diesem Button als übergeordnetes Objekt, „comp1", anstatt des üblichen `Shell`-Objekts, zugewiesen werden.

```
Composite comp1 = new Composite(shell, SWT.NONE);
Button button1 = new Button(comp1, SWT.PUSH);
```

Zu beachten ist hierbei, dass das `Composite`-Objekt wiederum einem übergeordneten Objekt, beispielsweise dem `Shell-Objekt`, zugeordnet werden muss.

2.4 Events

Grundlagen zu Events Mit den bisher vorgestellten Teilen von SWT lassen sich zwar Komponenten einer grafischen Oberfläche zeichnen, allerdings kann der Benutzer keine Aktionen ausführen, da die Komponenten nicht auf eventuelle Ereignisse reagieren. Die Ereignisse auf einer grafischen Oberfläche, wie das Anklicken eines Buttons oder das Drücken einer bestimmten Taste, während das Fenster aktiv ist, werden „Events" genannt. In SWT werden die Events von der `readAndDispatch`-Methode des Displays abgefangen und an die Widgets weitergeleitet, die von dem Event betroffen wurden, beispielsweise dem angeklickten Button. Damit das Widget das Event verarbeiten kann, muss ihm ein sogenannter „Listener" zugewiesen werden, also ein Objekt, das auf Events horcht, die beim Widget ankommen. Dieser Listener entscheidet dann, welche Aktionen je nach Event ausgeführt werden sollen.
Es gibt in SWT die „typed Listener" und die „untyped Listener". Die „typed Listener" werden speziell für eine Art von Event erstellt, beispielsweise der `selectionListener`, der auf alle Events horcht, die die Auswahl eines Widgets betreffen. Untyped Listener dagegen sind allgemeiner. Sie werden einem Widget zugewiesen und horchen daraufhin auf alle Events, die das Widget betreffen. Dadurch ist die Flexibilität größer, es werden aber auch leichter Programmierfehler gemacht. Im Rahmen dieser Ausarbeitung wird nur auf die typed Listener eingegangen, da sie einfacher sind und für die Verdeutlichung der Event-Verarbeitung ausreichen.
Jeder typed Listener bringt eine Vielzahl an Events mit, auf die gehorcht werden kann. Der `mouseListener` verfügt beispielsweise über die Methoden `mouseUp()`, `mouseDown()` und `mouseDoubleClick()`, die die entsprechenden Events abfangen. Einerseits hat dies den Vorteil, dass für mehrere Events jeweils nur ein Listener benötigt wird, allerdings haben sie auch den Nachteil, dass sie abstrakte Klassen sind. Daher muss bei der Erzeugung eines Objektes, das von einer dieser Klasse erbt, jede Methode der abstrakten Klasse überschrieben werden. Wird beim `mouseListener` nur die Methode `mouseDown()` benötigt, um einen einfach Klick der Maus abzufangen, so müssen dennoch die anderen beiden Methoden überschrieben werden. Um diese Programmierarbeit zu vermeiden, bietet

SWT die „Adapter“-Klassen. Sie sind Unterklassen der Listener, sind allerdings keine abstrakten Klassen mehr. Somit müssen nur die wirklich benötigten Methoden überschrieben werden. Zu jedem Listener gibt es eine gleichnamige Adapterklasse, im Beispiel wären dies `mouseListener` und `mouseAdapter`.
Die Informationen zu Events sind im wesentlichen an [4] und [8] angelehnt.

Events am Beispiel Das vorhandene Programm soll nun um zwei Funktionen erweitert werden, um eventuelle Mausklicks auf den Button und ein Drücken der Ende-Taste, wärend der Button per Tabulator ausgewählt ist, abzufangen und darauf zu reagieren. Tritt eines der genannten Events auf, so soll der Inhalt der beiden Textfelder, Benutzername und Passwort, in geeigneter Weise ausgegeben werden. Zunächst geschieht dies nur auf Textterminalbasis, in späteren Kapiteln wird dies auch grafisch verwirklicht.
Damit Listener überhaupt erst verwendet werden können, muss dem Beispielprojekt noch die Paketquelle „org.eclipse.swt.events“ hinzugefügt werden. Dazu wird im Beispielprojekt, bei den vorhandenen „Import“-Befehlen, ein weiterer Befehl hinzugefügt.

```
import org.eclipse.swt.events.*;
```

Um einem Widget einen Listener hinzuzufügen, muss zunächst der passende Listener ausgesucht werden und dann mit der entsprechenden `add`-Methode dem Widget zugewiesen werden. Hier wird, um dem vorhandenen Button einen `mouseListener` zuzuweisen, die Methode `addMouseListener()` benötigt. Da ein Adapter eine Unterklasse eines Listeners ist, kann mit dieser Methode auch ein Adapter zugewiesen werden. Den Methoden des `mouseAdapters` wird dabei immer das `mouseEvent` von der `readAndDispatch()`-Methode übergeben. Das `Event`-Objekt enthält weitere Informationen zum Event, wie die Zeit des Eventauftrittes, die hier allerdings nicht benötigt werden. Innerhalb der `mouseDown()`-Methode wird eine weitere Methode aufgerufen, die die Nutzerdaten im Textterminal ausgeben soll.

```
button.addMouseListener (new MouseAdapter() {
    public void mouseDown(MouseEvent e) {
        printUserData();
    }
});
```

Nun könnte ein weiterer Listener hinzugefügt werden, der das Drücken der Enter-Taste abfragt. Da es jedoch einen Listener, den `SelectionListener`, gibt, der sowohl den Mausklick, als auch das Drücken der Taste, zusammen abfragt, ist es sinnvoller den `MouseListener` zu ersetzen. Der `SelectionListener` enthält die Methode `widgetSelected()`, die auf ein Auswählen des Widgets reagiert. Zu beachten ist, dass die Methode beim Drücken der Enter-Taste nur aufgerufen wird, falls das Widget fokusiert ist.

```
button.addSelectionListener (new SelectionAdapter() {
```

```
    public void widgetSelected(SelectionEvent e) {
        printUserData();
    }
});
```

Schließlich wird noch die Methode `printUserData()` benötigt, die mit Hilfe der `getText`-Methoden der erstellten Textfelder, den Inhalt der selben ausgibt. Zu beachten ist hierbei, das die Widgets `userT` und `passT` als Klassenvariablen deklariert werden müssen, damit die Methode auf sie zugreifen kann.

```
static void printUserData() {
    System.out.println("Benutzername: " + userT.getText());
    System.out.println("Password: " + passT.getText());
}
```

Insgesamt bleibt zu sagen, dass diese Verwirklichung keine Passwortabfrage ist, sondern lediglich eine Ausgabe der Werte der Textfelder. Der nötige Java-Programmcode macht jedoch im Rahmen dieser Ausarbeitung keinen Sinn, da er mit der direkten Programmierung von grafischen Oberflächen nichts zu tun hat.

2.5 Layouts

Grundlagen zu Layouts Bisher wurden die Widgets im Fenster positioniert, indem mit der `setBounds`-Methode explizit die Größe und die Position des Widgets im Fenster festgelegt wurde. Da dies umständlich ist und in Projekten mit vielen Widgets schwer umsetzbar ist, gibt es in SWT, wie auch in Swing und AWT, das Konzept der „Layouts“. Layouts können Widgets zugewiesen werden, die andere Widgets enthalten, und steuern dann die Anordnung der Widgets im Fenster. So entfällt nicht nur der Aufwand der Positionierung der Widgets, es gibt sogar praktische Zusatzfunktionen, wie die automatische Änderung der Skalierung von Widgets bei der Skalierung des Fensters. In den meisten Projekten, die Layouts verwenden, wird man nicht mit einem Layout auskommen, sondern mehrere kombinieren. Stellt man sich beispielsweise einen gewöhnlichen Texteditor vor, so gibt es in den meisten Fällen eine Menüleiste und einen Schreibbereich. Diese beiden Komponenten sind untereinander angeordnet, was einem Layout entspräche. Die Menüleiste selbst hat wiederum mehrere Komponenten, die nebeneinander angeordnet sind, was ein weiteres Layout wäre. So wird schnell deutlich, dass bereits bei scheinbar einfachen grafischen Oberflächen eine Vielzahl von Layouts zur Anwendung kommen kann.

Layouts am Beispiel Das vorhandene Programm mit Widgets und Events wird nun um Layouts erweitert, was die manuelle Positionierung der Widgets ersetzt. Dabei wird das `FillLayout` und, als Alternative, das `GridLayout` vorgestellt. Die Programmierung der Layouts ist aus [9] und [7] entnommen.
Um Layouts in SWT zu nutzen, muss zunächst das Paket „org.eclipse.swt.layout“

importiert werden. Dies geschieht, indem wir im Hauptprogramm zu den vorhandenen Import-Befehlen einen weiteren hinzufügen.

```
import org.eclipse.swt.layout.*;
```

Nun werden zunächst sämtliche `setBounds`-Aufrufe entfernt, da diese Aufgabe ab jetzt vom Layout übernommen wird. Anschließend wird nach der Erzeugung des `Shell`-Objekts diesem mit `setLayout()` ein Layout zugewiesen, wobei zunächst das `FillLayout` verwendet wird.

```
shell.setLayout(new FillLayout());
```

Nun hat die vorhandene `Shell` ein Layout, das die Widgets horizontal anordnet. Dabei wird die Größe der Widgets so festgelegt, dass der gesamte Platz ausgefüllt ist. Daher der Name `FillLayout`.
Das vorgestellte `FillLayout` ist zwar einfach, bietet allerdings auch nicht allzuviele Möglichkeiten. Um eine etwas bessere Kontrolle über die Anordnung der Widgets zu haben, wird nun das `GridLayout` vorgestellt. Die Anordnung von Widgets mit einem `GridLayout` ähnelt einer Tabelle, bei der Zellen zusammengefasst werden können. Bei der Erzeugung eines `GridLayouts` muss festgelegt werden, wie viele Widgets in eine Zeile passen. Dazu wird dem Konstruktor als erster Parameter die Anzahl der Spalten übergeben. Im Beispiel sollen jeweils ein Label und das dazugehörende Textfeld nebeneinander angeordnet werden. Daher wird die Spaltenanzahl auf zwei festgelegt. Die Widgets werden der Reihenfolge nach eingefügt, in der sie initialisiert werden. Zunächst muss also das Label „userL“ initialisiert werden, danach das Textfeld „userT“. Danach ist die Anzahl der Widgets in einer Zeile erreicht, also positioniert das `GridLayout` das nächste Widget, in diesem Fall das Label „passL“ in der nächsten Zeile. Als zweiter Konstruktorparameter wird ein `boolean` übergeben, der angibt ob jede Spalte den gleichen Platz einnehmen soll, oder nur so viel, wie für die Darstellung der Widgets nötig ist. Da die Widgets nicht über das ganze Fenster verteilt werden sollen, wird hier die zweite Option gewählt, für die „false“ übergeben werden muss.

```
GridLayout gLayout = new GridLayout(2, false);
shell.setLayout(gLayout);
```

Ob die Widgets vor oder nach dem Layout initialisiert werden, spielt für deren Anordnung keine Rolle.
Mit dem nun vorhandenen `GridLayout` sind zwar schon, im Gegensatz zum `FillLayout`, kompliziertere Anordnungen möglich, allerdings lässt sich das `GridLayout` zusätzlich noch mit Objekten vom Typ `GridData` erweitern. So ist es zum Beispiel möglich, ein Widget über mehrere Spalten zu positionieren oder die Anordnung der Widgets in den Spalten festzulegen. Im vorhandenen Beispiel soll der Button zum Absenden der Daten über beide Spalten hinweg positioniert werden, damit er mittig zwischen den Labeln und den Textfeldern positioniert werden kann. Dazu muss ein neues Objekt vom Typ `GridData` erzeugt werden und bei diesem die Attribute `horizontalSpan`

und `horizontalAlignment` angepasst werden. Schließlich wird dem Widget das `GridData`-Objekt mit der Methode `setLayoutData()` zugewiesen.

```
GridData gd = new GridData();
gd.horizontalAlignment = SWT.CENTER;
gd.horizontalSpan = 2;
button.setLayoutData(gd);
```

Mehrere , auch verschiedene, Layouts lassen sich innerhalb eines Programmfensters verschachteln, indem die Widgets auf mehrere `Composite`-Objekte verteilt werden und jedem `Composite` ein Layout zugewiesen wird.

3 Erweiterung von SWT mit JFace

3.1 Was ist JFace

Schon anhand des kleinen Beispielprogramms lässt sich erkennen, dass SWT zwar einen großen Umfang an Funktionen bietet, jedoch ebenso sehr viel Programmieraufwand mit sich bringt. Um die Programmierung etwas zu beschleunigen wurde „JFace“ entwickelt. JFace ist keine eigenständige API für grafische Oberflächen, sondern setzt auf SWT auf. Es ist eine Abstraktionsschicht zu SWT. Dies bedeutet, dass oft verwendeter SWT-Programmcode in JFace zu kleineren Modellen zusammengefasst wurde. Viele Komponenten in grafischen Oberflächen lassen sich somit in JFace leichter und komfortabler umsetzen als in SWT, dafür sind nicht alle Funktionen von SWT in JFace zugänglich. Für große Projekte wird daher eher JFace verwendet, wobei an vielen Stellen immer wieder der SWT-Programmcode verwendet wird, um spezielle Anforderungen umzusetzen [2].
JFace ist nicht in der SWT-Programmbibliothek enthalten, sondern verwendet eine extra Bibliothek. Die Einbindung der JFace-Bibliotheken gestaltet sich etwas schwierig, da sie von mehreren Paketen abhängen. Um in JFace ein funktionierendes Programm zu programmieren, müssen in einem Eclipse-Projekt in den „Java Build Path“ zum Punkt „Libraries“ aus dem Eclipse-Plugin-Verzeichnis noch die Plugins „org.eclipse.jface.*“, „org.eclipse.equinox.common.*“ und „org.eclipse.core.commands.*“ hinzugefügt werden, wobei * für die Version des jeweiligen Plugins steht. Da JFace SWT-Programmcode verwendet, sind die SWT-Bibliotheken auch nötig, um ein Programm mit JFace zu schreiben.

3.2 JFace-Funktionalitäten am Beispiel

Da JFace an sich schon ein sehr großes Thema ist und ganze Bücher füllen kann, wird hier nur ein kleines Programmbeispiel vorgeführt, um darzustellen wo JFace ansetzt und wie es verwendet wird. Das bisherige Programm soll so abgewandelt werden, dass beim Anklicken des Buttons die Nutzerdaten nicht mehr im Textterminal ausgegeben werden, sondern mit einem grafischen Dialogfenster. Dies wird zunächst in SWT umgesetzt und anschließend mit JFace.
Bevor auf die Programmierung des Dialogfensters eingegangen wird, ist es notwendig den Grundaufbau eines Programms zu zeigen, das JFace verwendet, da

sich dieser bereits vom SWT-Programm unterscheidet [2]. Zunächst müssen zwei Pakete importiert werden. Dies sind „org.eclipse.swt.widgets“ und „org.eclipse.jface..window.ApplicationWindow“. Als nächstes muss eine Klasse erstellt werden, die die abstrakte Klasse `ApplicationWindow` aus dem JFace-Paket implementiert. Anstatt, wie in SWT, ein `Display`- und ein `Shell`-Objekt zu erzeugen, wird bei JFace nur das `ApplicationWindow` implementiert. Innerhalb der Klasse wird zunächst der Konstruktor ausprogrammiert, der wiederum den Konstruktor der Oberklasse aufruft.

```
import org.eclipse.swt.widgets.*;
import org.eclipse.jface.window.ApplicationWindow;

public class JFaceProgram extends ApplicationWindow {
    public JFaceProgram () {
        super(null);
    }
```

Anschließend wird eine `run`-Methode erstellt, die beim Erstellen eines Objektes der Klasse aufgerufen wird und das Fenster anzeigt. Innerhalb der Methode werden die Methoden `setBlockOnOpen(true)` und `open()` der Klasse ApplicationWindow aufgerufen. Die erste Methode sorgt dafür, dass das Programm, nach dem Aufruf der `open`-Methode, so lange auf bleibt, der Programmablauf also geblockt wird, bis das Fenster vom Benutzer geschlossen wird. Die `open`-Methode zeigt das Fenster an. Diese beiden Methodenaufrufe erledigen das, was in SWT die Event-Schleife erledigt. Nach der `open`-Methode wird schließlich noch ein Methodenaufruf gemacht, der das Fenster mit der `dispose`-Methode zerstört.

```
public void run () {
    setBlockOnOpen(true);
    open();
    Display.getCurrent().dispose();
}
```

Statt, wie in SWT, den Programmcode zur Erzeugung von Widgets vor die Event-Schleife zu schreiben, wird bei JFace die Methode `createContents()` ausprogrammiert. Diese Methode wird beim Öffnen des `ApplicationWindow` aufgerufen. Ihr wird vom `ApplicationWindow`-Objekt ein `Composite`-Objekt übergeben, das als übergeordnetes Widget für alle, innerhalb von `createContent()` erzeugten, Widgets fungiert. Dieses Composite muss daher auch am Ende der Methode zurückgegeben werden.

```
protected Control createContents(Composite parent) {
     return parent;
}
```

Um das Anordnen der Widgets in JFace genauso zu realisieren wie in SWT, wird am besten ein weiteres Composite erstellt, dem dann alle anderen Widgets zugewiesen werden. Diesem Composite können dann, wie in Kapitel 2 beschrieben,

Layouts zugewiesen werden.
Schließlich fehlt nur noch eine Hauptmethode, die ein Objekt der Klasse erzeugt und diese mit `run()` startet. Zu beachten ist nun, dass beim Erzeugen von Widgets nicht mehr „shell“ als übergeordnetes Objekt, sondern das Application-Window, also „parent“, zugewiesen werden muss. Das gesamte JFace-Programm sieht dann folgendermaßen aus.

```
import org.eclipse.swt.widgets.*;
import org.eclipse.jface.window.ApplicationWindow;

public class JFaceProgram extends ApplicationWindow {
    public JFaceProgram () {
        super(null);
    }
    public void run () {
        setBlockOnOpen(true);
        open();
        Display.getCurrent().dispose();
    }
    protected Control createContents(Composite parent) {
         return parent;
    }
    public static void main (String args[]) {
        JFaceProgram program = new JFaceProgram();
        program.run();
}
```

Nun kann auf die Programmierung des Dialogfensters eingegangen werden. Ein Dialogfenster ist ein Fenster, das sich bei einer Aktion öffnet und den Programmablauf so lange unterbricht, bis der Benutzer den Dialog mit einem Klick auf einen Button beendet. In einem Texteditor erscheint beispielsweise oft ein Dialogfenster, das den Benutzer bei Beenden des Editors fragt, ob die bearbeitete Datei gespeichert werden soll. Das Editorprogramm wird dabei so lange unterbrochen, bis der Benutzer den Dialog per Klick auf einen „Ja“- oder einen „Nein“-Button beendet. In SWT gibt es verschiedene Arten von Dialogfenstern, wobei im vorhandenen Programm eine `MessageBox` erzeugt werden soll, also ein Dialogfenster, das rein informeller Natur ist und nur einen Bestätigungs-Button besitzt. Mit SWT sind mehrere Schritte nötig, um die `MessageBox` zu erstellen [7]. Zunächst wird ein neues Objekt vom Typ `MessageBox` erzeugt. Anschließend wird mit `setMessage()` der anzuzeigende Text eingestellt und schließlich noch mit `setText()` der Titel des Fensters. Das Dialogfenster wird mit seiner `open`-Methode angezeigt.

```
MessageBox mBox = new MessageBox(shell);
mBox.setMessage("Benutzername: " + userT.getText()
  + "\nPassword: " + passT.getText());
mBox.setText("Nutzerdaten");
```

```
mBox.open();
```

Der Quellcode muss natürlich entsprechend, im Bezug auf die Sichtbarkeit der einzelnen Variablen angepasst werden, je nachdem wo er platziert wird.
Mit JFace lässt sich das gleiche Dialogfenster mit nur einem Befehl ausprogrammieren [2].

```
MessageDialog.openInformation(parent.getShell(), "Nutzerdaten",
  "Benutzername: " + userT.getText() + "\nPassword:
  " + passT.getText());
```

Das fertige Beispielprogramm, mit dem Dialogfenster in JFace realisiert, würde nach Anklicken des Buttons folgender Abbildung entsprechen.

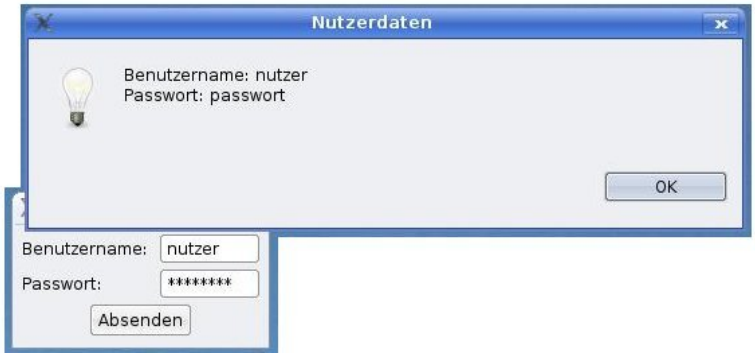


Abbildung 3. Beispielprogramm nach Anklicken des „Absenden“-Buttons unter Linux mit KDE 3.5

Anhand dieses Beispiels lässt sich bereits erahnen, dass für große Programme der Aufwand mit JFace, im Vergleich zum Aufwand reiner SWT-Programmierung, erheblich kleiner ausfällt. Da JFace nur SWT benutzt und abstrahiert, ist es des weiteren möglich bei Bedarf einfach wieder SWT zu verwenden.

4 Alternativen zu SWT und Vergleich

Wie bereits erwähnt, gibt es für Java noch zwei weitere APIs zur Erstellung von grafischen Oberflächen, AWT und Swing. Jede dieser Schnittstellen hat seine Vor- und Nachteile und keine konnte sich bisher endgültig als Beste etablieren. Je nach Aufgabenstellung kann jede der drei Schnittstellen ihre Berechtigung haben. Hier wird ein kleiner Vergleich dargelegt, ein weitaus ausführlicher Vergleich, von dem auch die hier genannten Informationen stammen, findet sich bei [5].

4.1 AWT

AWT steht für „Abstract Window Toolkit“ und wurde mit dem Java Development Kit 1.0 1995 herausgebracht. Seitdem ist es Teil der Java Bibliotheken [10, Kapitel 1]. Wie auch das SWT nutzt AWT die nativen Methoden des Betriebssystems, um grafische Komponenten zu zeichnen. Allerdings setzt es auf den kleinsten gemeinsamen Nenner aller Betriebssysteme. Dies bedeutet, dass in AWT nur die Komponenten vorhanden sind, für die es in allen unterstützten Betriebssystemen eine Komponente gibt. Da so wichtige Komponenten, wie ein Fortschrittsbalken, fehlen, reicht AWT häufig nicht aus, um gewünschte grafische Oberflächen zu programmieren. Das AWT benutzt, im Gegensatz zu SWT, „auto-disposal“. Komponenten müssen also nicht mühsam entfernt werden, sondern werden automatisch von der Java Virtual Maschine wieder gelöscht.
AWT arbeitet, im Gegensatz zu SWT, nicht nach dem Prinzip des „Top-Down Programming“. Komponenten müssen also beim Erstellen keine übergeordnete Komponente zugewiesen bekommen. Somit lässt sich der AWT-Programmcode flexibler und übersichtlicher programmieren.

4.2 Swing

AWT spielt in der GUI-Entwicklung unter Java, hauptsächlich aufgrund der Mängel im Funktionsumfang, heutzutage nur noch eine untergeordnete Rolle. Die beiden großen APIs sind Swing und SWT. Swing setzt jedoch auf AWT auf, weshalb dies nach wie vor erwähnt werden muss. Swing wurde 1997 mit der Java-Version 1.2 veröffentlicht [11, Kapitel 15.1.3]. Im Gegensatz zu SWT und AWT baut Swing nicht auf den nativen Betriebssystemmethoden auf, sondern zeichnet alle Komponenten selbst. Somit ist es die einzige richtige plattformunabhängige API für grafische Oberflächen unter Java. Da dieses Zeichnen aber deutlich rechenintensiver als das Verwenden der Betriebssystemmethoden ist, ist Swing merkbar langsamerer als SWT und AWT. Anzumerken ist hierbei, dass sich die Geschwindigkeit mit Java 1.5 jedoch deutlich verbessert hat.
Durch die Zeichnung, die von Swing übernommen wird, haben Programme, die mit Swing programmiert wurden, nicht das Look and Feel des Betriebssystems auf dem sie gerade laufen. Dies mag einerseits störend sein, da sie nicht zwangsläufig in das Bild der anderen Programme hineinpassen, andererseits ist mit Swing eine einfache globale Änderung des „Look and Feels“ über sogenannte Styles möglich.
Wie auch AWT, benutzt Swing das „auto-disposal“ der Komponenten und muss nicht zwingendermaßen nach dem „Top-Down“-Prinzip programmiert werden.

5 Fazit

Die SWT-Programmschnittstelle, die zusammen mit Eclipse entwickelt wurde, bietet die Möglichkeit grafische Oberflächen unter Java zu entwickeln. Dabei werden die nativen Methoden der jeweiligen Betriebssysteme aufgerufen, um

grafische Komponenten auf den Bildschirm zu zeichnen. Sind diese Komponenten im Betriebssystem nicht vorhanden, so werden sie emuliert. Gegenüber den anderen beiden Programmierschnittstellen, AWT und Swing, hat SWT den Vorteil, dass es sowohl schnell, durch den Aufruf der nativen Methoden, als auch vielfältig, durch die Emulation der nicht vorhandenen Komponenten, ist. Weiterhin passt es sich durch den Aufruf der nativen Methoden an das Aussehen der betriebssystemeigenen Programme an.
Die grafischen Komponenten in SWT „Widgets“ genannt werden nach dem „Top-Down-Prinzip“ programmiert. Jedem dieser Widgets muss also beim Erzeugen ein übergeordnetes Widget zugeordnet werden. Weiterhin muss bei jedem Widget angegeben werden, wo auf dem übergeordnetem Widget, es platziert werden soll. Dieses Konzept lässt sich mit Hilfe der „Layouts“, also automatischer Anordnungen nach einem bestimmten Schema, vereinfachen. Um auf Benutzereingaben zu reagieren, dienen in SWT die Events und die Event-Schleife. Zum Abfragen dieser gibt es abstrakte „Listener“- und die oft komfortableren „Adapter“-Klassen.
Mit Hilfe von „JFace“, das eine Abstraktionsschicht zu SWT bildet, lässt sich der Programmieraufwand, im Gegensatz zu SWT, erheblich reduzieren. Ist eine spezielle Funktion in JFace nicht vorhanden, so kann jederzeit auch wieder SWT zur Umsetzung dieser verwenden.
SWT, zusammen mit JFace, hat zwar viele Vorteile gegenüber den anderen beiden APIs, jedoch hat es sich nicht endgültig gegen sie durchgesetzt und es mag immer noch Stellen geben, an denen es mehr Sinn macht, AWT oder Swing zu verwenden.

Literatur

1. Wikipedia: Grafische Benutzeroberfläche. `http://de.wikipedia.org/wiki/Graphical_User_Interface` Version: 06.02.2008.
2. Harris, B., Warner, R., Harris, R.: The Definitive Guide to Swt and Jface. APress, Berkeley (2004)
3. SYS-CON Media: 'The Oscars of the Software Industry' @ JAVA DEVELOPERS JOURNAL. `http://java.sys-con.com/read/43939.htm` Version: 06.02.2008.
4. Scarpino, M.: SWT/JFace in action. Manning, Greenwich (2005)
5. Feigenbaum, B.: SWT, Swing or AWT: Which is right for you? `http://www.ibm.com/developerworks/opensource/library/os-swingswt/` Version: 06.02.2008.
6. Eclipse: Developing SWT applications using Eclipse. `http://www.eclipse.org/swt/eclipse.php` Version: 06.02.2008.
7. Hatton, T.: SWT. 1. edn. O'Reilly, Sebastopol (2004)
8. Ramachandran, S.: Basic SWT Widgets. `http://www.cs.umanitoba.ca/~eclipse/2-Basic.pdf` Version: 06.02.2008.
9. Ramachandran, S.: Eclipse Layouts. `http://www.cs.umanitoba.ca/~eclipse/4-Layouts.pdf` Version: 06.02.2008.
10. Zukowski, J.: Java AWT reference. O'Reilly, Sebastopol (1997)
11. Ullenboom, C.: Java ist auch eine Insel. 7., aktualisierte und erw. aufl. edn. Galileo Press, Bonn (2008)