Blau, Holger; Eicker, Stefan; Hofmann, Alexander; Spies, Thorsten

**Research Report**

# Reifegradüberwachung von Software

ICB-Research Report, No. 20

**Provided in Cooperation with:**
University Duisburg-Essen, Institute for Computer Science and Business Information Systems (ICB)

*Suggested Citation:* Blau, Holger; Eicker, Stefan; Hofmann, Alexander; Spies, Thorsten (2007) : Reifegradüberwachung von Software, ICB-Research Report, No. 20, Universität Duisburg-Essen, Institut für Informatik und Wirtschaftsinformatik (ICB), Essen

This Version is available at:
https://hdl.handle.net/10419/58161

ICB
Institut für Informatik und Wirtschaftsinformatik

Holger Blau
Stefan Eicker
Alexander Hofmann
Thorsten Spies

# Reifegradüberwachung von Software

20

ICB-RESEARCH REPORT

UNIVERSITÄT
DUISBURG
ESSEN

ICB-Research Report No. 20
August 2007

Die Forschungsberichte des Instituts für Informatik und Wirtschaftsinformatik dienen der Darstellung vorläufiger Ergebnisse, die i. d. R. noch für spätere Veröffentlichungen überarbeitet werden. Die Autoren sind deshalb für kritische Hinweise dankbar.

The ICB Research Reports comprise preliminary results which will usually be revised for subsequent publications. Critical comments would be appreciated by the authors.

---



---

**Authors' Address:**

Holger Blau, Stefan Eicker, Thorsten Spies

Universität Duisburg-Essen
D-45117 Essen
Germany

holger.blau@online.de
stefan.eicker@icb.uni-due.de
thorsten.spies@icb.uni-due.de

Alexander Hofmann

sd&m AG
81739 München
Germany

alexander.hofmann@sdm.de



---

**Managing Assistant and Contact:**

Jürgen Jung

Institut für Informatik und
Wirtschaftsinformatik (ICB)
Universität Duisburg-Essen
Universitätsstr. 9
45141 Essen
Germany

Email: icb@uni-duisburg-essen.de

**ISSN 1860-2770**

# Abstract

Durch die wachsende Bedeutung von Softwaresystemen (Stichwort: „Softwareintensive Systeme") beeinflusst die Softwareindustrie zunehmend auch andere Industriezweige. Aber obwohl sie sich damit zu einer „Leitindustrie" entwickelt hat, befindet sich ihre Entwicklung - zumindest im Vergleich zu anderen Industrien - noch in einem relativ unreifen Stadium. Angesichts der immer härteren Rahmenbedingungen, die sich insbesondere in einem hohen Termindruck, Budgetrestriktionen, einem dynamischen Umfeld und in der steigenden Komplexität von Software ausdrücken, ist eine Professionalisierung der Entwicklungsmethodik aber zwingend erforderlich, auch, um eine Verschlechterung der Produktqualität zu vermeiden. Das Voranschreiten der Industrialisierung der Softwareentwicklung bildet somit eine zentrale Herausforderung für die Softwarebranche.

Im vorliegenden Beitrag wird zur Verbesserung der Möglichkeiten eines kontinuierlichen Monitorings von Softwareprojekten eine Methode vorgestellt, die eine Aussage über die „Reife" des aktuellen Entwicklungsstands einer Software ermöglicht. Das Ziel der Methode besteht darin, die Eigenschaften der Produktqualität transparent zu machen, um auch unter den sich verschärfenden Rahmenbedingungen qualitativ hochwertige Software entwickeln zu können. Den Bewertungsgegenstand der Methode bildet demzufolge nicht – wie bei vielen Ansätzen - der zugrunde liegende Entwicklungsprozess, sondern das zu entwickelnde Produkt, die Software. Als Vergleichsmethodik wird in dem Beitrag die „Reifegrad-Absicherung für Neuteile" vom Verband der Automobilindustrie (VDA) herangezogen, da dieses Verfahren das konsolidierte Ergebnis in einem Industriezweig darstellt, der im Zuge der Industrialisierung eine kontinuierliche Verbesserung von Prozessen und Methoden erfahren hat.



# Acknowledgments

Dieser Beitrag bildet das Ergebnis einer Kooperation der sd&m AG (sd&m Research) und des Lehrstuhls für Wirtschaftsinformatik und Softwaretechnik der Universität Duisburg-Essen. Auf Seiten der sd&m AG ist das entsprechende Engagement in das Bestreben des Unternehmens einzuordnen, sich dem Thema *Produktqualität in der Softwarebranche* im Rahmen des Projekts „Software-Controlling" (Gesamtleitung: Alexander Hofmann) systematisch und zielführend zu widmen.


# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# 1 Einleitung

Die Softwareentwicklung befindet sich auf dem Weg der Industrialisierung. Software wird heute verteilt entwickelt, die Rollen im Softwareentwicklungsprozess spezialisieren sich und weltweit schreiten die Standardisierungsbemühungen voran. Getrieben wird diese Industrialisierung insbesondere durch den hohen Termindruck und starke Budgetrestriktionen in den Softwareentwicklungsprojekten. Zudem erfordern kurze Entwicklungszyklen (zwei bis drei Releases pro Jahr) und eine zunehmende fachliche Komplexität der Systeme eine Professionalisierung der Entwicklungsmethodik. Außerdem wählen IT-Entscheider Softwarelieferanten heute nicht zuletzt danach aus, wie schnell sie in der Lage sind, eine stabile Lösung zu liefern.

Durch die schwierigen Rahmenbedingungen der Softwareentwicklung besteht die Gefahr, dass sich die Qualität der entwickelten Software verschlechtert. So müssen inzwischen sechzig bis siebzig Prozent der IT-Budgets in Wartung und Weiterentwicklung investiert werden. Selbst ein funktionierendes System ist keine Garantie für eine gute Qualität, da weitere Qualitätskriterien wie Wartbarkeit und Flexibilität zunehmend an Bedeutung gewinnen. Die Herausforderung in den nächsten Jahren wird darin bestehen, bezüglich aller relevanten Kriterien qualitativ hochwertige Software unter den zunehmend härter werdenden Rahmenbedingungen zu erstellen.

In den letzten Jahren haben sich etliche Verfahren zur Beurteilung und Optimierung von Softwareentwicklungsprozessen etabliert (CMMI, etc.). Was jedoch fehlt, sind Ansätze zur Beurteilung von Softwareproduktreife, da selbst durch eine optimale Gestaltung der Entwicklungsprozesse eine hohe Produktqualität nicht garantiert werden kann. Nur die Transparenz über Prozess- und Produktreife und das Verständnis über deren Wechselwirkungen ermöglicht die Steuerung von erfolgreichen Softwareprojekten.

Klassische Industrien (wie z.B. der Automobilbau) sehen sich schon seit vielen Jahren mit erschwerten Marktbedingungen konfrontiert. Hier konnten folgende Entwicklungen beobachtet werden:

- Steigende Produktkomplexität: Die starke Nachfrage zu Innovationen im Fahrzeug führt zu stärkerer Vernetzung der einzelnen Komponenten. Variantenfertigung erhöht die Vielfalt. Die Zahl der Steuergeräte steigt.
- Kürzere Produktlebenszyklen: Es steht immer weniger Zeit für die Anläufe neuer Baureihen zur Verfügung. Die Anlaufkurven werden steiler.
- Zunehmender Wettbewerb: Durch Globalisierung entsteht verstärkter internationaler Konkurrenzdruck.
- Verteilte Entwicklung: Die Entwicklungs- und Fertigungstiefe verschiebt sich vom Autohersteller zum Lieferanten.

Letztlich führten diese Rahmenbedingungen zu einer Absenkung der Produktqualität, was sich in steigenden Gewährleistungs- und Kulanzkosten auswirkte. Aus diesem Grund entwickelten mehrere Automobilhersteller gemeinsam ein Verfahren zur „Reifegrad-Absicherung für Neuteile" [VDA 2005].

Die Einflussfaktoren des Automobilbaus sind an vielen Stellen sehr ähnlich zu denen der Softwareentwicklung. Deshalb wurde untersucht, welche Konzepte der Automobilindustrie sich auf die Softwareindustrie übertragen lassen. Das Ziel der Untersuchung bestand darin, einen systematischen Ansatz zur permanenten Reifegradbeurteilung eines Softwaresystems zu entwickeln. Tatsächlich konnte eine Methode für das Reifegradcontrolling von Software abgeleitet werden; neben Erkenntnissen aus dem Reifegradmanagement in der Automobilindustrie berücksichtigt sie auch aktuelle

Ansätze zur Steuerung von Codequalität (z.B. Metriken). Die Betrachtungsebene der Methode bildet die Komponentenebene.

Im Zentrum der Methode steht die Transparenz der Produktqualität. Das Monitoring der Produktreife erfolgt über ein Reifemodell.

# 2 Zugrundeliegende Begriffe und Methoden

## 2.1 Grundlagen

Der Begriff „Qualität" wird im Kontext von Software unterschiedlich ausgelegt. In der ISO Norm 8402 wird Qualität folgendermaßen beschrieben:

> „Qualität ist die Gesamtheit von Merkmalen einer Einheit bezüglich ihrer Eignung, festgelegte und vorausgesetzte Erfordernisse zu erfüllen." [DIN1995, 9]

Etwas differenzierter ist die Definition für Softwarequalität im IEEE-Standard 729 formuliert. Sie beschreibt unterschiedliche Blickwinkel: [Wall2001, 12f.]

1) „The totality of features and characteristics of a software product that bear on its ability to satisfy given needs; for example, conform to specifications."
2) „The degree to which software possesses a desired combination of attributes."
3) „The degree to which a customer or user perceives that software meets his or her composite expectations."
4) „The composite characteristics of software that determine the degree to which the software in use will meet the expectations of the customer."

Beide Ansätze betonen den Grundsatz, dass Qualität einen Soll-Ist-Vergleich zwischen den Anforderungen an ein Produkt und seinen Eigenschaften darstellt. Dabei wird deutlich, wie maßgeblich ein Qualitätsurteil vom verwendeten Qualitätsmodell und den sich daraus ableitenden Anforderungen beeinflusst wird. Die Qualitätsmodelle operationalisieren den Begriff Software, indem sie Qualitätsmerkmale und Indikatoren ableiten. Aufbauend auf den Qualitätsmodellen kann dann in der praktischen Anwendung eine Zuordnung von Metriken erfolgen.

Die existierenden Qualitätsmodelle lassen sich in zwei Gruppen einteilen. Auf der einen Seite steht die Bewertung von Produkten im Mittelpunkt, auf der anderen Seite die Bewertung von Produktionsprozessen. Bekannte produktorientierten Ansätze wurden von McCall – er unterscheidet elf Merkmale der Softwarequalität – und von Boehm – er definiert siebzehn Merkmale - entwickelt [Mili2005, 6ff.]. Weitere Ansätze, die in Unternehmen angewendet werden, sind z.B. CUPRIMDSO bei IBM [CSMD2001, 228] oder FURPS+ bei Hewlett Packard [Grad1992, 205ff.]. Außerdem wurde die ISO Norm 9126 definiert; sie basiert auf den Modellen von Boehm und McCall und definiert insgesamt 6 Merkmale [ISO2001, 7].

Zu den bekannten und weit verbreiteten prozessorientierten Ansätzen zählen die ISO 9001, CMM(I) und SPICE: Die ISO 9001 legt Vorgaben für Qualitätsmanagement- bzw. Qualitätssicherungssysteme fest und erlaubt es, die damit verbundenen Prozesse zu zertifizieren [Thal2000, 45]. Das Capability Maturity Model (Integration) kurz CMM(I) ist ein Reifemodell, das einer Organisation ermöglicht, seine Softwareprozesse zu definieren und diese weiterzuentwickeln [Oreg2002, 129]. Software Process Improvement and Capability dEtermination (SPICE) stellt eine Norm zur Bewertung und Verbesserung von Softwareprozessen dar. Es greift dabei bestehende Ansätze, zu denen unter anderem auch die ISO 9000 Familie und CMM gehören, auf und entwickelt sie weiter [MeHS1998, 109ff.].

Bei den prozessorientierten Ansätzen wird anstelle des Begriffs „Qualität" der Begriff „Reifegrad" als Prüfziel verwendet. Unter Reife(grad) wird die Qualität des Entwicklungsprozesses verstanden. Ein qualitativ hochwertiger Prozess stellt CMM(I) zu Folge die Produktion hochwertiger Produkte sicher.

> „Die Verfasser des CMM stellen […] einen Zusammenhang zwischen der Reife und der Effektivität eines Softwareprozesses her. Sie behaupten, je reifer ein Softwareprozeß sei, desto höher sei die Qualität der entwickelten Produkte, desto kürzer die Entwicklungszeiten und desto niedriger die Kosten." [MeHS1998, 93]

Es existieren zudem Definitionen des Begriffes „Reifegrad", die sich stärker auf das Produkt konzentrieren. Beim „Produkt-Reifegrad" werden im Entwicklungsprojekt Indikatoren festgelegt, die regelmäßig bezüglich ihrer Risiken bewertet werden [Wißl2005, 42f.]. Dieses Reifegradverständnis stellt das Produkt in den Vordergrund, orientiert sich aber immer noch stark am Prozess.

Im Gegensatz zu den bestehenden Qualitäts- und Reifegradansätzen wird als Grundlage der Methode „Reifegradüberwachung von Software" eine eigene Definition von Produktreife aufgestellt:

> „Produktreife, zu einem Zeitpunkt X der Softwareentwicklung, bestimmt sich über die Erfüllung zentraler Softwarequalitätsmerkmale, dem Fortschreiten der Entwicklung entsprechend der Projektvorgaben und der Wahrung von Strukturrichtlinien."

Deutlich hervorzuheben ist hierbei der Unterschied zwischen Softwarereife und Softwarequalität. Außerdem liegt der alleinige Fokus der Betrachtung auf dem Produkt. Formal wird somit Softwarereife definiert als:

*SOFTWAREREIFE = SOFTWAREQUALITÄT – MS1 + MS2*

MS1 (Merkmalssatz 1) umfasst dabei eine Anzahl an Merkmalen von Softwarequalität, die im Sinne eines qualitativ hochwertigen Produktes wichtig erscheinen, aber für die Erstellung eines reifen Produkts nicht notwendig sind. MS2 (Merkmalssatz 2) beinhaltet in der Softwarequalität nicht enthaltene Aspekte, wie z.B. den Entwicklungsstand eines Produktes (entwickelt sich das Produkt wie erwartet).

Softwarereife basiert einerseits auf einem existierenden Qualitätsmodell, das um Merkmale, die für Reife nicht relevant sind, verringert wird. Andererseits werden zusätzliche Merkmale hinzugefügt, die in den klassischen Qualitätsmodellen nicht explizit definiert sind.

Unabhängig von der Zusammensetzung der enthaltenen Merkmale in Reife und Qualität wird unter Reife zudem immer ein Soll-Ist-Vergleich zu einem Zeitpunkt X in der Entwicklung einer Software verstanden. Somit ist es möglich, dass das Ergebnis eines Reifemerkmals einen niedrigen Qualitätslevel erreicht und trotzdem als reif gilt. Dies erklärt sich daraus, dass zu diesem Zeitpunkt der Entwicklung mit keinem höheren Qualitätslevel zu rechnen ist und somit der erreichte Level trotzdem als „reif" bewertet werden kann.

Abbildung 1 verdeutlicht die Zusammenhänge: Die Farben der einzelnen Säulen zu jedem Meilenstein in der Entwicklung verdeutlichen die Bereiche, in denen das gemessene Ergebnis zu einem Urteil niedrige, mittlere, oder hohe Qualität bzw. Reife führt. Während die Bewertungsintervalle bei Reife sich im Verlauf der Entwicklung verändern, basiert das Qualitätsurteil immer auf den gleichen Intervallen.

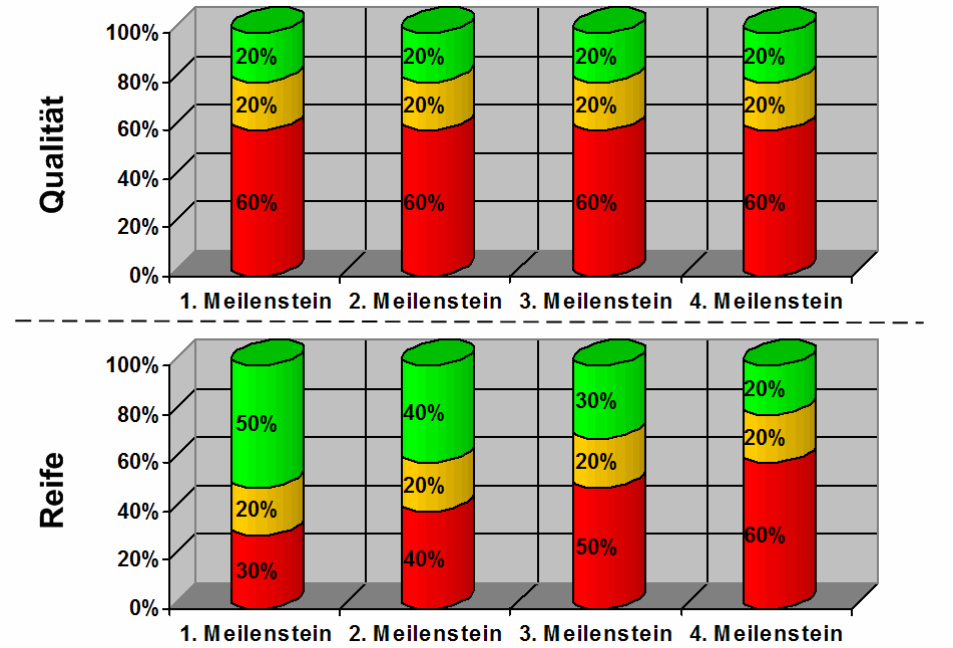



**Abbildung 1: Reife vs. Qualität**

In Tabelle 1 werden Messwerte den in Abbildung 1 definierten Intervallen für Reife und Qualität zugeordnet. Es ist zu erkennen, dass sich Reife an dem Fortschreiten der Entwicklung in einem Softwareprojekt orientiert.

| Meilenstein | 1. | 2. | 3. | 4. |
|---|---|---|---|---|
| Messwert (in %) | 55 | 65 | 75 | 85 |
| Reife | hoch | hoch | hoch | hoch |
| Qualität | niedrig | mittel | mittel | hoch |

**Tabelle 1: Reife- und Qualitätsverständnis**

Bei der Methode „Reifegradüberwachung von Software" handelt es sich demnach um einen produktorientierten Reifegradansatz, der ähnlich wie die bestehenden prozessorientierten Ansätze nicht nur Merkmale und Kriterien aufstellt, sondern diese auch zu einem Reifeurteil verdichtet. Etwas Vergleichbares findet sich auf Seiten der produktorientierten Ansätze nicht, denn diese zielen darauf ab, Merkmale für Qualität zu definieren, diesen Kennzahlen zuzuordnen und regelmäßig zu ermitteln, um den Verlauf der Entwicklung im Auge zu behalten.

Bei der Methode „Reifegradüberwachung von Software" wird für die Bewertung von Software auf die Ebene der Softwarekomponenten zurückgegriffen. Der Begriff der Komponente wird im Kontext der Softwareentwicklung unterschiedlich ausgelegt: McIlroy forderte in Anlehnung an andere Industriezweige auch in der Softwareindustrie den Einsatz von Komponenten [McIl1968, 138ff.]. Inzwischen existieren zahlreiche Definitionen in der Literatur. Eine der bekanntesten Definitionen für Softwarekomponenten stammt von Szyperski, der diese 1996 auf der European Conference on Object-Oriented Programming vorstellte.

> „A software component is a unit of composition with contractually specified interfaces and context dependencies only. A software component can be deployed independently and is subject to composition by third parties." [Szyp2002, 41]

In dieser Definition werden die zentralen Eigenschaften einer Komponente skizziert. Dazu zählen die fest zugesicherten Interfaces, über die eine Komponente mit seiner Umwelt interagiert. Außerdem wird darauf hingewiesen, dass eine Komponente unabhängig entwickelt und von Dritten zusammengesetzt wird. Die Komponente selber ist nach Szyperski's Definition eine „Unit of Composition", also eine Zerlegungs- bzw. Gliederungseinheit.

Lewandowskis Definition einer Softwarekomponente berücksichtigt weitere Aspekte. Nach seiner Definition ist eine Softwarekomponente der kleinste selbstverwaltende, unabhängige und nützliche Teil eines Systems [Lewa1998, 10]. Er weist sowohl auf den Zusammenhang innerhalb von Komponenten hin als auch auf den direkten Nutzen, den sie im Gegensatz zu einzelnen Klassen stiften.

Der im Rahmen von Quasar (Qualitätssoftwarearchitektur), einer Standardarchitektur für betriebliche Informationssysteme im Softwareentwicklungsunternehmen sd&m, geprägte Komponentenbegriff [Sied2003b, 99ff.] weist auf die Bedeutung der Komponente als Planungs- und Kontrollinstrument hin.

> „Die Komponente ist eine sinnvolle Einheit des Entwurfs, der Implementierung und damit der Planung." [Sied2003a, 4]

Da es sich bei den Komponenten um logische, funktional sinnvolle Elemente einer Software handelt, ist deren Anzahl deutlich geringer als die der Klassen. Die einzelnen Komponenten fassen jeweils größere, zusammenhängende Gruppen an Klassen zusammen. Wenn Elemente eines Systems beurteilt werden, so ist es sinnvoller, größere, eindeutig mit Funktionen verknüpfte Elemente zu beurteilen als eine Masse an kleineren Elementen. Diese ungeeignete Granularität von Klassen für Planungs- und Kontrollaufgaben stellen auch Gröttrup und Tensi in ihrer Veröffentlichung zur Verwendung von Komponenten in UML fest.

> „Es zeigt sich, dass Klassen sehr gut geeignet sind, um Informationen zu kapseln […]. Sie sind aber viel zu feinkörnig, um komplexere Softwaresysteme zu strukturieren" [GrTe2005, 90].

Sie siedeln beide Gestaltungselemente auf unterschiedlichen Abstraktionsebenen einer Softwarearchitektur an.

> „Klassen sind die Mikroarchitektur, Komponenten bilden die hierarchische Makroarchitektur der Anwendung." [GrTe2005, 91]

Für die Verwendung von Komponenten bei der Entwicklung einer Software sprechen, im Vergleich zur klassischen objektorientierten Entwicklung, Vorteile wie kürzere Entwicklungszeiten, höhere Qualität und niedrigere Preise [CrLL2000, 1][KiSc2003, 15]. Obwohl diese Punkte auf den Bedarf der Entwicklung von Bewertungsmethoden hindeuten, ist der tatsächliche wissenschaftliche Entwicklungsstand immer noch gering [NaHe2003, 3][GoBr2004, 2].

## 2.2 Die Referenzmethode VDA

Bei der „Reifegrad-Absicherung für Neuteile" handelt es sich um eine vom Verband der Automobilindustrie e.V. (VDA) veröffentlichte Methode, die den aktuellen Stand der Erkenntnisse der an der Veröffentlichung beteiligten Automobilunternehmen widerspiegelt [VDA2005, 10]. Damit stützt sich das enthaltene Erfahrungswissen auf alle am VDA beteiligten Unternehmen und repräsentiert die Best Practices der Automobilindustrie zur Reifegradbemessung. Die Methode ist Teil der Veröffentlichungsreihe „Qualitätsmanagement in der Automobilindustrie" und der aktuellen VDA 6.3 Norm. Ihr Ziel ist *„eine durchgängige Darstellung des Reifegrades im Rahmen des Projektablaufs für neue Produkte und Prozesse verbunden mit der objektiven Beurteilung der Produkt und Fertigungsprozess-Reife zu vereinbarten Zeitpunkten innerhalb des Produktrealisierungsprozesses."* [VDA2005, 10] Ausgelegt ist sie sowohl auf interne als auch auf externe Kunden-Lieferanten-Beziehungen. Sie besteht aus zwei zentralen Phasen:

*Phase 1 der VDA*

In der ersten Phase der VDA-Reifegrad-Absicherung werden reifegradkritische Umfänge innerhalb der Lieferkette identifiziert, um den Betrachtungsumfang in einem angemessenen Rahmen zu halten [VDA2005, 12-14]. Zu Beginn wird ein einheitlicher Kriterienkatalog verwendet, um eine Einschätzung der Kritikalität treffen zu können. Durch eine Beurteilung der einzelnen Lieferumfänge anhand der Kriterien erfolgt eine Risikoklassifizierung in A- (hohes Risiko), B- (mittleres Risiko) und C-Teile (niedriges Risiko). Diese Klassifizierung erfolgt auf allen Stufen der Lieferkette. Die Reifegradbeurteilung wird dann bei allen Lieferumfängen, die mit einer Risikoeinstufung von „A" (hohes Risiko) versehen sind, durchgeführt [VDA2005, 31].

*Phase 2 der VDA*

In der zweiten Phase der eigentlichen Reifegradbemessung werden die ausgewählten, mit einem hohen Reifegradrisiko versehenen Lieferumfänge anhand von Messkriterien bewertet und ein Reifegradurteil gefällt. Bei dieser Messung werden sowohl die Kunden als auch die Lieferanten in den Prozess der Bewertung mit eingebunden [VDA2005, 15-18], um ihnen Möglichkeiten zum frühzeitigen Eingreifen bei Fehlentwicklungen zu bieten. Die Reifegrad-Absicherung basiert auf einer Reifegrad-Meilenstein-Philosophie, die sieben verschiedene Reifegrade unterscheidet (RF 0-6) [VDA2005, 3]. Diese Reifegrade geben die Übereinstimmung mit Produkt-, Prozess- und Projektreifevorgaben zum Zeitpunkt des jeweiligen Meilensteines an. Die Reifegrade bauen aufeinander auf und basieren auf der Erfüllung von fest definierten Kriterien. Jede Phase unterteilt sich in Vorbereitung, Bewertung und Umsetzung (vgl. Abbildung 2).

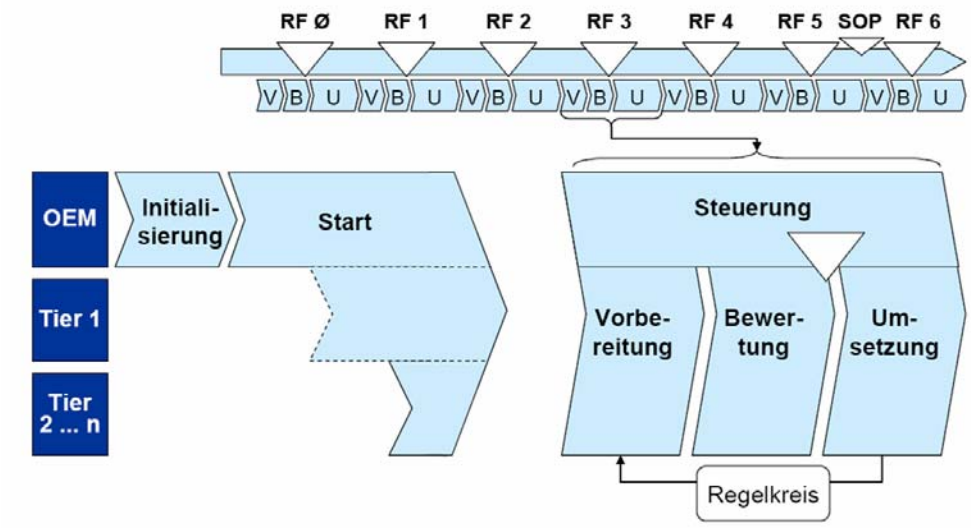



**Abbildung 2: Phasenmodell der Reifegrad-Absicherung [VDA2005, 15]**

Im Schritt Bewertung wird das Reifegradurteil ermittelt. Die Abstände der Meilensteine im Projektzeitplan sind abhängig vom Unternehmen, der Hierarchie in der Lieferkette und der Komplexität des Lieferumfangs. Die einzelnen Reifegrade enthalten Indikatoren, denen Messkriterien zugeordnet sind [VDA2005, 20]. Als Ergebnis der Bewertung werden die einzelnen Messkriterien für Reifegrade in Form einer Ampel veranschaulicht. Die Messkriterien sind so definiert, dass sie nur mit „ja" oder „nein" beantwortet werden können. Je nachdem, wie viele Messkriterien mit „ja" oder „nein" beantwortet und wie viele Projektziele erreicht werden, fällt das Urteil auf der Ampel aus [VDA2005, 26f.]. Ampeln vorangegangener Phasen werden auch in späteren Projektphasen mit angegeben.

Alle Reifegrad-Ampeln, die in der vorangegangenen Phase noch nicht mit „grün" bewertet wurden (so genannte offene Fragen), werden in der nachfolgenden Phase erneut ermittelt und ergeben so eine Reifegrad-Kaskade. Einmal mit „grün" bewertete Ampeln werden nicht neu bewertet, es sei

denn es kommt zu *„Projektziel beeinflussenden Änderungen“* [VDA2005, 27]. Am Ende des Projektes werden insgesamt sieben Ampeln für alle Reifegradurteile angegeben, die im Fall eines positiven Projektergebnisses alle „grün“ sind.

# 3 Methode „Reifegradüberwachung von Software“

## 3.1 Einschränkungen und Prinzipien der Methode

Die Methode „Reifegradüberwachung von Software“ ermöglicht es, Aussagen über die Reife des aktuellen Entwicklungsstandes einer Software zu treffen. Bewertungsgegenstand ist nicht der Entwicklungsprozess, sondern das zu entwickelnde Produkt - die Software. Dadurch grenzt sich die Methode von Bewertungsansätzen wie z.B. CMMI oder SPICE ab. Wie bereits in Kapitel 2 erwähnt, ist der Bewertungsgegenstand innerhalb der Software die Komponente, die bei größeren Softwareprojekten, im Rahmen der Architekturentwicklung, die unterste Gliederungsebene darstellt.

Bei der Entwicklung der Methode wurde der Fokus auf die spätere Methodenakzeptanz gelegt. Hierzu wurden Prinzipien formuliert, die dem Auftreten von Akzeptanz oder Verständnisproblemen entgegenwirken. Als Akzeptanzkriterien für den Aufbau eines Metrikenprogramms identifizieren Umarji und Emurian die vier Hauptkriterien Brauchbarkeit, Benutzerfreundlichkeit, Kontrolle und Verhalten in ihrem „Metrics Acceptance Model“ [UmEm2005, 12ff.]. Diese wurden beim Aufbau der Prinzipien für diese Methode berücksichtigt.

Folgende Prinzipien stehen bei der Methode im Vordergrund:

- Einfachheit
- Selbsterklärende Begrifflichkeiten
- Allgemeingültigkeit
- Angemessener Zeit- und Ressourceneinsatz
- Kontrollierbarkeit
- Orientierung an etablierten Verfahren aus anderen Industriezweigen
- Auswahl von geeigneten Bewertungskennzahlen aus der Softwareindustrie
- Durchgängigkeit des Bewertungsansatzes

*Einfachheit* bezieht sich auf den Aufbau der Methode und deren Struktur, um sowohl die Anwendung als auch die Interpretation der Methode und deren Ergebnisse für den Anwender zu erleichtern. Das resultiert auch in der Forderung nach *selbsterklärenden Begrifflichkeiten*, deren Ziel sich bereits aus der Bezeichnung ableiten lässt. Diese ersten beiden Prinzipien, gehen aus dem Grundgedanken der Benutzerfreundlichkeit des Metrics Acceptance Models hervor.

*Allgemeingültigkeit* zielt darauf ab, die Methode nicht auf einen speziellen Anwendungstyp auszurichten. Der *angemessene Zeit- und Ressourceneinsatz* muss immer berücksichtigt werden, da eine Methode deren Zeit- oder Ressourcenverbrauch zu hoch ausfällt, in der Praxis keine Verwendung finden wird. Diese beiden Prinzipien entsprechen dem Grundprinzip der Brauchbarkeit.

Das Prinzip *Kontrollierbarkeit* wurde aus dem Metrics Acceptance Model übernommen und drückt die Möglichkeit des Nutzers aus, die Methode zu kontrollieren und sie seinen Anforderungen entsprechend zu konfigurieren. Die Methode *orientiert* sich an der Vergleichsmethodik der VDA, um Erfahrungswissen aus anderen Industriezweigen zu integrieren. Hiermit wird auch dem Prinzip des Verhaltens aus dem Metrics Acceptance Model entsprochen. Die *Auswahl der Bewertungskennzahlen* für die Software wird auf Basis von Erfahrungswissen, Studien und den speziellen Anfor-

derungen der Methode getroffen. Mit diesem Prinzip wird die Brauchbarkeit der Methode aufgegriffen, da auf etablierte Verfahren gesetzt wird, um die Interpretier- und Nutzbarkeit der Ergebnisse sicherzustellen.

Die *Durchgängigkeit* des Bewertungsansatzes beinhaltet die Idee, möglichst früh mit der Bewertungsmethode anzufangen und diese dann konsequent im weiteren Projektverlauf anzuwenden.

## 3.2 Phasen der Methode

Softwareentwicklungsprojekte sind in unterschiedliche Phasen eingeteilt. Sie erstrecken sich vom Anforderungsmanagement, über die Konstruktion bis hin zum Betrieb und zur Wartung der Software. Die Methode „Reifegradüberwachung von Software" ist in drei Phasen unterteilt, die den Phasen des Softwareentwicklungsprozesses zugeordnet werden.

In der Vorphase „Risikoanalyse" (Phase 0) werden ausschließlich Informationen gesammelt, die erste Anhaltspunkte zur Identifikation kritischer Komponenten liefern können. Die Phase „Identifikation von kritischen Komponenten" (Phase 1) dient zur Identifikation der Komponenten einer Softwarearchitektur, die kritisch für den Erfolg des Softwareprojektes sind. In der Phase „Reifegradcontrolling" (Phase 2) erfolgt eine kontinuierliche Messung des Reifegrades der ausgewählten Komponenten. Die Methode ist auch in stark iterativen Entwicklungsprozessen einsetzbar, wobei insbesondere frühe Iterationen besonders wichtig sind. Abbildung 3 zeigt den schematischen Ablauf der Methode im Entwicklungsprozess für eine Iteration.

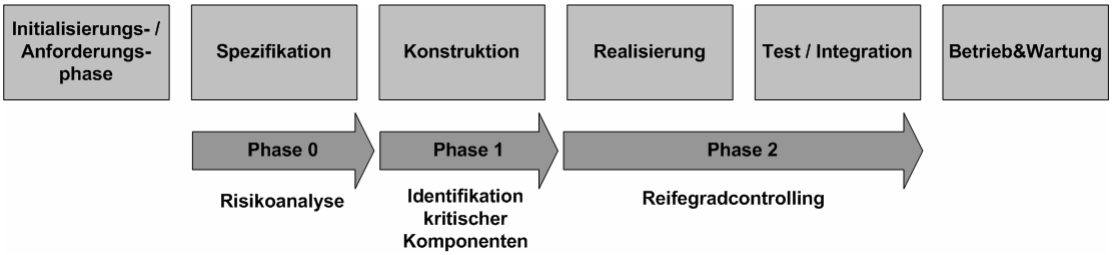



**Abbildung 3: Ablauf der Methode im Projekt**

Die gewählte Einteilung in Phasen orientiert sich an der VDA Reifegrad Absicherung (vgl. Kapitel 2) die ebenfalls erst eine Auswahl kritischer Lieferumfänge trifft, bevor sie diese kontinuierlich auf ihren Reifegrad überprüft. Eine Erweiterung zur Referenzmethode stellt die Phase 0 dar, die keine expliziten Schritte formuliert, sondern versucht, fachliche Informationen mit erhöhtem Risikopotenzial zu sammeln.

### 3.2.1 Phase 0: Risikoanalyse

In der Phase „Risikoanalyse" werden Informationen gesammelt, die in der nachfolgenden Phase 1 bei der Identifikation kritischer Komponenten helfen und zudem den Aufwand reduzieren sollen. Voraussetzung hierfür ist das Vorliegen eines Spezifikationsdokumentes, in dem möglichst umfassend alle wichtigen Eigenschaften einer Software, deren funktionalen und nicht funktionalen Eigenschaften sowie Test- und Dokumentationsanforderungen beschrieben werden [Scha2001, 35-37] [SSBS2004, 17] [PTLS2005, 11-12].

Der genaue Aufbau einer Spezifikation kann je nach Projekt und verwendeten Techniken unterschiedlich aussehen. Folgende Elemente finden sich jedoch in jeder Spezifikation wieder [Sied2003b, 24]:

- Daten- bzw. Objektmodell (beschreibt Datenstrukturen des Softwaresystems)
- Funktionsmodell (beschreibt Anwendungsfälle und Abläufe)
- Beschreibung der Benutzerschnittstelle
- Beschreibung der Schnittstellen (zu Nachbarsystemen und zum betrieblichen Umfeld)

Aus dem Funktionsmodell lassen sich bspw. Informationen gewinnen, welche der einzelnen Anwendungsfälle bzw. Abläufe besonders schwierig sind oder besonders häufig durchgeführt werden. Komponenten, die bei der Durchführung dieser Abläufe eine wichtige Rolle spielen, sind in ihrer inhaltlichen Komplexität hoch einzuschätzen. Außerdem stellt sich die Frage, welche Anwendungsfälle bzw. Abläufe auf Kundenseite regelmäßigen Änderungen unterliegen. Ein mögliches Beispiel für einen solchen Fall ist eine Komponente, die sich mit Abrechnungsvorgängen befasst, die regelmäßig geändert werden. Sie ist besonders kritisch für den Erfolg einer Software, da die ständigen Änderungen besonders hohe Anforderungen an die Struktur, die Schnittstellen und die Funktionen der Komponente richten.

### 3.2.2 Phase 1: Identifikation kritischer Komponenten

Für die Identifikation kritischer Komponenten in Phase 1 wird auf bestehende Architekturbewertungsverfahren zurückgegriffen, die auf der sog. Szenariotechnik basieren. Zur Bewertung von Softwarearchitekturen ist eine Vielzahl von Bewertungsverfahren mit unterschiedlichen Eigenschaften entwickelt worden. Die Methode „Reifegradüberwachung von Software“ schreibt kein konkretes Bewertungsverfahren vor; bei der Entscheidung für ein Verfahren muss jedoch darauf geachtet werden, dass es im Rahmen seiner Durchführung die Gewichtung und Priorisierung von Szenarios unterstützt. Zudem ist es notwendig, dass die Szenarios einer Softwarearchitektur zugeordnet werden können.

Der Vorteil der Integration eines bestehenden Architekturbewertungsverfahrens besteht darin, dass die Ergebnisse in Bezug auf die Szenarien, in die Methode „Reifegradüberwachung von Software“ einfließen, wodurch Zeit und Ressourcen eingespart werden können. Wird eine Architekturbewertung nicht explizit durchgeführt, können die Verfahren bis zur Priorisierung der Szenarios dennoch angewendet und dann frühzeitig abgebrochen werden.

Zu den bekannten Verfahren, die sowohl wissenschaftlich evaluiert als auch in der Praxis angewendet wurden, zählen bspw. SAAM (Software Architecture Analysis Method) und ATAM (Architecture Analysis Tradeoff Method). In viele andere Bewertungsverfahren sind entweder Teilaspekte von SAAM und ATAM eingeflossen, oder es handelt sich dabei um Spezialisierungen [EiHM2007, 3-5].

SAAM basiert auf Szenarien, die nichtfunktionale Qualitätsattribute eines Systems abbilden, die vom bewerteten System unterstützt werden müssen [DoNi2002, 639]. Diese Szenarien werden auf die Softwarearchitektur übertragen, um festzustellen, welche Komponenten einer Softwarearchitektur durch die Szenarien beeinflusst werden [DoNi2002, 642]. Das Vorgehen von ATAM ist als Weiterentwicklung von SAAM [ClKK2002, 43] detaillierter, da es sich auf mehrere Qualitätsattribute und die Identifikation von Trade-Off und Sensitivitätspunkten stützt. Im Bereich der Übertragung von Szenarien auf die Architektur gehen beide Verfahren analog vor.

Das Ergebnis der Auswertung wird, analog zur Vergleichsmethode der VDA Reifegrad-Absicherung, zur Einschätzung der Kritikalität von Komponenten genutzt (vgl. Kapitel 2.2). In der Reifegrad-Absicherung wird zur Auswahl kritischer Lieferumfänge eine ABC-Analyse durchgeführt [VDA2005, 13]. Im Gegensatz zur klassischen ABC-Analyse werden bei der Beurteilung von Komponenten unter A-Teilen nicht die mit der höchsten Wertschöpfung [Bruh2001, 131], sondern die Teile (Komponenten) mit dem höchsten Risikopotenzial zusammengefasst.

Das Mapping von Szenarien auf die Softwarearchitektur, das im Folgenden als „Komponentenüberdeckung" bezeichnet wird, wird in Abbildung 4 veranschaulicht.

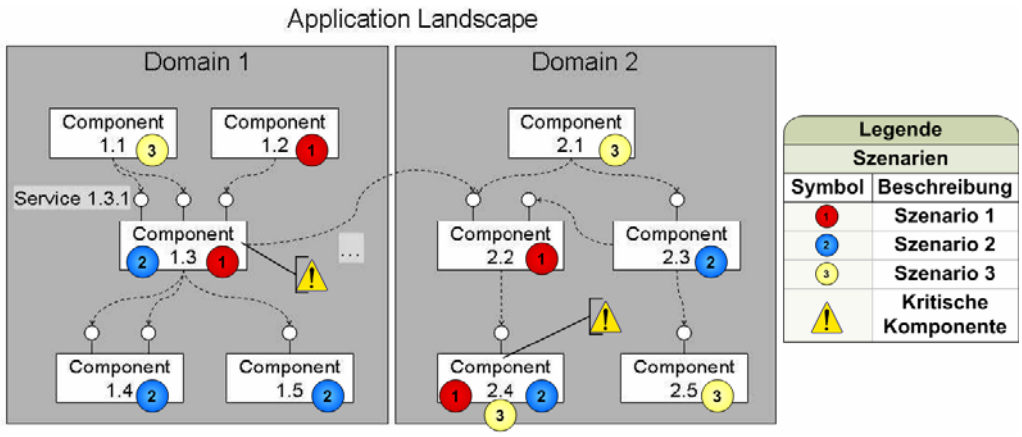



**Abbildung 4: Mapping der Szenarien auf die Softwarearchitektur (vgl. [VoHH2006, 3])**

Kritisch eingeschätzt werden Komponenten, die mehrere Szenarien gleichzeitig oder Szenarien hoher Priorität realisieren. Bei der Priorisierung von Szenarien kann auf die Ergebnisse der Identifikation von kritischen Themengebieten und Use Cases zurückgegriffen werden, die in der Phase „Risikoanalyse" ermittelt wurden.

Als Ergebnis der Komponentenüberdeckung innerhalb der ersten Phase gibt der Nutzer ein Urteil im Spektrum von *niedrig* über *mittel* bis zu *hoch* kritisch ab. Dieses Urteil setzt sich aus den Merkmalen „Menge der Szenarien" und „Bedeutung der Szenarien" zusammen. Ob das Urteil gleichwertig aus beiden Merkmalen (z.B. mithilfe einer Matrix) oder aus dem Gesamteindruck des Nutzers (z.B. des Architekten) hervorgeht, ist nicht explizit festgelegt.

Damit liegt als Ergebnis der „Komponentenüberdeckung" ein Urteil zur Einschätzung der Kritikalität der Komponenten einer zu bewertenden Softwarearchitektur vor.

Um die Aussagekraft des Urteils zur Kritikalität von Komponenten zu verbessern, wird dieses mit einem zweiten Urteil verdichtet, dass sich aus einer Auswahl zusätzlicher Merkmale zusammensetzt. Das zweite Urteil, das aus subjektiven Einschätzungen eines Chefarchitekten resultiert, wird im Folgenden als „Klassifikation durch den Chefarchitekten" bezeichnet. Die Zusammenstellung der folgenden Merkmale wurde sowohl aus der Analyse bestehender Sätze für Risiken bei der Softwareentwicklung [HaFP2005, 4-5] [Wall2001, 19] als auch aus dem in der Referenzmethode verwendeten Kriteriensatz [VDA2005, 30f.] hergeleitet.

- Fachliche Stabilität
    - Erwartete Änderung
    - Unklare Anforderung
- Qualifikation der Mitarbeiter
- Neuigkeitsgrad
    - Neue Fachlichkeit
    - Neue Technik

- Komplexität
    - Inhaltliche Komplexität
    - Abhängigkeiten

Unter dem Merkmal „Fachliche Stabilität“ wird das potenzielle Risiko zukünftiger Änderungen an der Komponente betrachtet. Gründe hierfür können sowohl die unklare oder unbestimmte Formulierung von Anforderungen seitens des Kunden („Unklare Anforderung“) sein, als auch erwartete Änderungen, die sich aus den implementierten fachlichen Anforderungen sowie den unterstützen Prozessen ergeben („Erwartete Änderung“).

Das Merkmal „Qualifikation der Mitarbeiter“ bewertet die Kompetenz des ausgewählten Teams in Bezug auf ein konkretes Entwicklungsprojekt (die Entwicklung der speziellen Komponente).

Unter dem Merkmal „Neuigkeitsgrad“ werden einerseits Anforderungen zusammengefasst, die aus der Verwendung einer neuen Technologie (z.B. eines neuen Patterns) entstehen („Neue Technik“) oder aus einer neuen Fachlichkeit in dem das Entwicklungsteam bisher noch kein Erfahrungswissen vorzuweisen hat („Neue Fachlichkeit“).

Das Merkmal „Komplexität“ ist in die Unterpunkte „Inhaltliche Komplexität“ und „Abhängigkeiten“ unterteilt, da auf Komponentenebene nicht nur der inhaltliche Anspruch der Entwicklung einer einzelnen Komponente berücksichtigt werden muss. Während auf der Beurteilungsebene einer gesamten Software, nur ein Beurteilungsobjekt existiert, ist es bei der Bewertung von Komponenten eine unbestimmte Anzahl miteinander in Beziehung stehender Objekte, deren Verknüpfungen die kritischen Bereiche einer Architektur beeinflussen. Diese Art der Komplexität wird mit dem Unterpunkt „Abhängigkeiten“ in die Risikobewertung einbezogen.

Die Möglichkeit, neben den vier festgelegten Soll-Merkmalen noch ein bis zwei weitere Kann-Merkmale für das Risiko einzelner Komponenten zu definieren, wird offen gelassen (vgl. Prinzip der Kontrollierbarkeit). Die ausgewählten 4 + X Merkmale werden zu einem Urteil des Chefarchitekten zusammengefasst. Dazu wird zunächst jedes Merkmal mit einem subjektiven Urteil von niedrig, mittel oder hoch bewertet und mit den Werten 1 (niedrig), 2 (mittel) und 3 (hoch) einer Rationalskala zugeordnet. Die Verrechnung erfolgt nach Abgabe aller Urteile, anhand folgender Formel:

$$UC_k = \frac{\sum_{m=1}^{M} U_m}{M}$$

$UC_k$ = Urteil des Chefarchitekten zur einer Komponente k

$U_m$ = Urteil zum Merkmal m

M = Anzahl aller Merkmale

$\sum_{m=1}^{M}$ = Summe aller M Urteile zu einer Komponente

Damit errechnet sich ein Urteil eines Chefarchitekten zu einer Komponente k, indem die Summe aller Urteile gebildet und durch die Anzahl aller Merkmale (M) geteilt wird. Abbildung 5 zeigt das Ergebnis eines Beispiels mit vier Soll-Merkmalen und einem Kann-Merkmal „Kann1“. Der ermittelt Wert 2,4 wird abschließend wieder auf die Nominalskala mit folgenden Intervallen übertragen.

Liegt der Wert zwischen 1 (>= 1) und 1,5 (<= 1,5) wird das Urteil als „niedrig“, zwischen 1,5 (> 1,5) und 2,5 (<= 2,5) als „mittel“ und zwischen 2,5 (> 2,5) und 3 (<= 3) als „hoch“ eingestuft. Damit würde das errechnete Ergebnis 2,4 als Urteil auf der Nominalskala „mittel“ ergeben.

| Name des Merkmals m | Urteil zum Merkmal ($U_m$) |
|---|---|
| Fachliche Stabilität | 3 |
| Qualifikation der Mitarbeiter | 1 |
| Neuigkeitsgrad | 3 |
| Komplexität | 2 |
| Kann1 | 3 |
| Σ | 12 |

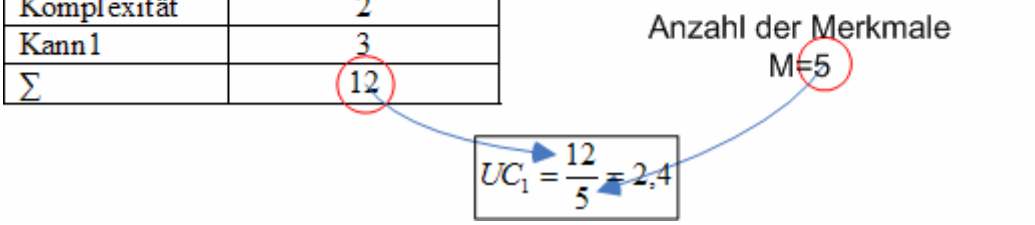



**Abbildung 5: Berechnung des Urteils eines Chefarchitekten**

Auch bei der „Klassifikation durch den Chefarchitekten" wird auf die Ergebnisse der „Risikoanalyse" (Phase 0) zurückgegriffen. Die Erkenntnisse zu kritischen Themengebieten können bspw. bei der Beurteilung des Merkmals Komplexität genutzt werden. Ein zweiter Punkt, der von Interesse ist, stellt das Merkmal „Fachliche Stabilität" dar. Dieses Merkmal ist insbesondere dann als hoch einzustufen, wenn Anwendungsfälle aufgegriffen werden, zu denen aus der Phase 0 der Hinweis existiert, dass sie regelmäßigen Änderungen unterliegen. Auch das Merkmal „Neuigkeitsgrad" kann im Falle einer GUI-Komponente, durch einen Hinweis aus der Vorphase entsprechend klassifiziert werden

Insgesamt liefert Phase 1 „Identifikation kritischer Komponenten" für jede Komponente der Softwarearchitektur zwei Urteile auf der Nominalskala. Die Phase endet mit der Verdichtung der beiden Urteile, die eine Entscheidung bei der Auswahl kritischer Komponenten einer Architektur erlaubt. Dazu werden beide Urteile auf die Achsen einer 3x3 Matrix übertragen. In der entstehenden Matrix entfallen jeweils 3 Felder auf die Urteile niedrig, mittel und hoch. Abbildung 6 zeigt zusammenfassend den gesamten Ablauf der Phase.

Komponenten, die mit dem Urteil „hoch" versehen sind, gelten als kritische Komponenten und werden im weiteren Projektverlauf einer konsequenten Überprüfung durch die Phase 2 der Methode „Reifegradüberwachung von Software" unterzogen. Komponenten, die mit dem Urteil mittel oder niedrig bewertet wurden, werden als nicht kritische Komponenten keiner näheren Überprüfung unterzogen.

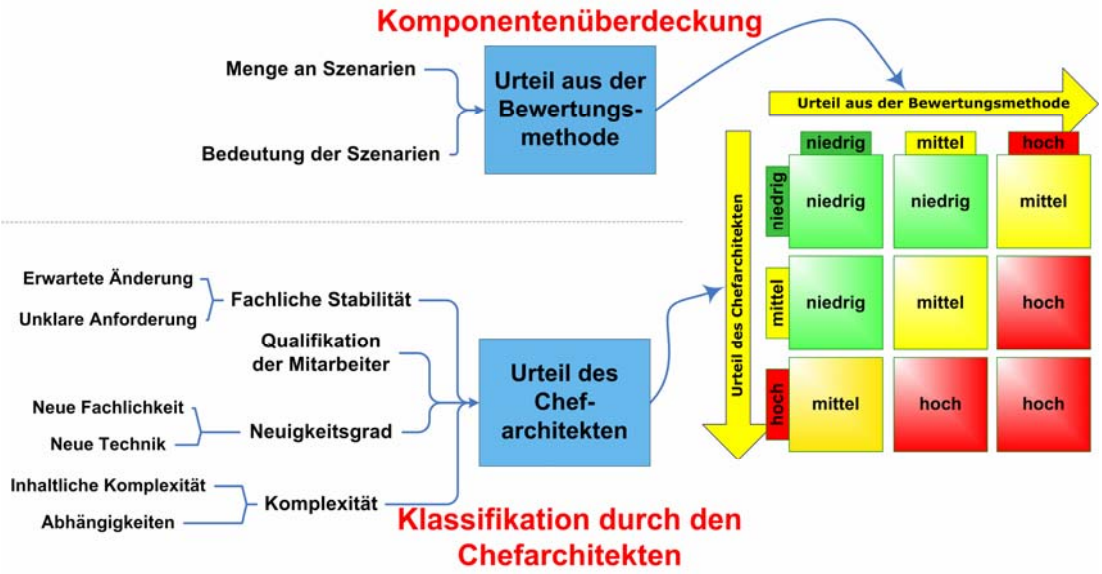



**Abbildung 6: Phase 1 der Methode**

### 3.2.3 Phase 2: Reifegradcontrolling

Bei der Methode „Reifegradüberwachung von Software“ wurden auch in Phase 2 „Reifegrad-controlling“ Prinzipien der Referenzmethode übertragen. Dazu zählt die Abbildung der Ergebnisse auf eine Ampel, die den im Kapitel 3.1 aufgestellten Prinzipien „Einfachheit“ und „Selbsterklärende Begrifflichkeiten“ entspricht. Die Ampel, als visuelles Hilfsmittel zur Abbildung von Aussagen, erleichtert es, gewonnene Ergebnisse auf einen Blick zu erfassen. Außerdem abstrahiert das Ampelurteil von den deutlich komplexeren Kennzahlen, die dem Urteil zugrunde liegen und deren Interpretation für Außenstehende nicht einfach ist. Die enge Einbindung des Kunden kann ebenso als Prinzip aufgegriffen werden, da eine unzureichende Integration des Anwenders in ein Entwicklungsprojekt eine der Hauptursachen für ein mögliches Scheitern darstellt [Ewus1997, 75]. In der entwickelten Methode wird der Bedeutung des Kunden in zwei Punkten Rechnung getragen. Einerseits wird darauf geachtet, die vom Kunden gestellten Anforderungen zu erfüllen, andererseits wird das Urteil des Kunden direkt als Kriterium in die Bewertung mit einbezogen.

Die Integration der Reifemessung in den Projektverlauf wird ebenfalls aufgegriffen. Wichtig ist, dass die Methode einem Planungsprozess unterliegt, in dem sich die Projektbeteiligten über geeignete Zeitintervalle abstimmen. Dazu können in der Methode „Reifegradüberwachung von Software“ bspw. Projektmeilensteine als vorgegebene Zeitpunkte zur Berechnung eines Reifegrades festgelegt werden. Im Gegensatz zur Vergleichsmethodik beschränkt sich die Methode jedoch nicht ausschließlich auf die Meilensteine als Zeitpunkte der Messung. Mithilfe automatischer Testtechniken lassen sich Kennzahlen ohne den Einsatz zusätzlicher Ressourcen generieren und somit, abhängig von ihrer Art, in unterschiedlichen Abständen ermitteln. Kennzahlen, deren Ermittlung nicht automatisch erfolgen kann, werden zum jeweiligen Zeitpunkt der Projektmeilensteine aktualisiert. Kennzahlen, die automatisch ermittelt werden können, werden in kürzeren Abständen erneuert (z.B. über Nightly Builds).

Auch die Idee, Indikatoren zu verwenden, denen Messkriterien zugeordnet sind, wird in der neuen Methode aufgegriffen. Deren Zuordnung erfolgt über ein festgelegtes Reifemodell. Die Ergebnisse werden jedoch nicht verwendet, um einen mehrstufigen Reifegrad zu ermitteln, wie es sowohl in der Vergleichsmethodik als auch in Verfahren wie CMMI und SPICE [MeHS1998, 94-96, 114-116] [Oreg2002, 133-136, 177f.] üblich ist. Gründe hierfür sind vor allem die Besonderheiten der Softwareentwicklung gegenüber anderen Industriezweigen finden.

Im Gegensatz zur Softwareindustrie sind in anderen Industriezweigen die Erfolgsquoten von Entwicklungsprojekten deutlich höher und die Ursachen auftretender Fehlern können aufgrund der besseren Messbarkeit einfacher ermittelt werden. Derzeitige Softwareentwicklungsprojekte weisen deutlich niedrigere Erfolgsquoten und eine erschwerte Messbarkeit des Bewertungsgegenstandes auf. Da die Komponenten einer Software häufigeren Änderungen unterliegen, kann nicht gewährleistet werden, dass bereits einmal als positiv bewertete Eigenschaften weiterhin sichergestellt sind. Trotz der Verwendung von spezifizierten Schnittstellen und der Kapselung von Aufgaben und Funktionen in Klassen und Komponenten treten immer noch starke Wechselwirkungen zwischen logisch miteinander verknüpften Elementen einer Software auf, die dazu führen, dass Änderungen an einem Element andere Elemente beeinflussen.

## Reifemodell der Methode

Das erstellte Reifemodell beschränkt sich auf fünf Merkmale, da dies dem Prinzip der Einfachheit entspricht. Außerdem kann damit sichergestellt werden, dass die generierten Aussagen auf einen Blick erfassbar bleiben. Die Grenze der Wahrnehmung eines Menschen liegt nach Miller bei maximal sieben gleichzeitig zu verarbeitenden Kanälen [Mill1956, 81ff.].

Das Reifemodell orientiert sich an den Qualitätsmerkmalen der ISO Norm 9126 [ISO2001, 7]. Es umfasst die folgenden Merkmale, anhand derer die Kennzahlen in Phase 2 der Methode gegliedert und verdichtet werden:

- Funktionalität
- Zuverlässigkeit
- Benutzbarkeit
- Wartbarkeit
- Entwicklungsstand

Bei der Auswahl der Merkmale war ausschlaggebend, in wie weit Aussagen zu zentralen Einflussfaktoren eines Softwareentwicklungsprozesses im Sinne der Definition von Softwareproduktreife (vgl. Kapitel 2.1) getroffen werden können. Als zentrale Einflussfaktoren wurden die Anforderungen an die Software, die Stakeholder, die Struktur der Software, die Planung und das Auftreten von Fehlern identifiziert.

In Abbildung 7 werden die Reifegradmerkmale und deren Teilmerkmale dargestellt. Die grün hinterlegten Merkmale wurden aus der ISO 9126 Norm übernommen. Bei den hellgrün hinterlegten Merkmalen handelt es sich um Erweiterungen im Sinne der Reifedefinition. Weiß hinterlegte Merkmale wurden aus der ISO Norm nicht übernommen. Auffällig ist, dass das Merkmal „Reife" in der ISO 9126 bereits existiert. Es entspricht jedoch nicht dem hier definierten Reifeverständnis.

Mit dem Merkmal „Funktionalität" können Aussagen über die Angemessenheit, Richtigkeit und Vollständigkeit einer Komponente hinsichtlich der spezifizierten Anforderungen getroffen werden. Das Merkmal „Zuverlässigkeit" hingegen lässt Aussagen über die Fehlertoleranz, Stabilität und Fehlerdichte einer Komponente zu. Für Aussagen über die Verständlichkeit, Bedienbarkeit und Aufrufbarkeit einer Komponente aus Sicht der Kunden und Entwickler wird das Merkmal „Benutzbarkeit" herangezogen.

Bei der Reifegradüberprüfung liegt der Fokus insbesondere darauf, Strukturrichtlinien zu wahren. Das Merkmal „Wartbarkeit" beschreibt die Fähigkeit einer Softwarekomponente, Strukturvorgaben zu erfüllen und somit analysierbar, modifizierbar und austauschbar zu bleiben.

Ein weiterer wesentlicher Aspekt in Bezug auf die Definition von Softwareproduktreife ist das Fortschreiten der Entwicklung hinsichtlich der Projektvorgaben. Das Merkmal „Entwicklungsstand" betrachtet die Entwicklung einer Software zum Zeitpunkt X, in Relation zum erwarteten Fortschritt.

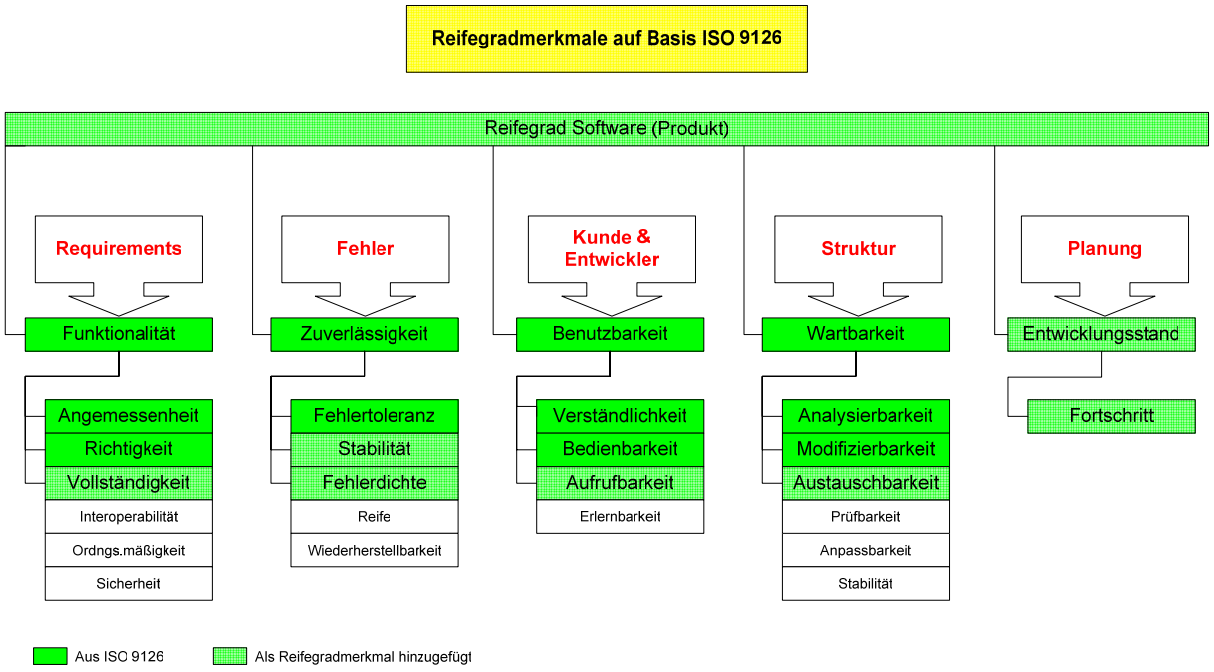



**Abbildung 7: Reifemodell auf Basis der ISO 9126 Norm**

## Auswahl einer Bewertungsskala

In der hier vorgestellten Methode werden zwei Typen von Kennzahlen unterschieden, die in die Berechnung des Reifegrades einer Komponente eingehen. Dies sind einerseits Metriken, die automatisch ermittelt werden können und bspw. Aussagen über die Struktur des Programms oder die auftretenden Fehlerquoten erlauben (technische Kennzahlen, TK). Andererseits existieren subjektive Kennzahlen (SK), die auf Angaben von Kunden, Testern, Softwarearchitekten und -entwicklern beruhen.

Bei den in der Methode verwendeten Kennzahlen lassen sich unterschiedliche Typen von Skalen identifizieren. Die Ordinalskala wird für subjektive Kennzahlen verwendet, in denen ein Stakeholder eine direkte Angabe zur Software macht. Die technischen Kennzahlen hingegen liegen entweder auf der Rationalskala (z.B. Testüberdeckung) oder der Absolutskala (z.B. Lines Of Code) vor.

Um die Messwerte für die Reifeberechnung in Phase 2 der Methode anzugleichen, werden alle ermittelten Kennzahlen mittels Intervallen auf die Ordinalskala übertragen. Die Aussagekraft der Ordinalskala, mit einer ansteigenden Ordnung zwischen den Kategorien, eignet sich insbesondere für Aussagen, die bei einer Bewertung von Software getroffen werden. Deshalb sieht Zuse die Ordinalskala sogar als Basis der Softwarebewertung [Zuse1997, 142]

Für die technischen Kennzahlen werden Intervalle definiert, die eine Abbildung auf die Ordinalskala (hier Notenskala) ermöglicht. Bei den subjektiven Kennzahlen ist eine Abbildung nicht notwendig, da sie bereits in der richtigen Skalenform vorliegen.

Da die einzelnen Kennzahlen unterschiedlich detaillierte Aussagen liefern, wird es nicht nur eine, sondern drei mögliche Formen der Abbildung auf die Notenskala geben, die eine unterschiedlich differenzierte Aussage erlauben (vgl. Tabelle 2). Das entspricht dem Prinzip der Kontrollierbarkeit, da der Nutzer der Methode „Reifegradüberwachung von Software“ in der Lage ist, die aus seiner Sicht am besten geeignete Form der Abbildung zu wählen. Es sollte jedoch darauf geachtet werden, die Abbildungsvorschriften „Grenzwert“ und „Grobe Bewertung“ nur dann einzusetzen, wenn eine detailliertere Aussage nicht möglich ist.

| Abbildungsvorschrift | Einteilung mittels | Ausprägungen | Zugeordnete Noten | Detailgrad der Aussage |
|---|---|---|---|---|
| *Grenzwert* | Grenzwert | Gut | 1 | Grob |
| | | Schlecht | 6 | |
| *Grobe Bewertung* | Intervalle | Gut | 1 | Mittel |
| | | Mittel | 3,5 | |
| | | Schlecht | 6 | |
| *Detaillierte Bewertung* | Intervalle | Sehr Gut, Gut… Ungenügend | 1- 6 | Hoch |

**Tabelle 2: Abbildungsvorschriften auf die Notenskala**

Mit diesem Vorgehen ist sichergestellt, dass alle Eingangswerte des Verfahrens auf der gleichen Skala vorliegen. Die weitere Verdichtung erfolgt ab diesem Punkt der Methode auf einer Rationalskala mit Werten von 1 bis 6. Die Bedeutung der einzelnen Kennzahlen kann über eine Gewichtung spezifisch festgelegt werden. Hier obliegt die Kontrolle der Methode „Reifegradüberwachung von Software" wieder dem Anwender, der sie seinen Anforderungen entsprechend anpassen kann.

### Verdichtung und Abbildung der Kennzahlen

Die Berechnungsformel zur Ermittlung von Werten der einzelnen Merkmale des Reifemodells, die zur Messung der Reife einer Komponente herangezogen werden, lautet:

$$M_j = \frac{\sum_{i=1}^{N} K_i * g_i}{\sum_{i=1}^{N} g_i}$$

$M_j$ = Merkmal j eines Reifeurteils einer Komponente

$K_i$ = Kennziffer i

$g_i$ = Gewichtung der Kennziffer i

$\sum_{i=1}^{N}$ = Summe über alle Kennzahlen eines Merkmals einer Komponente

Jedes Merkmal einer Komponente errechnet sich aus der Summe aller gewichteten Kennzahlen, die durch die Summe der Kennziffergewichte geteilt wird. Ein Beispiel zur Berechnung eines Merkmals auf Basis von drei Kennzahlen und ihrer Gewichte wird in Abbildung 8 dargestellt.

Nachdem dem Anwender der Methode die Ergebnisse der einzelnen Merkmale vorliegen, werden diese zu einer Reifegradkennziffer einer Komponente verdichtet. Für die Verdichtung der einzelnen Merkmale wird erneut eine Gewichtung definiert, die den Stellenwert der einzelnen Merkmale des Reifemodells, bei der Berechnung eines Reifegrades festlegt. Die Skala für die Gewichte ist wie auch im vorherigen Schritt eine Notenskala mit Werten von 1 bis 6.

| Kennziffer-bezeichnung | Wert der Kennziffer ($K_i$) | Gewichtung ($g_i$) | $K_i$*$g_i$ |
|---|---|---|---|
| $K_1$ | 1 | 6 | 6 |
| $K_2$ | 3 | 2 | 6 |
| $K_3$ | 2 | 4 | 8 |
| $\sum$ | ---------- | 12 | 20 |

Für die weitere Berechnung wird die Summe der Kennziffern nicht benötigt

$$M_1 = \frac{20}{12} = 1{,}66667$$

**Abbildung 8: Berechnung eines Merkmals**

Die Formel zur Berechnung des Reifegradurteils einer Komponente lautet:

$$R = \frac{\sum_{j=1}^{N} M_j * h_j}{\sum_{j=1}^{N} h_j}$$

R = Reifegrad einer Komponente

$M_j$ = Merkmal j

$h_j$ = Gewichtung des Merkmals j

$\sum_{j=1}^{N}$ = Summe über alle Merkmale eines Reifegrades einer Komponente

Der Reifegrad einer Komponente errechnet sich aus der Summe aller gewichteten Merkmale, die durch die Summe der Merkmalsgewichte geteilt wird. Ein Beispiel zur Berechnung eines Reifegrads einer Komponente, auf Basis von drei Merkmalen und ihrer Gewichte, wird in der folgenden Abbildung dargestellt (vgl. Abbildung 9).

| Merkmals-bezeichnung | Wert des Merkmals ($M_j$) | Gewichtung ($h_j$) | $M_j$*$h_j$ |
|---|---|---|---|
| $M_1$ | 1,66667 | 2 | 3,33334 |
| $M_2$ | 2,35 | 4 | 9,4 |
| $M_3$ | 3,7 | 5 | 18,5 |
| $\sum$ | ---------- | 11 | 31,23334 |

Für die weitere Berechnung wird die Summe der Merkmale nicht benötigt

$$R = \frac{31{,}23334}{11} = 2{,}83939$$

**Abbildung 9: Berechnung eines Reifegrads**

In Abbildung 10 werden alle in den beiden Berechnungsschritten relevanten Parameter dargestellt. Die einem Merkmal zugehörigen Kennzahlen ($K_i$) werden mit ihrem/-en Gewicht/-ungen ($g_i$) multipliziert und dann durch die Summe der Gewichte geteilt, um zu einem Ergebnis für ein Merkmal zu kommen. Diese Berechnung erfolgt für jedes einzelne Merkmal. Die Ergebnisse aller Merkmale ($M_j$)

werden wieder mit ihren Gewichten ($h_i$) multipliziert und durch die Summe aller Gewichte geteilt, um zum Reifeergebnis für die jeweilige kritische Komponente zu kommen.

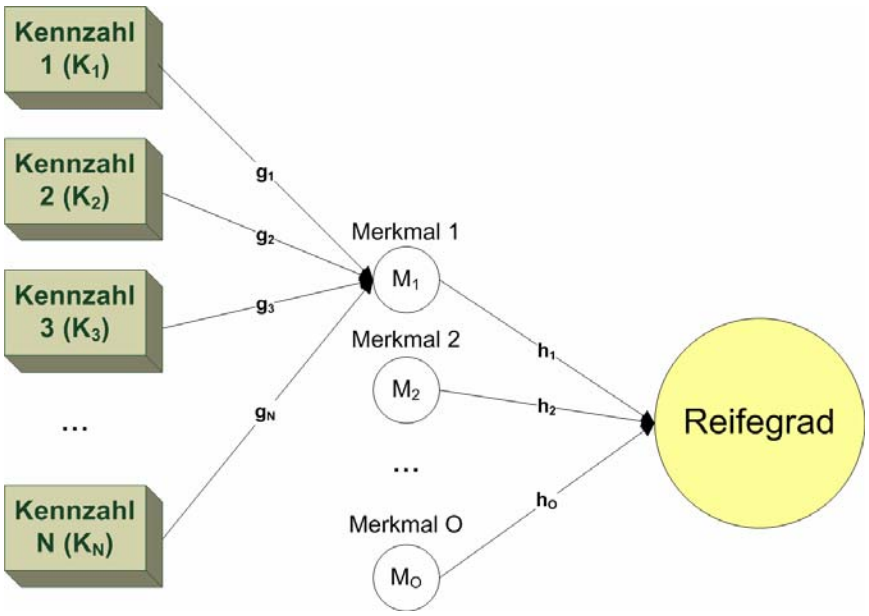



**Abbildung 10: Schritt 2 und 3 der 2.Phase der Methode**

Abschließend gilt es, die ermittelten Ergebnisse auszuwerten. Dazu werden die Ergebnisse auf Ampeln übertragen. Um dies zu ermöglichen, müssen erneut Intervalle verwendet werden, die eine Zuordnung der Werte von der Rationalskala auf die Ampel erlauben (vgl. Tabelle 3).

| Ampelphase | Aussage | Zugehöriges Intervall |
|---|---|---|
| Rot | Negatives Ergebnis | [4,5 – 6] |
| Gelb | Mittleres Ergebnis | [2,5 – 4,5) |
| Grün | Positives Ergebnis | [1 - 2,5) |

**Tabelle 3: Definition der Ampelphasen**

Mit den Ampelphasen kann nun eine Aussage zu den generierten Urteilen der Merkmale und des Reifegrades einer Komponente getroffen werden. Eine negative Aussage wird im Intervall zwischen 6 und 4,5 getroffen (inklusive 4,5 und 6), eine mittelmäßige im Intervall zwischen 4,5 und 2,5 (exklusive 4,5 und inklusive 2,5) und eine positive zwischen 2,5 und 1 (exklusive 2,5 und inklusive 1).

Da bei der Zuordnung der Ergebnisse zu den Ampelphasen bei vorherigem Runden insgesamt zwei Nachkommastellen benötigt werden, reichen fünf Nachkommastellen aus, um keine Ungenauigkeiten durch Rundungen entstehen zu lassen. Es werden bewusst mehr als zwei Nachkommastellen genutzt, da nach der Berechnung der Merkmale mit den bereits gerundeten Ergebnissen weitergerechnet wird.

Die Abbildung der Ergebnisse auf eine Ampel dient dazu, das Ergebnis der Methode bezüglich der Reife einer Komponente besser interpretieren zu können. Im Falle eines negativen (rote Ampel) oder mittelmäßigen (gelbe Ampel) Urteils, stellt sich sofort die Frage nach den Ursachen für dieses Ergebnis. Ampeln werden ebenso verwendet, um die Zwischenurteile für die einzelnen Merkmale

des Reifegradverständnisses abzubilden. So können im entstehenden Ampelbaum (vgl. Abbildung 11) die Merkmale identifiziert werden, die verbessert werden müssen.

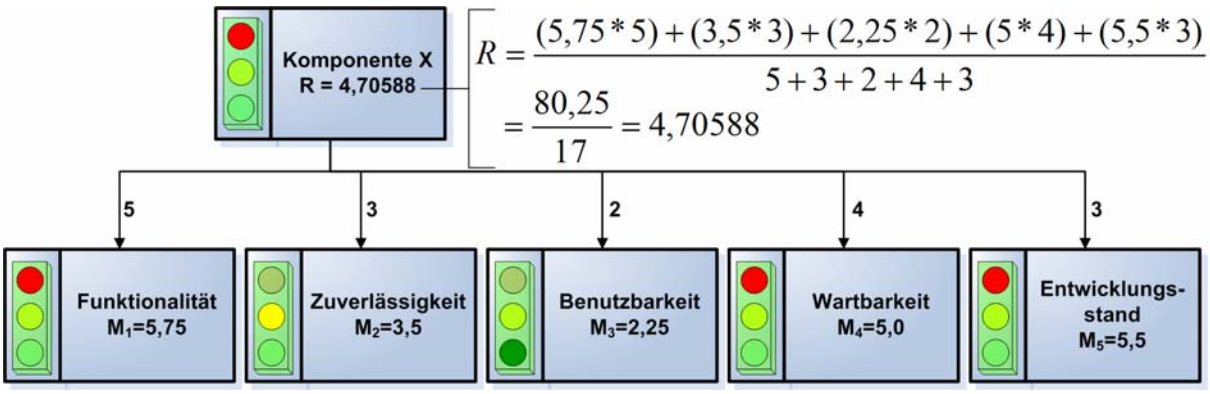



**Abbildung 11: Ampelbaum einer Komponente**

# 4 Evaluation der Methode

Zur Evaluierung der Methode „Reifegradüberwachung von Software" wurde ein Prototyp implementiert, mit dessen Hilfe die Reifegradkennziffern kritischer Komponenten auf Basis eines Kennzahlensatzes ermittelt werden können.

Da bei der Ermittlung einer Reifegradkennziffer neben den subjektiven Urteilen (subjektiven Kennzahlen) auch Metriken (technische Kennzahlen) einbezogen werden, wurde bei der Implementierung auf ein bestehendes System zur Ermittlung von Qualitätsurteilen auf Basis von Metriken zurückgegriffen. Bei dem System handelt es sich um das Software Cockpit [BeRi2007][BSRR2007] der Firma sd&m. Durch die Integration des Prototyps mit dem SoftwareCockpit konnte auf Ergebnisse in Form von Anomalien[1] und Metriken der Werkzeuge FindBugs, CheckStyle, JavaNCSS, JUnit, EMMA und CPD zurückgegriffen werden.

## 4.1 Aufbau von Testkonfigurationen

Zum Testen des Prototyps und unterschiedlicher Konfigurationen wurde ein einfacher Kennzahlensatz zusammengestellt, der sich sowohl aus subjektiven als auch aus technischen Kennzahlen zusammensetzt. Die Auswahl eines umfangreicheren Kennzahlensatzes, der auf das vorgestellte Reifemodell zugeschnitten ist, ist Gegenstand der aktuellen Forschungsaktivitäten.

Die ausgewählten Kennzahlen wurden den Merkmalen des Reifemodells (vgl. Kapitel 3.2) zugeordnet. Die Anzahl der technischen Kennzahlen, die im Rahmen der Testkonfigurationen verwendet wurden, ist noch sehr gering.

Der Aufbau des im Prototyp getesteten Reifemodells gestaltet sich folgendermaßen:

- Reife
    - Funktionalität
        - Implementierte Funktionalität (SK)

[1] Unter einer Anomalie wird im Software Cockpit ein Verstoß gegen eine Vorschrift in Form eines Grenzwertes oder eines Intervalls zu einer Metrik verstanden.

        - Übereinstimmung mit der Erwartungshaltung (SK)
    - Zuverlässigkeit
        - Testfälle mit Fehlern (fehlgeschlagene Testfälle/Anzahl Testfälle) (TK)
    - Benutzbarkeit
        - Bewertung der Performance (SK)
        - Externe Service-Schnittstellen (Aufruf) (SK)
        - Externe Service-Schnittstellen (Dokumentation/Bedingungen) (SK)
    - Wartbarkeit
        - Anzahl Methoden (TK)
        - Non Commenting Source Statements (NCSS) (TK)
    - Entwicklungsstand
        - Subjektiver Fertigstellungsgrad (SK)

Neben der Zuordnung von Kennzahlen zu den Merkmalen des Reifemodells wurde eine Konfigurationsdatei definiert, in der die Gewichte der Merkmale und Kennzahlen sowie die Intervalle für die technischen Kennzahlen enthalten sind.

Zur Gewinnung von Reifegradergebnissen wurden zwei Testkonfigurationen erstellt, die auf Basis empirischer Referenzwerte zur Herleitung von Schwellenwerten für die technischen Kennzahlen aufgebaut wurden. Beim Aufbau der Testkonfigurationen wurden Referenzwerte von Lakos [Lako1996, 481] und Meyer [Meye1995, 51] herangezogen.

Die Verwendung von Konfigurationsdateien bietet den Vorteil, die Referenzwerte jederzeit anpassen und somit einem Feintuning unterziehen zu können. In den gewählten Testkonfigurationen sind alle Merkmale und Kennzahlen des Reifemodells gleich gewichtet.

## 4.2 Betrachtung und Analyse der generierten Ergebnisse

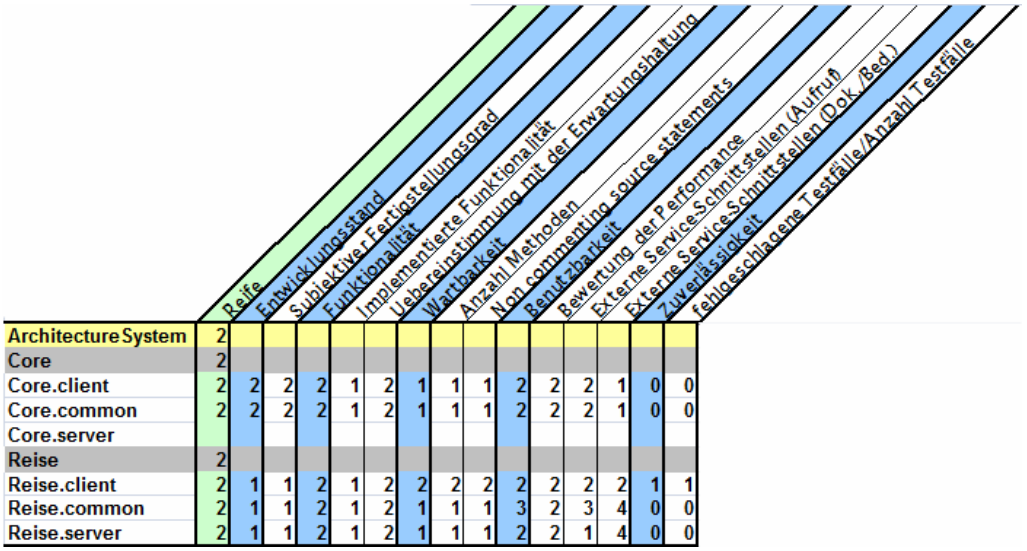
Abbildung 12 und Abbildung 13 zeigen die generierten Ergebnisse der beiden Testkonfigurationen in Form einer Matrix. Die Abbildung der Ergebnisse auf das Ampelsystem wurde im Rahmen der ersten Prototypentwicklung noch nicht realisiert. In der ersten Spalte sind die Elemente der Software aufgelistet (grau unterlegt = Domänen, gelb unterlegt = die gesamte Software). Die erste Zeile stellt das Reifemodell und seine Kennzahlen dar (blau unterlegt = Merkmale, grün unterlegt = das Reifeurteil).

In beiden Konfigurationen wurde die Komponente „Core.server" nach Durchführung von Phase 0 und Phase 1 als unkritisch deklariert und nicht in die Berechnung einbezogen.

Auf Ebene der kompletten Software (ArchitectureSystem) und der einzelnen Domänen findet sich jeweils das schlechteste Urteil aller enthaltenen Reifeurteile. Die Einträge in den blauen Merkmalsspalten errechnen sich aus dem Ergebnis der Summe aller zugeordneten (gewichteten) Kennzahlen, die durch die Anzahl der enthaltenen Kennzahlengewichte geteilt wird.

Bei der Berechnung des Merkmals „Zuverlässigkeit" bestand die Schwierigkeit darin, dass lediglich für die Komponente „Reise.client" Testfälle deklariert und durchgeführt wurden. Für die anderen Komponenten wurde dieses Merkmal automatisch aus der Berechnung ausgelassen (vgl. „0" Einträge in den beiden Tabellen). Es stellt sich die Frage, ob ein Merkmal zukünftig aus der Berechnung ausgeschlossen werden sollte, falls keine aussagekräftigen Ergebnisse vorliegen. Eine Alterna-

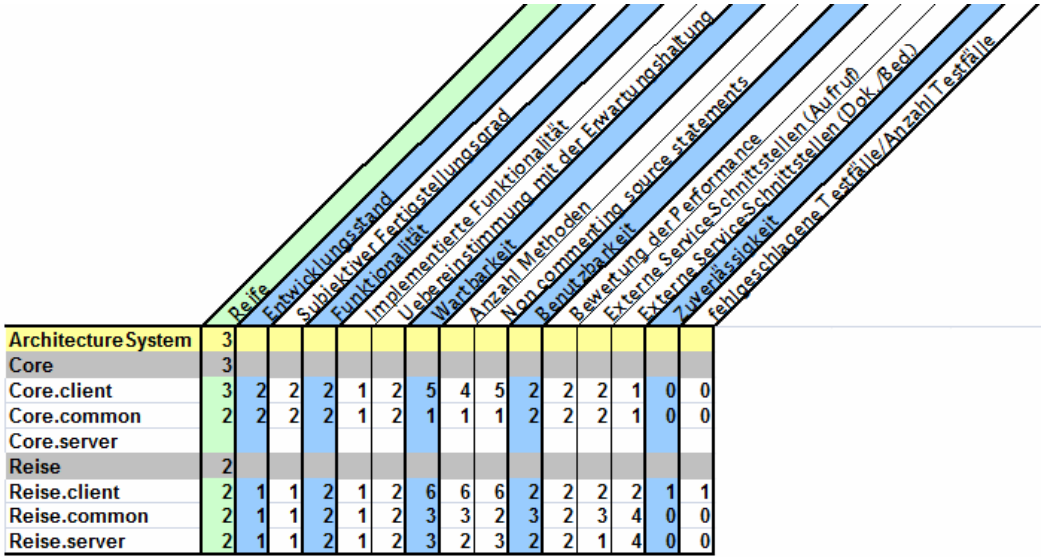
tive wäre, das Merkmal mit dem Urteil „6“ zu bewerten, um auf den Missstand des Fehlens erforderlicher Daten hinzuweisen.



| | Reife | Entwicklungsstand | Subjektiver Fertigstellungsgrad | Funktionalität | Implementierte Funktionalität | Uebereinstimmung mit der Erwartungshaltung | Wartbarkeit | Anzahl Methoden | Non commenting source statements | Benutzbarkeit | Bewertung der Performance | Externe Service-Schnittstellen (Aufruf) | Externe Service-Schnittstellen (Dok./Bed.) | Zuverlässigkeit | fehlgeschlagene Testfälle/Anzahl Testfälle |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Architecture System | 2 | | | | | | | | | | | | | | |
| Core | 2 | | | | | | | | | | | | | | |
| Core.client | 2 | 2 | 2 | 2 | 1 | 2 | 1 | 1 | 1 | 2 | 2 | 2 | 1 | 0 | 0 |
| Core.common | 2 | 2 | 2 | 2 | 1 | 2 | 1 | 1 | 1 | 2 | 2 | 2 | 1 | 0 | 0 |
| Core.server | | | | | | | | | | | | | | | |
| Reise | 2 | | | | | | | | | | | | | | |
| Reise.client | 2 | 1 | 1 | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 |
| Reise.common | 2 | 1 | 1 | 2 | 1 | 2 | 1 | 1 | 1 | 3 | 2 | 3 | 4 | 0 | 0 |
| Reise.server | 2 | 1 | 1 | 2 | 1 | 2 | 1 | 1 | 1 | 2 | 2 | 1 | 4 | 0 | 0 |

**Abbildung 12: Ergebnisse der Testkonfiguration 1 (Lakos)**

Die subjektiven Kennzahlen der Merkmale „Entwicklungsstand“ und „Funktionalität“ sind in beiden Konfigurationen durchweg sehr gut bewertet, was sich mit dem fortgeschrittenen Entwicklungsstand der Anwendung erklären lässt. Auffällig bei der Verteilung der Urteile war die Ähnlichkeit der Urteile zu einzelnen Merkmalen von verschiedenen Komponenten innerhalb derselben Domäne. Dieses Verhalten lässt sich damit erklären, dass in einer kleineren Anwendung weniger unterschiedliche Funktionalität vorliegt, als es bei größeren der Fall ist. Dieser Effekt würde bei einer umfangreicheren Anwendung, bei der gezielt wenige kritische Komponenten aus verschiedenen Domänen ausgewählt werden, vermutlich nicht auftreten.



| | Reife | Entwicklungsstand | Subjektiver Fertigstellungsgrad | Funktionalität | Implementierte Funktionalität | Uebereinstimmung mit der Erwartungshaltung | Wartbarkeit | Anzahl Methoden | Non commenting source statements | Benutzbarkeit | Bewertung der Performance | Externe Service-Schnittstellen (Aufruf) | Externe Service-Schnittstellen (Dok./Bed.) | Zuverlässigkeit | fehlgeschlagene Testfälle/Anzahl Testfälle |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Architecture System | 3 | | | | | | | | | | | | | | |
| Core | 3 | | | | | | | | | | | | | | |
| Core.client | 3 | 2 | 2 | 2 | 1 | 2 | 5 | 4 | 5 | 2 | 2 | 2 | 1 | 0 | 0 |
| Core.common | 2 | 2 | 2 | 2 | 1 | 2 | 1 | 1 | 1 | 2 | 2 | 2 | 1 | 0 | 0 |
| Core.server | | | | | | | | | | | | | | | |
| Reise | 2 | | | | | | | | | | | | | | |
| Reise.client | 2 | 1 | 1 | 2 | 1 | 2 | 6 | 6 | 6 | 2 | 2 | 2 | 2 | 1 | 1 |
| Reise.common | 2 | 1 | 1 | 2 | 1 | 2 | 3 | 3 | 2 | 3 | 2 | 3 | 4 | 0 | 0 |
| Reise.server | 2 | 1 | 1 | 2 | 1 | 2 | 3 | 2 | 3 | 2 | 2 | 1 | 4 | 0 | 0 |

**Abbildung 13: Ergebnisse der Testkonfiguration 2 (Meyer)**

Eine weitere Auffälligkeit der generierten Ergebnisse sind die, im Vergleich zu „Funktionalität“ und „Entwicklungsstand“, schlechteren Urteile des Merkmals „Benutzbarkeit“ (insbesondere bei den Kennzahlen hinsichtlich der externen Service-Schnittstellen). Dies könnte ein Hinweis darauf sein, dass die Anwendung einen guten Entwicklungsstand hat, aber insgesamt relativ schlecht dokumen-

tiert und umständlich formuliert wurde, was insbesondere bei der Wartung und Weiterentwicklung zu Schwierigkeiten führt.

Da in den beiden Testkonfigurationen unterschiedliche Schwellenwerte für die technischen Kennzahlen verwendet werden, kommt es bei dem Merkmal „Wartbarkeit", insbesondere bei der Bewertung der Komponente „Reise.client" und „Core.client", zu sehr unterschiedlichen Ergebnissen. Auch bei den anderen Komponenten sind Abweichungen festzustellen. Welche der beiden Konfigurationen jedoch aussagekräftigere Ergebnisse liefert, lässt sich derzeit noch nicht beurteilen. Hier besteht ein erheblicher Bedarf, durch das Testen einer größeren Anzahl von Anwendungen, verlässliche Referenzwerte für die verwendeten technischen Kennzahlen zu ermitteln.

# 5 Fazit und Ausblick

Die im Rahmen des Beitrags vorgestellte Methode „Reifegradüberwachung von Software" unterstützt eine systematische und kontinuierliche Reifegradbeurteilung von Software und ermöglicht somit eine konsequente Steuerung der Produktqualität. Die geforderte Zielsetzung, Eigenschaften der Produktqualität transparent zu machen, wird durch die verschiedenen Phasen, das Reifemodell und die unterschiedliche Granularität der Bewertungsverfahren (Kennzahlen- und Merkmalsverdichtung) erfüllt. Bei der Entwicklung der Methode wurde sowohl auf Erkenntnisse aus dem Reifegradmanagement in der Automobilindustrie als auch auf aktuelle Bewertungs- und Messverfahren der Softwareindustrie (Architekturevaluierungsverfahren, Metriken) zurückgegriffen.

In Bezug auf die Kennzahlen und Merkmale sind durchaus weitere Dimensionen der Verdichtung denkbar. Beispielsweise wäre es möglich, Reifegradurteile durch eine weitere Verdichtung auf der Ebene einzelner Subsysteme (Domänen) oder gar für das gesamte Softwaresystem zu gewinnen. Durch die unterschiedliche Granularität der Urteile können somit verschiedene, am Entwicklungsprozess beteiligte Rollen in den Überwachungsprozess mit einbezogen werden. In wie weit hoch verdichtete Kennzahlen und Merkmale aussagekräftige Ergebnisse liefern, bleibt zu überprüfen.

Bei dem abschließenden Vergleich der erstellten Methode „Reifegradüberwachung von Software" mit der Referenzmethode „Reifegrad-Absicherung für Neuteile" wurden folgende wesentliche Gemeinsamkeiten und Unterschiede herausgestellt.

Zu den Gemeinsamkeiten zählen die Vorauswahl kritischer Lieferumfänge/Komponenten, um den Aufwand der Betrachtung kleinstmöglich zu halten [VDA2005, 12] sowie die durchgängige Darstellung des Reifegrades im Projektverlauf [VDA2005, 10]. In beiden Methoden wird darüber hinaus eine starke Einbindung aller am Projekt beteiligten Stakeholder fokussiert [VDA2005, 10]. Eine weitere wesentliche Gemeinsamkeit besteht zudem darin, ein frühzeitiges Aufdecken von Abweichungen von den Projektzielen zu ermöglichen, um diesen entgegensteuern zu können [VDA2005, 12].

Die Unterschiede beider Methoden sind insbesondere auf die verschiedenen Bewertungsgegenstände - Lieferumfänge (materiell) und Softwarekomponente (immateriell) - zurückzuführen. Im Gegensatz zur „Reifegradüberwachung von Software" existiert bei der „Reifegrad-Absicherung für Neuteile" keine Unterteilung von Messintervallen für subjektive und technische Kennzahlen. Des Weiteren beruht die Verrechnungslogik der Methode „Reifegrad-Absicherung für Neuteile" auf nacheinander zu erfüllenden Reifegraden mit unterschiedlichen Messkriterien, die in einer Reifegradkaskade dargestellt werden [VDA2005, 15-20]. Die Berechnung in der Methode „Reifegradüberwachung von Software" basiert hingegen auf einem Ampelbaum, in dem jedes Komponentenurteil auf einem festgelegtem Merkmalssatz für Reife mit gleich bleibenden Kennzahlen beruht (siehe Kapitel 3.2.3). Auch die Definition eines Maßnahmenkatalogs für den Fall negativer Reifeurteile ist in der erstellten Methode noch nicht vorgesehen. Ein solcher Maßnahmenkatalog wird jedoch als sinnvolle Erweiterungsmöglichkeit angesehen.

In Bezug auf den Reifebegriff bewertet die Methode „Reifegrad-Absicherung für Neuteile“ sowohl die Produkt- als auch die Fertigungsprozessreife [VDA2005, 10]. Die Methode „Reifegradüberwachung von Software“ ist allein auf die Bewertung der Produktreife ausgelegt und kann als Ergänzung zu bereits existierenden Verfahren zur Bewertung der Softwareprozessreife verwendet werden. Wie bereits betont, wird aufgrund der starken Wechselwirkungen, die bei der Änderung von Komponenten auftreten können, kein mehrstufiger Reifegrad ermittelt.

Der Aufbau der Methode lässt sich abschließend in einer Methoden-Landkarte zusammenfassen, die die zuvor beschrieben Phasen und deren Bestandteile darstellt (siehe Abbildung 14).

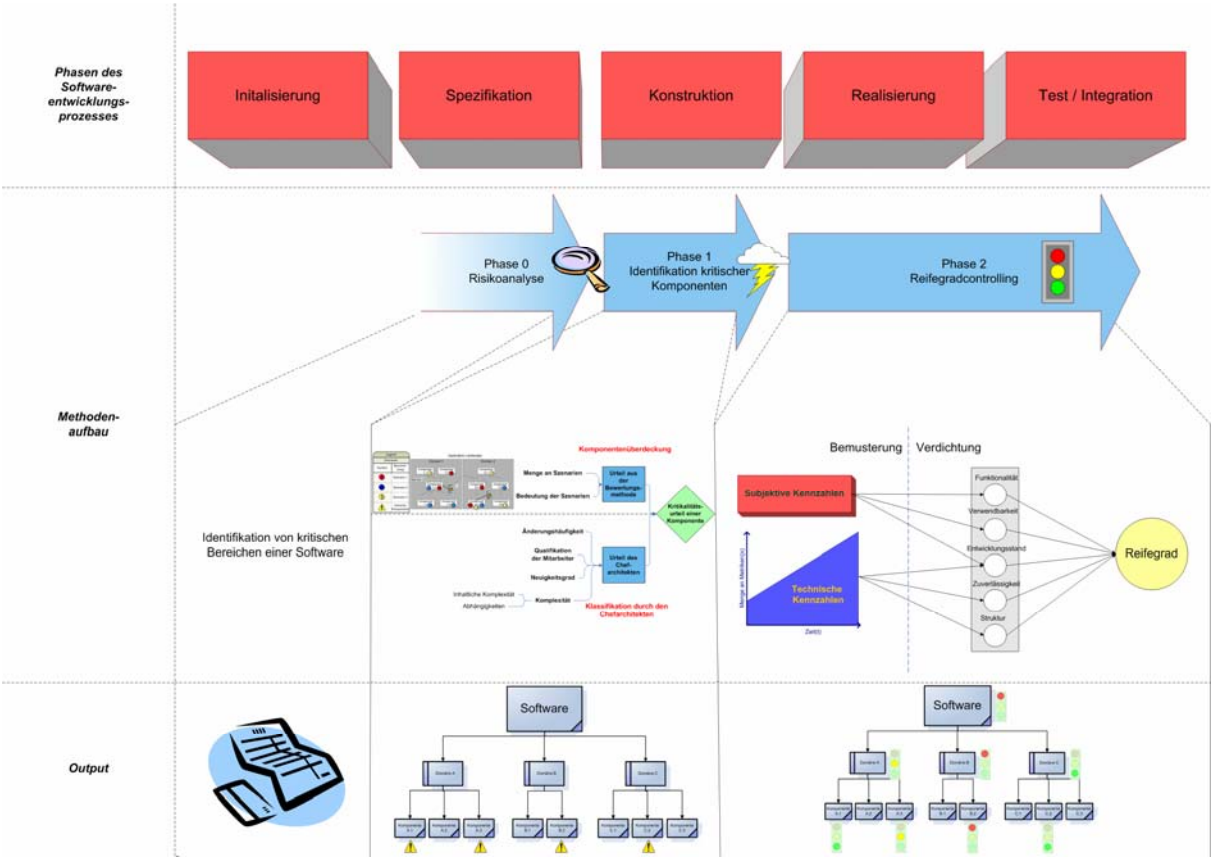



**Abbildung 14: Methoden-Landkarte**

Derzeitige Schwächen bei der Anwendung der Methode sind insbesondere in der Verdichtung von Kennzahlen und im Fehlen aussagekräftiger Referenzwerte für Metriken auf Komponentenebene zu sehen.

Bezüglich der Verdichtung von Kennzahlen bleibt es zu hinterfragen, welche Akzeptanz die gewonnen Ergebnisse besitzen. Beim Einsatz der Methode in aktuellen Entwicklungsprojekten war zu beobachten, dass der Fokus der Projektbeteiligten auf einer kleinen Anzahl wichtiger Indikatoren lag, die nicht weiter verdichtet wurden. Auch die Merkmale des Reifemodells wurden eher als Orientierungshilfe verwendet und die Ergebnisse letztendlich „am runden Tisch“ besprochen.

Die generierten Reifeaussagen in der 2. Phase der Methode hängen stark von den zugrunde liegenden Kennzahlen ab. Für die Bewertung von Komponenten mithilfe technischer Kennzahlen liegen kaum brauchbare Referenzwerte vor, die die Definition von Intervallen erlauben. Insofern sollte vor dem konkreten Einsatz der Methode sichergestellt werden, dass aussagekräftige Referenzwerte für die verwendeten technischen Kennzahlen vorliegen (vgl. Kapitel 4.2).

Durch den generischen Aufbau der Methode existieren zahlreiche Konfigurations- und Erweiterungsmöglichkeiten. Dazu zählen insbesondere die Definition von Zusatzkriterien in Phase 1 und

das Aufstellen eines eigenen Kennzahlensatzes für Phase 2. Auch die Gewichtungen in den Phasen 1 und 2 können genutzt werden, um die Methode auf die Bedürfnisse des Anwenders abzustimmen. Damit besteht die Möglichkeit, die Methode in vielen Bereichen auf Basis des im Umgang mit ihr gewonnenen Erfahrungswissens anzupassen, ohne sie vollständig überarbeiten zu müssen.

# Literatur


[Bruh2001] Bruhn, Manfred: **Marketing. Grundlagen für Studium und Praxis**, (5.Aufl.), Gabler Verlag, Wiesbaden, 2001.

[BeRi2007] Bennicke, Marcel; Richter, Jan-Peter: **Architecture of a Generic Software Control Centre**, 1st Workshop on Measurement-based Cockpits for Distributed Software and Systems Engineering Projects (SOFTPIT 2007), In conjunction with the IEEE International Conference on Global Software Engineering, München, 2007

[BSRR2007] Bennicke, Marcel; Steinbrückner, Frank; Radicke, Mathias; Richter, Jan-Peter: **Das sd&m Software Cockpit: Architektur und Erfahrungen**, Workshop on Applied Program Analysis 2007, 37. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Lecture Notes in Informatics (LNI), GI Proceedings 110, Band 2, Bremen, 2007.

[ClKK2002] Clements, Paul; Kazman Rick; Klein, Mark: **Evaluating Software Architecture – Methods and Case Studies**, Addison-Wesley, Boston, 2002.

[CrLL2000] Crnkovic, Ivica; Larsson, Magnus; Lüders, Frank: **State of the Practice: Component-based Software Engineering Course**, Proceedings of 3rd International Workshop of Component-Based Software Engineering, IEEE Computer Society, Limerick, Ireland, 2000.

[CSMD2001] Chulani, Sunita; Santhanam, P.; Moore, Darrell; Davidson, Gary: **Deriving a software quality view from customer satisfaction and service data**, Proceedings of the European Software Control and Metrics (ESCOM) Conference, London, 2001, S. 225-232.

[DIN1995] DIN EN ISO 8402, Qualitätsmanagement Begriffe, DIN, 1995.

[DoNi2002] Dobrica, Liliana; Niemelä, Eila: **A Survey on Software Architecture Analysis Methods**, IEEE Transactions on Software Engineering 28 No. 7, IEEE Computer Society, 2002, S. 638-653.

[EiHM2007] Eicker, Stefan; Hegmanns, Christian; Malich, Stefan: **Auswahl von Bewertungsverfahren für Softwarearchitekturen**, ICB-Research Report No.14, Essen, 2007.

[Ewus1997] Ewusi-Mensah, Kweku: **Critical issues in abandoned information systems development projects**, Communications of the ACM 40 No.9, 1997, S. 74-80.

[GoBr2004] Goulão, Miguel Alfonso; Brito e Abreu, Fernando Manuel: **Cross-Validation of a Component Metrics Suite**, Jornadas Ibéricas de Ingeniería del Software y Bases de Datos (JISBD'2004). Málaga, , 2004-11, Abruf am: 2007-08-24.

[Grad1992] Grady, Robert B.: **Practical software metrics for project management and process improvement**, Prentice Hall, New Jersey, 1992.

[GrTe2005] Gröttrup, Ulrike; Tensi, Thomas: **Komponentenmodellierung mit den Mitteln der UML 1.x und UML 2.0**, In: Breu, Ruth; Matzner, Thomas; Nickl, Friederike; Wiegert, Oliver (Hrsg.): Software Engineering. Oldenbourg Wissenschaftsverlag, München 2005, S. 89-108.

[HaFP2005] Hartmann, Júlio; Fontoura, Lisandra M.; Price, Roberto T.: **Using Risk Analysis and Patterns to Tailor Software Processes**, XIX Simpósio Brasileiro de Engenharia de Software, Uberlândia, 2005.

[ISO2001] International Standard: **Software Engineering – Product Quality - Quality Model (Part 1)**, ISO/IEC 9126-1, 2001.

[KiSc2003] Killi, Ole Morten, Schwarz, Henrik: **An Empirical Study of Quality Attributes of the GSN System at Ericsson**, MSc Thesis, http://www.idi.ntnu.no/grupper/su/su-diploma-2003/killi_schwarz-empirical_study_ericsson_external-v1.pdf, NTNU, 2003, Abruf am: 2007-08-24.

[Lako1996] Lakos, John: **Large-Scale C++ software design**, Addison Wesley, Redwood City, 1996.

[Lewa1998] Lewandowski, Scott M.: **Frameworks for component-based client/server computing**, ACM Computing Surveys, 30 (1), Providence, 1998, S. 3-27.

[McIl1968] McIlroy, M. Douglas: **Mass produced software components**, In: Naur, Peter; Randell, Brian (Hrsg.): Proceedings of NATO Conference on Software Engineering. Garmisch 1968, S. 88-99.

[MeHS1998] Mellis, Werner; Herzwurm, Georg; Stelzer, Dirk: **TQM der Softwareentwicklung**, (2.Aufl.), Vieweg, 1998.

[Meye1995] Meyer, Bertrand: **Object Success: A Manager's Guide to Object-Oriented Technology And Its Impact On the Corporation**, Prentice-Hall, Upper Saddle River, 1995.

[Mili2005] Milicic, Drazen: **Software Quality Models and Philosophies**. In: Software quality attributes and trade-offs. Blekinge Institute of Technology, http://www.bth.se/tek/besq.nsf/(WebFiles)/5A52350A52726F51C12570A8004CB613/$FILE/Software_quality_attributes.pdf, 2005, Abruf am: 2007-08-24, S. 3-19.

[Mill1956] Miller, George A.: **The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information**. In: The Psychological Review Vol. 63, S.81-97, 1956.

[NaHe2003] Narasimhan, V. Lakshmi; Hendradjaya, Bayu: **A New Suite of Metrics for the Integration of Software Components**, 1st International Workshop on Object Systems and Software Architectures (WOSSA 2004), Victor Harbor, 2004.

[Oreg2002] O´Regan, Gerard: **A Practical Approach to Software Quality**, Springer Verlag, New York, 2002.

[PTLS2005] Parviainen, Päivi; Tihinen, Maarit; Lormans, Marco; van Solingen, Rini: **Requirements Engineering: Dealing with the Complexity of Sociotechnical Systems Development**, Requirements Engineering for Sociotechnical Systems, Mate, Jose Luis; Silva, Andres (Hrsg.), Information Science Publishing, Hershey, 2005.

[Scha2001] Schach, Stephen R.: **Object-Oriented and Classical Software Engineering**, (5.Aufl.), McGraw-Hill, New York, 2001.

[Sied2003a] Siedersleben, Johannes: **Quasar: Die sd&m Standard-Architektur, Teil 1**, sd&m AG, München, 2003.

[Sied2003b] Siedersleben, Johannes: **Softwaretechnik Praxiswissen für Softwareingenieure**, (2.Aufl.), Carl Hanser Verlag, 2003.

[SSBS2004] Scheffczyk, Jan; Stutz, Christiane; Borghoff, Uwe M.; Siedersleben, Johannes: **Formale Konsistenzsicherung in informellen Software-Spezifikationen**, Informatik

Forschung und Entwicklung, Ausgabe Bd. 19/Heft 1, Springer Verlag, Trier, 2004, S. 17-29.

[Szyp2002] Szyperski, Clemens: **Component Software: Beyond Object-Oriented Programming**, Second Edition, ACM Press, Addison-Wesley Professional, New York, 2002.

[Thal2000] Thaller, Georg Erwin: **ISO 9001 Software-Entwicklung in der Praxis**, Verlag Heinz Heise, Hannover, 2000.

[UmEm2005] Umarji, Medha; Emurian, Henry: **Acceptance Issues in Metrics Program Implementation**, Software Metrics, 2005. 11th IEEE International Symposium, IEEE Computer Society Press, Baltimore, 2005, S. 10-19.

[VDA2005] Verband der Automobilindustrie: **Qualitätsmanagement in der Automobilindustrie**, Reifegrad-Absicherung für Neuteile, Henrich Druck + Medien, Frankfurt am Main 2005.

[VoHH2006] Voß, Markus; Hess, Andreas; Humm, Bernhard: **Towards a Framework for Large Scale Quality Architecture**, In: Hofmeister, Ch., et.al. (Eds.): Perspectives in Software Quality – Short Papers of the 2nd International Conference on the Quality of Software Architectures (QoSA), Interner Bericht 2006-10, Universität Karlsruhe, Fakultät für Informatik, ISSN 1432-7864, 2006.

[Wall2001] Wallmüller, Ernest: **Software Qualitätsmanagement in der Praxis**, Carl Hanser Verlag, München, 2001.

[Wißl2005] Wißler, Frank Eugen: **Ein Verfahren zur Bewertung technischer Risiken in der Phase der Entwicklung komplexer Serienprodukte**, Dissertation, Stuttgart, 2005.

[Zuse1997] Zuse, Horst: **A Framework of Software Measurement**, De Gruyter, Berlin, 1997.

# Previously published ICB - Research Reports

## 2007

*No 19 (June 2007)*
*Schauer, Carola: "Relevance and Success of IS Teaching and Research: An Analysis of the ‚Relevance Debate'*

*No 18 (May 2007)*
*Schauer, Carola: "Rekonstruktion der historischen Entwicklung der Wirtschaftsinformatik: Schritte der Institutionalisierung, Diskussion zum Status, Rahmenempfehlungen für die Lehre"*

*No 17 (May 2007)*
*Schauer, Carola; Schmeing, Tobias: "Development of IS Teaching in North-America: An Analysis of Model Curricula"*

*No 16 (May 2007)*
*Müller-Clostermann, Bruno; Tilev, Milen: "Using G/G/m-Models for Multi-Server and Mainframe Capacity Planning"*

*No 15 (April 2007)*
*Heise, David; Schauer, Carola; Strecker, Stefan: "Informationsquellen für IT-Professionals – Analyse und Bewertung der Fachpresse aus Sicht der Wirtschaftsinformatik"*

*No 14 (March 2007)*
*Eicker, Stefan; Hegmanns, Christian; Malich, Stefan: "Auswahl von Bewertungsmethoden für Softwarearchitekturen"*

*No 13 (February 2007)*
*Eicker, Stefan; Spies, Thorsten; Kahl, Christian: "Softwarevisualisierung im Kontext serviceorientierter Architekturen"*

*No 12 (February 2007)*
*Brenner, Freimut: "Cumulative Measures of Absorbing Joint Markov Chains and an Application to Markovian Process Algebras"*

*No 11 (February 2007)*
*Kirchner, Lutz: "Entwurf einer Modellierungssprache zur Unterstützung der Aufgaben des IT-Managements – Grundlagen, Anforderungen und Metamodell"*

*No 10 (February 2007)*
*Schauer, Carola; Strecker, Stefan: "Vergleichende Literaturstudie aktueller einführender Lehrbücher der Wirtschaftsinformatik: Bezugsrahmen und Auswertung"*

*No 9 (February 2007)*
*Strecker, Stefan; Kuckertz, Andreas; Pawlowski, Jan M.: "Überlegungen zur Qualifizierung des wissenschaftlichen Nachwuchses: Ein Diskussionsbeitrag zur (kumulativen) Habilitation"*

*No 8 (February 2007)*
*Frank, Ulrich; Strecker, Stefan; Koch, Stefan: "Open Model - Ein Vorschlag für ein Forschungsprogramm der Wirtschaftsinformatik (Langfassung)"*

## 2006

*No 7 (December 2006)*
*Frank, Ulrich: "Towards a Pluralistic Conception of Research Methods in Information Systems Research"*

*No 6 (April 2006)*
*Frank, Ulrich: "Evaluation von Forschung und Lehre an Universitäten – Ein Diskussionsbeitrag"*

*No 5 (April 2006)*
*Jung, Jürgen: "Supply Chains in the Context of Resource Modelling"*

*No 4 (February 2006)*
*Lange, Carola: "Development and status of the Information Systems / Wirtschaftsinformatik discipline: An interpretive evaluation of interviews with renowned researchers, Part III – Results Wirtschaftsinformatik Discipline"*

## 2005

*No 3 (December 2005)*
*Lange, Carola: "Development and status of the Information Systems / Wirtschaftsinformatik discipline: An interpretive evaluation of interviews with renowned researchers, Part II – Results Information Systems Discipline"*

*No 2 (December 2005)*
*Lange, Carola: "Development and status of the Information Systems / Wirtschaftsinformatik discipline: An interpretive evaluation of interviews with renowned researchers, Part I – Research Objectives and Method"*

*No 1 (August 2005)*
*Lange, Carola: „Ein Bezugsrahmen zur Beschreibung von Forschungsgegenständen und -methoden in Wirtschaftsinformatik und Information Systems"*

The Institute for Computer Science and Business Information Systems (ICB), located at the Essen Campus, is dedicated to research and teaching in Applied Computer Science, Information Systems as well as Information Management. The ICB research groups cover a wide range of expertise:

| Research Group | Core Research Topics |
|---|---|
| **Prof. Dr. H. H. Adelsberger**<br>Information Systems for Production and Operations Management | E-Learning, Knowledge Management, Skill-Management, Simulation, Artificial Intelligence |
| **Prof. Dr. P. Chamoni**<br>MIS and Management Science / Operations Research | Information Systems and Operations Research, Business Intelligence, Data Warehousing |
| **Prof. Dr. F.-D. Dorloff**<br>Procurement, Logistics and Information Management | E-Business, E-Procurement, E-Government |
| **Prof. Dr. K. Echtle**<br>Dependability of Computing Systems | Dependability of Computing Systems |
| **Prof. Dr. S. Eicker**<br>Information Systems and Software Engineering | Process Models, Software-Architectures |
| **Prof. Dr. U. Frank**<br>Information Systems and Enterprise Modelling | Enterprise Modelling, Enterprise Application Integration, IT Management, Knowledge Management |
| **Prof. Dr. M. Goedicke**<br>Specification of Software Systems | Distributed Systems, Software Components, CSCW |
| **Prof. Dr. R. Jung**<br>Information Systems and Enterprise Communication Systems | Process, Data and Integration Management, Customer Relationship Management |
| **Prof. Dr. T. Kollmann**<br>E-Business and E-Entrepreneurship | E-Business and Information Management,E-Entrepreneurship/ E-Venture, Virtual Marketplaces and Mobile Commerce, Online-Marketing |
| **Prof. Dr. B. Müller-Clostermann**<br>Systems Modelling | Performance Evaluation, Modelling and Simulation, SAP Capacity Planning for R/3 and mySAP.com, Tools for Queueing Network Analysis and Capacity Planning, Communication Protocols and Distributed Systems, Mobile Systems |
| **Prof. Dr. K. Pohl**<br>Software Systems Engineering | Requirements Engineering, Software Quality Assurance, Software-Architectures, Evaluation of COTS/Open Source-Components |
| **Prof. Dr.-Ing. E. Rathgeb**<br>Computer Networking Technology | Computer Networking Technology |
| **Prof. Dr. R. Unland**<br>Data Management Systems and Knowledge Representation | Data Management, Artificial Intelligence, Software Engineering, Internet Based Teaching |
| **Prof. Dr. S. Zelewski**<br>Institute of Production and Industrial Information Mangement | Industrial Business Processes, Innovation Management, Information Management, Economic Analyses |

For more information visit us on the Web: **http://www.icb.uni-due.de**

ISSN 1860-2770