Metriken-basierte Bewertung von Software-Komponenten

Reiner Dumke*, Andreas Schmietendorf*#

* Otto-von-Guericke-Universität Magdeburg, Fakultät Informatik, Institut für verteilte Systeme, Postfach 41 20, D-39016 Magdeburg, Telefon: ++49-391-6718664, Fax: ++49-391-6712810, E-Mail: dumke|schmiete@ivs.cs.uni-magdeburg.de

\# T-Nova Deutsche Telekom Innovationsgesellschaft mbH, Entwicklungszentrum Berlin, Wittestraße 30N, D-13476 Berlin, Telefon: ++49-30-43577-633, Fax: ++49-30-43577-460, E-Mail: A.Schmietendorf@telekom.de

Abstract: Der vorliegende Artikel zeigt die Ergebnisse einer durchgeführten Fallstudie zur Analyse telekomspezifischer und informationstechnischer Softwarekomponenten. Diese wurden Anhand eines definierten Kriteriensatzes einer allgemeinen Bewertung unterzogen. Aufbauend auf diesen Ergebnissen wird ein Konzept zur metriken-basierten Bewertung von JavaBeans vorgestellt, wobei erste empirische Ergebnisse präsentiert werden.

Subject: Komponenten, Java Beans, Metriken, Meßmodelle

1 Einführung

Die industrielle Produktion zeichnet sich durch einen hohen Grad an Wiederverwendung bereits erzeugter Zwischenprodukte aus. Insbesondere wird so eine Verbesserung der Qualität, der Produktbereitstellungzeiten und der Produktivität erreicht. Grundlage eines derartigen Vorgehens ist ein arbeitsteiliger Prozeß zwischen „Zulieferer“ und „Endmontage“ sowie Standards hinsichtlich der funktionalen und qualitativen Eigenschaften der eingesetzten Zwischenprodukte, um die kundenseitigen Anforderungen an das endgültige Produkt erfüllen zu können. Insbesondere der Verwendung von Komponenten wird das Potential zugesprochen, die Softwareentwicklung aus der heute zumeist handwerklichen Fertigung zu einer echten Ingenieur-Disziplin zu führen.

2 Fallstudien zur Untersuchung von Softwarekomponenten

2.1 Zielstellung der Fallstudien

Die hier in Auszügen dargestellten Fallstudien dienten dazu, bereits existierende Softwareentwicklungen auf der Basis von Komponenten einer plazierenden Bewertung zu unterziehen. Insbesondere sollten so Schlußfolgerungen für den Einsatz von Komponenten bei der Deutschen Telekom AG und die Auswirkungen hinsichtlich Qualität, Produktivität, Einsetzbarkeit, usw. abgeleitet werden.

2.2 Komponenten für Intelligent Network Anwendungen

Das Intelligente Netzwerk (IN) ist sowohl eine Netzwerk-Architektur als auch eine Technologie zur Entwicklung und Bereitstellung von Diensten in der Telekommunikation. Das IN existiert zusätzlich zum einfachen Telefonnetzwerk und besitzt eine gewisse Steuerfunktion. Ziel ist es, Verbindungsanforderungen (Call‘s) nach einer einfach zu konfigurierenden Logik zu verarbeiten. Die Entwicklung von IN-Diensten (z.B. Televotum „TED“) erfolgt auf der Basis von SIBs. SIBs (Service Independent building Blocks) sind wiederverwendbare Grundbausteine, die jeweils eine einzelne komplexe Aktivität beschreiben und in ihrer Komposition einen IN-Dienst repräsentieren. Durch diese Modularität ist es möglich, gleichartige Teilaufgaben verschiedener Dienste an ein und dasselbe SIB zu delegieren und damit bei der Entwicklung neuer Dienste auf bereits bestehende Komponenten zurückzugreifen.

Die Hauptmerkmale der SIBs sind zusammenfassend:

- SIBs modellieren Dienste,
- SIBs sind standardisierte, wiederverwendbare, netztransparente Einheiten,
- SIBs sind unabhängig vom spezifischen Dienst, den sie beschreiben, bzw. von irgendwelchen physikalischen Architekturüberlegungen,
- SIBs besitzen vereinheitliche, stabile Schnittstellen, mit einem oder mehreren Inputs sowie mit einem oder mehreren Outputs.

Durch Kombination dieser wiederverwendbaren SIBs innerhalb grafisch orientierter Entwicklungsumgebungen lassen sich neue Dienste zusammenstellen, ohne am IN-System etwas zu ändern. Zur Beschreibung eines SIB‘s als wiederverwendbare Komponente gehören alle Informationen aus der Datenumgebung des SIB sowie einige zusätzliche Informationen. Folgende Beschreibungsschablone für diese textorientierte Möglichkeit wird dabei verwendet:

<u>Pflichtangaben:</u>

- Textdefinition des SIB,
- die meisten SIBs erwarten spezielle Eingabeparamter: z.B. Zeitfenster im SIB Anrufverteilung nach Nutzer-Zeitfenstern, Suchschlüssel im SIB

´getUserRec´ oder Defaultansagekennung im SIB ´Announce´,

- Service Support Data (SSD) sind statische Daten und Tabellen, die während der Ausführung des SIBs zur Verfügung stehen. Z.B.: Ansagetabelle, Gebührentabelle, Nutzertabelle,
- Call Instance Data (CID) sind dynamische, d.h. callspezifische Daten. Sie existieren nur während des Rufes, nach beendetem Ruf sind sie gelöscht. Sie werden unterteilt in:
  - Eingabe-CID (Call Instance Data),
  - Ausgabe-CID (Call Instance Data),
- Funktionsweise - Beschreibung der vom SIB ausgeführten Operationen.

Zusatzangaben:

- Identifikation der Dienste für welches das SIB genutzt werden kann,
- grafische Repräsentation,
- SDL-Diagramm (Standard Description Language).

Für Designer und Implementierer von IN-Diensten reicht oft eine formalisierte, komprimierte Sprachbeschreibung (z.B. IDL-ähnlich) aus. Eine solche Beschreibungsschablone enthält dann im wesentlichen die:

- (erweiterte) Textdefinition,
- Input-Parameter,
- Output-Parameter,
- Beschreibung der Attribute.

2.3 Komponenten für TMN-Anwendungen

Die Entwicklung von Informationssystemen zum Management öffentlicher Netze der Telekommunikation (TMN Telecommunication-Management-Network) erfolgt auf der Basis sogenannter MOs (Managed Objects). Ein MO ist eine wiederverwendbare Komponente und abstrahiert eine zu managende Ressource des Telekommunikationssystems, wie z.B. eine Vermittlungsstelle, in einer für das Informationssystem adäquaten Form. Die Eigenschaften eines Managed Objects sind folgendermaßen definiert:

- **Attributes und Attribute Groups** (Attribute), die die inneren Zustände nach außen sichtbar machen und durch Operationen geändert werden können,
- **Actions** (Aktionen), die auf das MO angewendet werden können,
- **Behaviour** (Verhalten) des MO's, das als Reaktion auf Ereignisse und als Antwort auf Operationen auftreten kann,
- **Notifications** (Nachrichten), die das MO senden kann,
- **Parameter** von Aktionen, die ihr Verhalten bestimmen,
- **Name Binding** (Namensbindung) realisiert die Enthaltenseinshierarchie – MOs werden durch Namen identifiziert und sind in Form eines Baumes in der MIB (Management Information Base) logisch angeordnet,
- **Packages** (Pakete), die Attribute(-Gruppen), Nachrichten und Operationen enthalten können.

Durch die Verwendung von **formalen Sprachen** wird eine systemunabhängige Spezifikation von Daten und Protokollen erreicht. Hierfür ist eine aus ASN.1 (Abstract Syntax Notation 1, X.208) abgeleitete Sprache mit dem Namen GDMO (Guidelines for the Definition of Managed Objects, X.722) entstanden. Kernpunkte der Spezifikation ist die Abstraktion von Eigenschaften einer zu verwaltenden Ressource gegenüber dem Managementsystem, die Modularisierung von Managementinformationen durch Packages sowie ein objektorientiertes Design über „Managed Object Classes" inklusive der von der objektorientierten Programmierung her bekannten Konzepte von Vererbung (inheritance) und Datenkapselung (encapsulation). Die Datendeklaration für die Realisierung von Managed Objects mit gleichen Eigenschaften bezeichnet man als "Managed Object Class".

2.4 Komponenten am Beispiel von Java Beans

Das von SUN angebotene JavaBean-Modell bietet die Möglichkeit, Softwarekomponenten mittels der Spache Java implementieren zu können. Der Java-Sprachumfang wurde dafür nicht geändert, wohl aber das JDK[1] um ein weiteres Package ergänzt.

Entsprechend der von SUN ausgegebenen Spezifikation handelt es sich bei Java-Beans um wiederverwendbare Softwarekomponenten, welche im Rahmen von Builder-Tools visuell hinsichtlich anpaßbarer Eigenschaften verändert werden können. Die Technologie der Java-Beans basiert auf den folgenden Grundprinzipien:

Introspection: Diese sowohl implizit als auch explizit angebotene Eigenschaft bietet die Möglichkeit, daß Java-Beans zur Laufzeit Auskunft hinsichtlich ihrer möglichen Funktionen und Eigenschaften geben können. Speziell können die als public definierten Methoden sowie ausführbare Ereignisse abgefragt werden. Auf dieser Grundlage kann innerhalb entsprechender Bean-Builder-Tools auf einen direkten Zugriff auf den Quellcode verzichtet werden.

Customisation: Existierende Beans können entsprechend der aktuellen Erfordernisse visuell abgewandelt und angepaßt werden. Hinsichtlich der Erweiterbarkeit folgen diese der objekorientierten Philosophie mit Polymorphie und Kapselung.

Event-Handling: Beans können 'Ereignisse' auf der Basis des ab der JDK 1.1 Version angebotenen Delega-

[1] JDK Java Development Kit

tion-Event-Handling-Modell (mittels Event-Sources und Event-Listenern) untereinander austauschen.

Properties: Die Eigenschaften ("Properties") einzelner Beans können leicht verändert werden. Damit der GUI-Builder die Properties später als solche erkennt und ein Interface für sie anbieten kann, muß deren Definition einer bestimmten Syntax entsprechen: den ***Signature Patterns***.

Methoden: werden genutzt, um entsprechende Properties zu ändern. Grundsätzlich werden neben den Properties alle als „public" definierten Methoden für die Beans angeboten. Über eine BeanInfo-Klasse kann speziell definiert werden, welche Methoden für den späteren Nutzer des Bean tatsächlich sichtbar sein sollen.

Persistenz: Für die dauerhafte Abspeicherung von Daten bietet das Bean-Modell einen entsprechenden Persistenzmechanismus. Diese Funktionalität wird unter anderem benötigt, um Konsistenzbedingungen bei der visuellen Anpassung von Properties (Ändernde Zugriffe auf Daten der Bean-Komponente) zu gewährleisten.

2.5 Vergleichende Bewertung der Ansätze

Tabelle 1: Vergleich der untersuchten Komponentenansätze

Kriterien	**TMN-MO**	**IN-SIB (CS1R)**	**Java-Beans**
Eigenständige Funktionalität (Plug & Play Einsatz)	gut	sehr gut	sehr gut
Klar definierte Schnittstellen	sehr gut	sehr gut	gut
Kommunikationsfähigkeit (Interaktion)	schlecht[2]	schlecht[3]	gut
Lose Kopplung	gut	gut	sehr gut
Plattformunabhängigkeit			
- Fachliche Sicht (allgemeingültiger Standard vorh.)	sehr gut	sehr gut	gut[5]
- IV-technische Sicht	schlecht	schlecht[4]	sehr gut
Aussagen über Qualität nach ISO 9126 (Gütesiegel)			
- Übertragbarkeit (Anpaßbarkeit, Installierbarkeit)	gut	schlecht	gut
- Anwendbarkeit (Erlernbarkeit, Beherrschbarkeit, Verständlichkeit)	gut	gut	gut
- Effizienz (raum-, zeit- und ressourcenbezogen)	keine Angaben	keine Angaben	keine Angaben
- Funktionalität (Angemessenheit, Interoperabilität, Genauigkeit)	keine Angaben	keine Angaben	gut
- Zuverlässigkeit (Fehlertoleranz, geringe Fehlerhäufigkeit, Fehlerverfolgbarkeit)	keine Angaben	keine Angaben	keine Angaben
- Wartbarkeit (Analysierbarkeit, Änderbarkeit, Stabilität, Testbarkeit)	schlecht	schlecht	schlecht
Beschreibungsformen der Komponenten			
- Grafische Darstellung	ja	ja[6]	ja[7]
- textuelle Beschreibung	ja	ja	ja
- formalisierte Sprachbeschreibung	ja	ja (vgl. IDL)	nein
Navigierbarkeit /Intuitive Nutzbarkeit	gut	gut	gut
Standardisierung	sehr gut	sehr gut[8]	gut
Domainspezifika	sehr gut	sehr gut	gut
Dokumentierbarkeit	schlecht	gut	schlecht
Muster von Kopplungsarten	keine Angaben	vorhanden[9]	vorhanden
Vergröberungsmöglichkeiten (Hierarchiebildung)	keine Angaben	nicht vorh.[10]	vorhanden[11]
Granularität	optimal	nicht optimal[12]	optimal

[2] wegen ausgeprägter Domain-Spezifika
[3] wegen ausgeprägter Domain-Spezifika
[4] zumeist werden firmenspezifische Anpassungen vorgenommen
[5] kein firmenunabhängiger Standard verfügbar
[6] Firmenstandard
[7] Firmenstandard (vgl. SUN)
[8] mehrere Standardisierungsgremien vorhanden (ITU, ETSI)
[9] es sind Regeln und Einschränkungen für die Kopplung vorhanden, aber keine direkten Muster
[10] ab CS2 vorhanden
[11] aus Beans können wiederum Beans erstellt werden (kein implementierungsunabhängiges Modell)
[12] eine feinere Granulatität wäre wünschenswert

3 Formalisierung der Bewertungskriterien durch Metriken

3.1 Voraussetzungen für den Einsatz von Software-Metriken

Für den effektiven Einsatz von Software-Metriken bzw. die Einführung sogenannter Metrikenprogramme wurde von der Otto-von-Guericke-Universität Magdeburg das CAME-Framework [Dumke 1998] vorgeschlagenen, welches sich an der Klassifikation von [Fenton 1991] mit Produkt, Prozeß und Ressource als Ansatzpunkte für die Softwaremessung orientiert.

Das Framework besteht aus einer CAME[13]-Strategie, einem CAME[14]-Framework und den CAME[15]-Tools für die Aufnahme und Verarbeitung von Metriken überhaupt. Während die CAME-Strategie auf Aspekte wie die Notwendigkeit der Existenz einer Gruppe, welche die Einführung von Softwaremetriken vorantreibt, und den Bedarf von Managemententscheidungen hinweist, bezieht sich das CAME-Framework auf z.B. den Auswahlprozeß von Metriken, die Analyse von Skaleneigenschaften dieser Metriken, den erreichten Abdekkungsgrad und die Effizienz des Metrikeneinsatzes durch eine Toolunterstützung. Da bei der Deutschen Telekom bereits strategische Entscheidungen hinsichtlich des Einsatzes von Softwaremetriken vorliegen, konzentriert sich die Analyse auf die Metrikenauswahl (CAME-Framework) bzw. die Ermittlung eines Abdekkungsgrades verfügbarer Metriken für die Wiederverwendbarkeit überhaupt. Konkret werden für die Metrikenauswahl die folgenden Kriterien herangezogen:

Zweckmäßigkeit - zielt auf die Frage, inwieweit ein ausgewähltes Maß die im Unternehmen gesetzten Zielstellungen, wie z.B. das Senken von Fehlerraten, erfüllen kann. Hierfür kann sinnvoll die **G**oal-**Q**uestion-**M**etric Methode nach [Basili 1988] verwendet werden.

Merkmalsabdeckung - Die ausgwählten Metriken sollten auf möglichst viele innerhalb der Entwicklung produzierte Ergebnisse bzw. Software-Artefakte (z.B. Modelle, Programmcode, Dokumentationen) und auf alle Phasen der Entwicklung angewendet werden können.

Aufwandsminimierung - Die Ausführung kontinuierlicher Messungen innerhalb des Software-Life-Cycle erfordert die Nutzung von Meßtools, welche weitgehend automatisch Metriken aus vermessenen Software-Artefakten zu Verfügung stellen können.

[13] C - Community, A - Acceptance, M - Motivation, E - Engagement
[14] C - Choice, A - Adjustment, M - Migration, E - Efficiency
[15] C - Computer, A - Assisted, M - Measurement, E - Evaluation

Empirische Bewertung - Da zu Beginn der Einführung eines Softwaremaßes zumeist keine eigenen Erfahrungen vorliegen, ist es sinnvoll, die innerhalb der Meßtools bzw. in der Literatur vorgeschlagenen „Anfangseinstellungen“ (Schwellwerte) zu übernehmen.

Datenschutz und -sicherheit - Es sollte nicht möglich sein, anhand aufgenommener Metriken Rückschlüsse auf personenbezogene Daten durchführen zu können.

Skaleneigenschaften (nominal, ordinal, intervall, ratio) - der verwendeten Metriken bestimmen den Informationsgehalt der verwendeten Daten und die über diesen ausführbaren statistischen Analysemethoden und mathematische Operationen.

Persistenz – Für Langzeitanalysen und Durchführung von statistischen Verfahren (strukturprüfende- und strukturentdeckende) müssen Metriken dauerhaft in entsprechenden Metrikendatenbanken gespeichert werden können. Dabei dürfen die Zusammenhänge zum Diskurs der Messung nicht verloren gehen.

2.2 Empirische Untersuchungen verfügbarer Bean-Bibliotheken

Die im vorherigen Kapitel durchgeführte Bewertung der Komponententechnologien TMN-MO, IN-SIB und JavaBeans erfolgte auf der Basis ordinaler, d.h. plazierender Bewertungskriterien. Die ermittelten „Meßwerte“ (optimal, sehr gut, gut, vorhanden, schlecht, keine Angaben) wurden durch ein Bewertungsteam vergeben und tragen somit einen subjektiven Charakter. Wenngleich die untersuchten Komponenten in vielen Aspekten den Voraussetzungen für eine erfolgreiche Wiederverwendung entsprechen, kann insbesondere bei der Zusicherung qualitativer Eigenschaften (entsprechend ISO 9126) der Bedarf weiterführender Untersuchungen ausgemacht werden. Diese Untersuchungen betreffen sowohl die Konkretisierung und Validierung des Bewertungsmodells als auch die Erweiterung des Erfahrungshintergrunds in weiteren Anwendungs-Domainen.

In einem ersten Schritt werden derzeit empirische Untersuchungen anhand im Quellcode verfügbarer JavaBeans-Komponentenbibliotheken durchgeführt. Ziel ist es, Lösungsansätze und Methoden aufzuzeigen, welche die Erzeugung von Komponenten mit qualitätsgesicherten Eigenschaften nach ISO 9126 ermöglichen sowie die qualitativen Eigenschaften der Komponenten innerhalb dieser selbst dokumentieren. Bei der Wiederverwendung soll der Entwickler in die Lage versetzt werden, sich von den qualitativen Eigenschaften durch Metriken, die das Außenverhalten der Komponenten beschreiben, zu überzeugen. Im folgenden sind Beispiele für erwartete Aussagen sowie die Ergebnisse ausgewählter initialer Messungen dargestellt:

- Metriken, die die erfolgreiche Wiederverwendung für andere Projekte festhalten,
- Statische Quellcodemetriken des Java Bean hinsichtlich folgender Eigenschaften,

- Größe und Umfang einer Java Bean Komponente im Quell- und Java-Bytecode,
- Komplexitätsmetriken (Komponenten sollten eine niedrige Modul-Komplexität haben),
- Kommentierung einer Komponente (z.B. prozentualer Anteil Kommentarzeilen),
- Metriken zur externen Abhängigkeiten:
  - niedrige fan-in- und fan-out-Werte werden erwartet,
  - Übersichtliche Anzahl von Ein- und Ausgabeparametern,
- Anzahl Packages/Klassen je Komponente: ein Bean sollte mind. **3 Klassen** enthalten, das eigentliche Bean, eine Bean-Info-Klasse und eine Customizer bzw. Editor-Klasse,
- Abstraktheit der Komponente als Maß für ihre Einsatzvielfalt,

- Metriken zur Testabdeckung und Nachweis geringer Fehlerraten,
- Metriken zur Darstellung des Performanceverhaltens (raum-, zeit- und ressoucenbezogen).

Tabelle 2: Ausgewählte Ergebnisse einer initialen Vermessung von Beans

Meßwert x		**PACKAGE**	**METRIKEN DER EIGENTLICHEN BEAN-KLASSE**				
		Efferente Kopplung	**Vererbungstiefe**	**% Anteil von public Methoden**	**Maximaler McCabe je Methode**	**% Anteil Kommentare**	**eLOC**
BDK[16] 12 Beans	x_{min}	2	0	57	1	2,7	17
	x_{max}	8	2	100	15	62,4	403
	$\bar{x}_{BDK}$	2,91	0,92	77	4,66	29,9	101,9
Standard Library 5 Beans	x_{min}	-	0	80	-	-	-
	x_{max}	-	0	90	-	-	-
	$\bar{x}_{Standard}$	3	0	84	-	-	-
Prototypische EJB[17] Entwicklung 6 Beans	x_{min}	3	0	90	2	4,3	59
	x_{max}	7	0	100	11	24,2	255
	$\bar{x}_{EJB}$	5	0	97	4,66	13,6	118,49
23 Beans	$\frac{1}{23}\sum_{i=1}^{23} x_i$	**3,5**	**0,48**	**86**	**4,66**	**24,4**	**112,8**

Die 23 vermessenen Beans aus drei verschiedenen Applikationsdomainen zeigen unter anderem die folgenden Trends (hier nur ausgewählte Beispiele):

- Es werden keine abstrakten Klassen und nur eine sehr geringe Vererbungstiefe verwendet. Dieses ist zum einem ein Indiz, daß derzeit zumeist sehr technische und weniger fachliche Komponenten entwickelt werden, zum anderen deutet dies auf Schwächen im Design hin.
- Der durchschnittliche Wert der McCabe-Metrik (zyklomatische Zahl eines Programmflußgraphen – hier einer Methode) liegt bei ca. 1,5. In der Tabelle 4 sind jeweils die Maximalwerte der McCabe-Metrik bezogen auf die entsprechenden Methoden einer Klasse dargestellt.
- Mit der „efferenten“ Kopplung (Anzahl von Klassen außerhalb der Packung, von denen die Klassen der Packung abhängen) zeigt sich, daß bei hohen Werten nicht von einer eigenständigen Komponente ausgegangen werden kann.
- Überraschend waren auch der hohe Anteil an als „public“ definierten Methoden. Zwar kann mit der Bean-Info-Klasse diese Sichtbarkeit nochmals eingeschränkt werden, dennoch zeigt sich der Trend, daß auf eine Wartbarkeit bei der Entwicklung wenig Wert gelegt wurde. Dieses gilt sowohl für am Markt verfügbare Bibliotheken, als auch für Eigenentwicklungen.
- Die verwendeten Attribute der 23 Beans wurden fast ausnahmslos als „private“ definiert, was dem Prinzip der Kapselung in der Objektorientierung entspricht und „Seiteneffekte“ verhindern hilft.

[16] BDK – Bean Development Kit
[17] EJB – Enterprise Java Beans

Interessant sind auch die im Zusammenhang mit der prototypischen EJB-Entwicklung verbrauchten Aufwände. Diese liegen deutlich über den bisherigen Erfahrungen bzw. mit Methoden der Aufwandsschätzung (z.B. Object Point) ermittelten Werten. Dabei werden die Ressourcen weniger durch die eigentliche Entwicklung gebunden, als vielmehr durch die vielfältigen Features der verwendeten Produkte (z.B. Applikationsserver Gemstone). Für deren Anwendung und Administration werden ein erheblicher Grad an Einarbeitung benötigt.

4 Zusammenfassung und Ausblick

Die hier vorgestellten Ergebnisse eines Vergleichs verschiedener Komponentenansätze und erster empirischer Untersuchungen stellen einen initialen Schritt dar. Neben den eher die internen Charakteristika von Komponenten beschreibenden Metriken werden auch Aussagen zur Software-Architektur und Anwendungsdomänen benötigt, die ebenfalls starken Einfluß auf die Wiederverwendbarkeit besitzen.

Schwerpunkte weiterer Untersuchungen beziehen sich auf die Identifizierung der „richtigen Größe“ einer Komponente, die Definition notwendiger Prozesse (inklusive Qualitätssicherung) bei der Komposition, die Gewährleistung von QoS-Eigenschaften (inkl. Performanceverhalten) der Komponente selbst oder auch die Berücksichtigung eines hohen domänenspezifischen Wiedererkennungswertes für die einfache Verwendung von Komponenten.

Auf der Basis dieser Untersuchungen und weiteren derzeit durchgeführten empirischen Untersuchungen sollen Programmierstandards (initial für JavaBeans und EJBs) festgelegt werden, die aufzeigen, welche Schwellwerte einzuhalten sind, um „gute“ wiederverwendbare Komponenten zu erstellen.

5 Quellenverzeichnis

[Basili 1988] Basili, V.; Rombach, D.: The TAME-Project: Towards Improvement-Oriented Software Engineering. IEEE Trans. on Software Engineering, 14(1988)6, S. 758 – 773

[Dumke 1998] Dumke, R., Foltin, E., Winkler, A.S.: A Framework for Object-Oriented Software Measurement and Evaluation. Proceedings of the IASTED Conference on Software Engineering, Oct. 28-31, 1998, Las Vegas, S. 129 - 132

[Fenton 1991] Fenton, N.: Software Metrics - A Rigourous Approach. London: Chapmann & Hall Inc., 1991

[Foltin 1999] Foltin, E.; Schmietendorf, A.; Dumke, R.: Requirements and Design of an industrial Metrics Database. CONQUEST'99, Nürnberg, 27- 29. September 1999, S. 145 - 153

[Gamma 1996] Gamma, E.; Helm, R.; Johnson, R.; Vissides, J.: Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software, Addison-Wesley, 1996

[Griffel 1998] Griffel F.: Componentware – Konzepte und Techniken eines Softwareparadigmas, dpunkt-Verlag, Heidelberg 1998

[Schmietendorf 1999] Schmietendorf, A., Dumke, R., Foltin, E., Dimitrov, E., Wipprecht, M.: Conception and Experience of Metrics-Based Software Reuse in Practice, Proceedings of IWSM99, 9th International Workshop on Software Metrics, Sept. 08-09, 1999, Montreal, Canada, pp. 178 – 189