Bachelorarbeit

Realisierung eines web-gestützten service-basierten Ideenmanagement-Systems für Metriken

Realization of a web-supported service-based idea-management-system for metrics

von

Frederic Evers

Vorgelegt der:	Fakultät für Mathematik, Informatik und Naturwissenschaften der Rheinisch-Westfälischen Technischen Hochschule Aachen im März 2010
Angefertigt am:	Lehr- und Forschungsgebiet Informatik 3 Prof. Dr. rer. nat. Horst Lichter
Gutachter:	Prof. Dr. rer. nat. Horst Lichter Prof. Dr. rer. nat. Bernhard Rumpe
Betreuer:	Dipl.-Inform. Matthias Vianden

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Aachen, den 25. März 2010

— Frederic Evers —

Inhaltsverzeichnis

Abbildungsverzeichnis

A. Anhang

Inhalt

A.1. Anhang zu „Grundlagen“

A.1.1. Beispiele für Metriken

Eine bekannte Metrik ist das *CO*nstructive *CO*st *MO*del, kurz COCOMO. Sie wurde von Barry Boehm bei Boeing entwickelt und ist inzwischen auch in der Industrie weit verbreitet. Mit ihr kann man den Aufwand eines Projektes in Personenmonaten abschätzen. Ihre Grundgleichung lautet: $PM = PK \cdot KLOC^{KE} \cdot AM$. PM entspricht dem geschätzten Aufwand in Personenmonaten, der sich durch das Produkt von Produktivitätskoeffizient(PK), 1.000 LOC (KLOC) hoch Komplexitätsexponent (KE) und Aufwandsmultiplikator (AM) berechnet. PK und KE sind Maße für die jeweilige Produktivität der Softwareorganisation und müssen innerhalb der Organisation durch Erfahrungsdaten bestimmt werden. Als mittlere Werte werden 2,94 für PK und 1,1 für KE angegeben. Der Aufwandsmultiplikator stellt ein Produkt aus zwanzig Einflüssen dar und ist bei mittleren Projekten eins. Wichtig bei der Verwendung von COCOMO ist, dass jede Softwareorganisation ihre eigenen Werte ermittelt, um die Schätzungen präziser zu machen. Ansonsten kann es zu starken Fehlschätzungen kommen. So konnte Siemens [vgl. 23] durch die Anpassung der COCOMO-Koeffizienten die Aufwandsschätzungen deutlich verbessern. Mit den Standardkoeffizienten wurden nur 43% der Projektaufwände mit einer Genauigkeit von ± 20% richtig eingeschätzt. Nach der Kalibrierung der Koeffizienten aus den eigenen Erfahrungsdaten lagen 95% im 20%-Schätzbereich. COCOMO ist somit eine berechnete, prognostische, robuste, empirische und objektive Prozessmetrik (vgl. Abschnitt 2.1.1).

Eine bekannte Produktmetrik ist die „Zyklomatische Komplexität“ nach McCabe. Sie soll die Komplexität eines Softwaremoduls anhand einer einzelnen Zahl darstellen. Um sie zu berechnen muss man den Kontrollflussgraphen des Moduls

bestimmen. Der Kontrollflussgraph besteht aus Knoten die die Anweisungen des Codes repräsentieren und gerichteten Kanten, die die Übergänge zwischen diesen darstellen. Die zyklomatische Komplexität (CC) bestimmt sich dann wie folgt: $CC = e - n + p$, wobei e der Anzahl der Kanten, n der Anzahl der Knoten und p der Anzahl der Verbindungen nach außen (bei Single Entry-Single Exit ist $p = 2$) entspricht (siehe Abb. A.1). Auf diese Weise kann man also eine Zahl

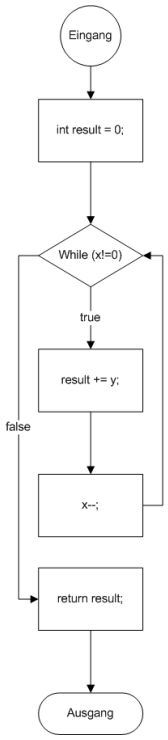

Abbildung A.1.: Beispiel des Kontrollflussgraphen für eine Methode, die die natürlichen Zahlen x und y multipliziert. Es gilt: $e = 5$, $n = 5$ und $p = 2$ und somit ist $CC = 2$.

bestimmen, die die Komplexität wiedergibt und die umso größer wird, je größer die Komplexität zumindest nach diesem Modell ist. Somit handelt es sich bei der zyklomatischen Komplexität um eine modell-basierte, objektive, robuste, diagnostische und berechnete Produktmetrik.

Problematisch ist die Anwendung der CC aber bei objekt-orientierten Systemen, da hier die einzelnen Methoden im besten Fall sehr kurz und wenig komplex sind, so dass die Komplexität nicht unbedingt richtig wiedergegeben wird. Deshalb gibt es auch spezielle OO-Metriken, wie z.B. die Metrik-Suite von Chidamber u. Kemerer [6], die aus mehreren einzelnen Metriken besteht. Dazu gehören z.B. WMC (Weighted Method Complexity) und CBO (Coupling between Objects). WMC ist die Summe der Komplexitäten der Methoden einer Klasse, wobei die Komplexität z.B. durch McCabes CC bestimmt werden kann. Die Kopplung von einem Objekt der Klasse A (CBO) wiederum wird gemessen, indem die Klassen gezählt werden, die A benutzen oder von A benutzt werden. Das

heißt man bestimmt die Anzahl der Klassen, die Methoden von A aufrufen bzw. deren Methoden von A aufgerufen werden, sowie die gegenseitige Benutzung von Instanzvariablen. Dies ist insofern sinnvoll zu bestimmen, da eine niedrige Kopplung anzustreben ist um die Komplexität zu verringern. Mit dieser Metrik können dementsprechend Probleme bei der Kopplung leicht erkannt werden, insbesondere da solche Berechnungen auch durch Werkzeuge übernommen werden können.

A.2. Anhang zu „Implementierung“

A.2.1. Vererbungsstrategien

Bei der *Single Table Per Concrete Entity Class Strategy* wird jede Klasse in einer separaten Tabelle gespeichert, die alle geerbten und neu definierten Attribute der Klasse beinhaltet. Dann sind direkte polymorphe Abfragen nicht mehr möglich. Die dritte Strategie baut, wie der Name bereits sagt, auf Join-Tables. So werden in den Tabellen der Unterklassen nur die neu hinzugekommenen Attribute gespeichert. Hinzu kommt ein Fremdschlüssel auf die Tabelle der Oberklasse, in der die geerbten Attribute zu finden sind. Hier sind polymorphe Abfragen wieder möglich, allerdings sind diese weniger performant, da man immer einen *JOIN* ausführen muss, um die Gesamtheit der Daten zu erhalten. Dies ist insbesondere bei tiefen Vererbungshierarchien problematisch, da für jede Ebene ein JOIN ausgeführt werden muss. Im implementierten Metrikvorschlagssystem ist die Single Table Per Class Hierarchy Strategy also eindeutig die beste Wahl, da sie weder Platz verschwendet, noch Probleme mit polymorphen Abfragen hat und außerdem sehr performant ist.

A.2.2. JPQL

Die Abfragesprache JPQL *Java Persistence Query Language* ist der *SQL* sehr ähnlich, setzt aber auf Objektorientierung. So werden hier nicht Datenbankspalten angegeben, sondern die Attribute der Objekte, deren Mapping auf Datenbankspalten völlig ignoriert wird. Grundsätzlich ist JPQL aber genau wie SQL aufgebaut. In BNF-Syntax sieht ein Select-Ausdruck folgendermaßen aus:

```
select_statement ::= select_clause from_clause
[where_clause] [groupby_clause] [having_clause]
[orderby_clause]
```

Man beginnt also mit einer Select-Klausel gefolgt von einer From-Klausel. Darauf können eine Where-, Group-By-, und Having-Klausel folgen. Die Funktionaltität entspricht also in etwa der Funktionalität von SQL, denn es können beispielsweise auch Unterabfragen gebildet werden. Sollte man die Abfragen aber

weiter optimieren wollen und nicht auf die Konvertierung der JPQL-Abfragen in SQL-Abfragen setzen wollen, so sind auch weiterhin native SQL-Abfragen möglich. Diese wurden im Metrikvorschlagssystem aber nicht eingesetzt.

Diese JPQL-Abfrage, liefert alle Metrikvorschläge, deren aktueller Status „Neu“ ist:

```
SELECT m FROM MetricProposal m, ProposalVersion p
    WHERE p MEMBER OF m.versions AND
    p.status = rwth.swc.ba.ejb3.enums.ProposalStatus.
       NEW AND p.versionnumber =
        (SELECT MAX(v.versionnumber) FROM m.versions v)
```

Wie man hier sieht, folgen die Abfragen nicht der Datenbankstruktur, sondern dem Domänenmodell. Denn im Domänemodell besitzt ein Metrikvorschlag das Attribut „versions“, was einer Liste von Vorschlagsversionen entspricht. Dieses wird hier auch über `m.versions` auch abgefragt. Es wird also nicht über die Fremdschlüssel der Join-Tabelle *Proposal_ Version* gearbeitet. Dies ist im Sinne des Seperation-of-Concerns-Prinzips auch wünschenswert, da man so die Datenbankrepräsentation ändern kann, ohne dass sich die JPQL-Abfragen ändern müssen. Wird beispielsweise die unidirektionale Eins-zu-Viele-Beziehung sinnvoll unterstützt, so dass man die Datenbank anpassen kann, muss man die JPQL-Abfrage nicht ändern. Außerdem sieht man, dass auch die angesprochenen geschachtelten Abfragen möglich sind. Auch Aggregatfunktionen wie *MAX, AVG*, etc. sind möglich.

A.3. Anhang zu „Evaluation“

A.3.1. Testfälle

Dieser Abschnitt enthält die im Unit-Test getesteten Testfälle.

Testfälle für den MetricProposalController

geprüfte Methode	Eingabe	Resultat
storeMetricProposal	korrekte Eingabe	korrekt
	keine E-Mail-Adresse angegeben	Exception
	E-Mail-Adresse nicht im RFC-822-Format	Exception
	E-Mail-Adresse ist nicht in DB vorhanden	Exception
submitMetricProposal	korrekte proposalNumber im Zustand DRAFT	korrekt

	korrekte proposalNumber im Zustand READY_FOR_PROCESSING	korrekt
	korrekte proposalNumber im Zustand PROCESSING	korrekt
	korrekte proposalNumber mit falschem Zustand	Exception
	nicht vorhandene proposalNumber	Exception
addExpert	gültige proposalNumber und E-Mail-Adresse im Zustand NEW	korrekt
	gültige proposalNumber und ungültige E-Mail-Adresse im Zustand NEW	Exception
	gültige proposalNumber und gültige E-Mail-Adresse, aber nicht im Zustand NEW	Exception
	ungültige proposalNumber mit gültiger E-Mail-Adresse	Exception
	ungültige proposalNumber und ungültige E-Mail-Adresse	Exception
changeMetricProposal	gültige proposalNumber	korrekt
	nicht vorhandene proposalNumber	Exception
	gültige proposalNumber, aber nicht im gütligen Zustand	Exception
getMetricProposal-ByProposalNumber	gültige proposalNumber	korrekt
	nicht vorhandene proposalNumber	Exception
rejectMetricProposal	gültige proposalNumber und im Zustand READY_FOR_PROCESSING oder PROCESSING	korrekt
	gültige proposalNumber und in ungültigem Zustand	Exception
	nicht vorhandene proposalNumber	Exception
setExpert	gültige proposalNumber und E-Mail-Adresse	korrekt
	nicht vorhandene proposalNumber und gültige E-Mail-Adresse	Exception
	gültige proposalNumber und leere E-Mail-Adresse	Exception
	gültige proposalNumber und E-Mail-Adresse nicht nach RFC-822-Format	Exception
	gültige proposalNumber und E-Mail-Adresse eines Kunden	Exception
getMetricProposals-ByClient	gültige E-Mail-Adresse	korrekt
	leere E-Mail-Adresse	Exception
	E-Mail-Adresse nicht in RFC-822-Format	Exception
	E-Mail-Adresse nicht in DB	Exception
getMetricProposals-ByExpert	gültige E-Mail-Adresse	korrekt

leere E-Mail-Adresse	Exception
E-Mail-Adresse nicht in RFC-822-Format	Exception
E-Mail-Adresse nicht in DB	Exception
E-Mail-Adresse eines Kunden	null

Tabelle A.1.: MetricProposalController-Testfälle

Testfälle für den UserController

geprüfte Methode	Eingabe	Resultat
storeClient	korrekte Eingabe	korrekt
	kein RFC-822-Format	Exception
	E-Mail-Adresse leer	Exception
	kein Name angegeben	Exception
changeClient	korrekte Eingabe	korrekt
	keine alte E-Mail-Adresse angegeben	Exception
	alte E-Mail-Adresse nicht im RFC-822-Format	Exception
	alte E-Mail-Adresse nicht in der DB	Exception
	neue E-Mail-Adresse nicht im RFC-822-Format	Exception
	keine neue E-Mail-Adresse angegeben	Exception
	neue E-Mail-Adresse ist entspricht E-Mail-Adresse, die bereits in der DB ist und ist ungleich der alten E-Mail-Adresse	Exception
	kein Name angegeben	Exception
	alle Parameter leer	Exception
getClientByE-Mail	korrekte Eingabe	korrekt
	keine E-Mail-Adresse angegeben	Exception
	E-Mail-Adresse nicht im RFC-822-Format	Exception
	E-Mail-Adresse, die nicht in der DB existiert	Exception
getExpertByE-Mail	korrekte Eingabe	korrekt
	E-Mail-Adresse eines Kunden	Exception
	E-Mail-Adresse leer	Exception
	E-Mail-Adresse nicht im RFC-822-Format	Exception
	E-Mail-Adresse, die nicht in der DB existiert	Exception
getEveryExpertE-MailAddress		korrekt

Tabelle A.2.: UserController-Testfälle

Testfälle für den RatingController

geprüfte Methode	Eingabe	Resultat
storeRating	korrekte Eingabe	korrekt
	kein gültiger RatingValue	Exception
	RatingValue leer	Exception
	proposalNumber existiert nicht	Exception
	Metrikvorschlag mit angegebener proposalNumber ist nicht im Status READY_FOR_RATING	Exception
	alle Parameter leer	Exception
getRating	korrekte Eingabe	korrekt
	angegebene proposalNumber existiert nicht	Exception
	aktuelle Version des Metrikvorschlags mit angegebener proposalNumber hat keine Bewertung	Exception
getRatingValues		korrekt

Tabelle A.3.: RatingController-Testfälle

A.4. Anhang zu „Ausblick und Zusammenfassung“

A.4.1. Verwendung der Interzeptoren

Man kann einzelne Methoden oder auch ganze Beans mit `@Interceptors(Interceptor.class)` annotieren, so dass die angegebenen Interzeptoren bei Aufrufen der Methoden reagieren. Sollte man eine ganze Bean annotieren, so kann man aber auch mit `@ExcludeClassInterceptors` bestimmte Methoden davon ausschließen. In der Interzeptorklasse werden Interzeptormethoden mit `@AroundInvoke` annotiert. Die Methoden müssen dabei als einzigen Parameter einen `InvocationContext` anbieten. Der InvocationContext enthält alle wichtigen Daten, die man braucht, um die Details des Methodenaufrufs zu kennen. So bietet der InvocationContext beispielsweise die Methoden `getParameters()` und `getMethod()` an. Außerdem kann der Interceptor auch die Daten mit `setParameters()` verändern. Der Nutzen im Metrikvorschlagssystem wäre also das Umschließen der fachlichen Methoden, wie `submitMetricProposal()`, um zu prüfen, ob der Aufrufende die Berechtigung dazu hat. Dazu wäre ein richtiger Einloggvorgang sinnvoll, der auch ein Passwort umfasst. Im Interzeptor würde dann geprüft, ob der Aufrufer der Methode eingeloggt ist und außerdem über die Berechtigung dieses Methodenaufrufs verfügt.

Über den Sessioncontext kann dann per `isCallerInRole(String roleName)`-Methode, geprüft werden, ob der Aufrufer Mitglied der geforderten Rolle ist.

Wie bei EJB 3.0 üblich, können die Rollen per Annotation benutzt werden. So können mit `@RolesAllowed(String [] value)` Enterprise Beans annotiert werden, so dass nur noch die angegebenen Rollen Zugriff auf die Methoden der Bean erhalten. Natürlich kann man die `@RolesAllowed`-Annotation auch auf Methodenebene verwenden, um so unterschiedliche Zugriffsrechte auf unterschiedliche Methoden zu definieren. Auch eine Kombination aus Klassen- und Methodenannotation ist erlaubt, wobei die Methodenannotation stärker ist.

Literaturverzeichnis

[1] Albrecht, A.J.: Measuring application development productivity. In: *Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium, Monterey, California* (1979), S. 83–92

[2] Basili, V.R.: Software modeling and measurement: the Goal/Question/-Metric paradigm. (1992)

[3] Basili, V.R. ; Caldiera, G. ; Rombach, H.D.: The goal question metric approach. In: *Encyclopedia of software engineering* 1 (1994), S. 528–532

[4] Bergsten, H.: *JavaServer pages.* O'Reilly & Associates, Inc. Sebastopol, CA, USA, 2002

[5] Binder, R.: *Testing object-oriented systems: models, patterns, and tool.* Addison-Wesley Professional, 1999

[6] Chidamber, S.R. ; Kemerer, C.F.: A metrics suite for object oriented design. In: *IEEE Transactions on software engineering* 20 (1994), Nr. 6, S. 476–493

[7] Comsic: *Home Page.* http://www.cosmicon.com/, März 2010

[8] Crocker, D. u. a.: *Standard for the format of ARPA Internet text messages.* 1982

[9] Fenton, Norman E. ; Pfleeger, Shari L.: *Software Metrics: A Rigorous and Practical Approach.* Boston, MA, USA : PWS Publishing Co., 1998. – ISBN 0534954251

[10] Fowler, Martin: *POJO.* http://www.martinfowler.com/bliki/POJO.html, März 2010

[11] Geiger, Walter: Vorstudie zur QZ-Kolumne Januar 2004 zum Begriff Qualitätsziel. (2003), November

[12] Hunter, J. ; Crawford, W.: *Java servlet programming.* O'Reilly & Associates, Inc. Sebastopol, CA, USA, 1998

[13] IBM: *Migrating legacy Hibernate applications to OpenJPA and EJB 3.0.* http://www.ibm.com/developerworks/websphere/techjournal/0708_vines/0708_vines.html (OpenJPA conventions), März 2010

[14] IBM: *PK93265: JAVAX.XML.BIND.JAXBEXCEPTION: [LMY.BEAN; IS NOT KNOWN TO THIS CONTEXT] ERROR OCCURS WHILE INVOKING A JAX-WS WEB SERVICE.* http://www-01.ibm.com/support/docview.wss?uid=swg1PK93265, März 2010

[15] Ihns, O. ; Harbeck, D. ; Heldt, S. ; Koschek, H.: *EJB 3 professionell.* dpunkt.verlag GmbH, 2007. – ISBN 978–3–89864–421–0

[16] ISO/IEC: Systems and software engineering - Measurement process (ISO/IEC 15939:2007(E)). (2007)

[17] Kneuper, R.: *CMMI: Verbesserung von Softwareprozessen mit Capability Maturity Model Integration.* dpunkt-Verl., 2006

[18] Ludewig, Jochen ; Lichter, Horst: *Software Engineerung - Grundlagen, Menschen, Prozesse, Techniken.* dpunkt.verlag GmbH, 2007. – ISBN 3–89864–268–2

[19] Mens, K. ; Lopes, C. ; Tekinerdogan, B. ; Kiczales, G.: Aspect-oriented programming. In: *Lecture Notes in Computer Science* (1998), S. 483–496

[20] O'Malley, CE ; Draper, SW ; Riley, MS: Constructive interaction: A method for studying human-computer-human interaction. In: *Proceedings of IFIP Interact* Bd. 84, 1984, S. 269–274

[21] Ryser, J. ; Glinz, M.: A practical approach to validating and testing software systems using scenarios. In: *QWE 1999, 3rd International Software Quality Week Europe* (1999)

[22] Sazama, Frank: Goal Attribute Measure - Warum wir das A gegen das Q ausgetauscht haben! (2008)

[23] Seibert, S.: Softwaremessung, quantitative Projektsteuerung und Benchmarking. In: *Projektmanagement, Jg* 14 (2003), S. 26–34

[24] September, A.: IEEE Standard Glossary of Software Engineering Terminology. (1990)

[25] Sermersheim, J.: Lightweight Directory Access Protocol (LDAP): The Protocol. (2006), Juni

[26] Starke, G. ; Tilkov, S.: SOA-Expertenwissen. In: *Methoden, Konzepte und Praxis serviceorientierter Architekturen* (2007)

[27] Sun Microsystems: *Authorization Service (JAAS).* http://java.sun.com/j2se/1.5.0/docs/guide/security/jaas/JAASRefGuide.html, März 2010

[28] The Apache Software Foundation: *OpenEJB.* http://openejb.apache.org/, März 2010

[29] Vianden, Matthias ; Hoffmann, Veit ; Lichter, Horst ; Neumann, Karl-Joachim: Ein metamodell basierter Ansatz zur Metrikdefinition und Dokumentation. (2010)

[30] White, S.A.: Introduction to BPMN. In: *IBM Cooperation* (2004), S. 2008–029