Überwachung der Qualität der Architektur einer Software-Produktlinie am Beispiel eines web-basierten Wertpapierinformationssystems*

Martin Verlage[1], Jean-Francois Girard[2], Dharmalingam Ganesan[2], Thomas Kiesgen[1]

[1]MARKET MAKER Software AG
Karl-Marx-Strasse 13
67655 Kaiserslautern
m.verlage@market-maker.de
t.kiesgen@market-maker.de

[2]Fraunhofer IESE
Sauerwiesen 6
67661 Kaiserslautern
girard@iese.fraunhofer.de
ganesan@iese.fraunhofer.de

Abstract: Die kontinuierliche Modifikation und Erweiterung von Software-Produktlinien führt auch zu beabsichtigten und unbeabsichtigten Änderungen an der (Referenz-)Architektur. Wichtige Architekturmerkmale müssen kontrolliert werden, damit die Software nicht degeneriert. Es wird über Erfahrungen bei der Installation und Durchführung eines Prozesses der Architekturüberwachung in einem Wertpapierinformationssystem anhand von Metriken über einen Zeitraum von mehr als drei Jahren berichtet. Klassen mit hohen Messwerten bei einzelnen Metriken und starken Veränderungen dieser Werte werden einer Analyse durch Entwickler, Designer und Manager unterzogen. So werden Probleme in der Software frühzeitig identifiziert.

1 Einleitung

Produktlinien-Engineering ist ein Ansatz zur Entwicklung von Familien von Software-Produkten, der auf Wiederverwendung „großer" Komponenten („reuse in the large") basiert [CN02]. Es entsteht eine generische Rahmenarchitektur sowie eine Menge strategischer Komponenten („assets"). Für das Ableiten einer Produktinstanz (oder „Produkt") werden Komponenten konfiguriert, gegebenenfalls erweitert und integriert. Idealerweise sind die Änderungen minimal. In der Forschung genießt die frühe Phase einer Produktlinie, das „Scoping" zur Identifikation produktlinien-weiter Anforderungen und generischer Eigenschaften, einige Aufmerksamkeit. Von der Qualität der Durchführung des Scoping hängt es ab, wie stark die Eingriffe in den späteren Phasen sein werden [SV02]. Man wird aber nie vermeiden können, dass Änderungen notwendig sein werden. Die zentrale Frage lautet, wie man die Güte der Architektur einer Produktlinie bewahren kann, wenn sie über einen längeren Zeitraum verändert wird.

Eine Änderung kann aus verschiedenen Gründen veranlasst werden. Neue Anforderungen werden bekannt, Rahmenbedingungen ändern sich, Optimierungen

* Diese Arbeit wurde unterstützt durch das Ministerium für Bildung und Forschung im Rahmen des Eureka 2023 Program, ITEA project ip000004 CAFÉ (From Concepts to Application in System Family Engineering).

sollen erfolgen oder Fehler müssen korrigiert werden. Der wesentliche Unterschied zwischen einer „normalen“ Software-Entwicklung und der Fortschreibung einer Produktlinie ist, dass veränderte generische Komponenten kompatibel zu früher abgeleiteten Produktinstanzen bleiben müssen, wenn man ein Auseinanderdriften der Produktlinie vermeiden will. Für die Beherrschung dieses Problems ist sowohl eine aufmerksame Planung als auch die Kontrolle der durchgeführten Änderungen notwendig. In diesem Papier konzentrieren wir uns auf die Organisation, Durchführung und Werkzeuge solcher Kontrollen und erweitern die Erfahrungen aus [GV+04]. Die betrachtete Applikation ist ein web-basiertes Wertpapier-Informationssystem. Objekt-orientierte Metriken für Kopplung, Größe und Komplexität wurden ausgewählt. Die Überwachung ist ein Prozess der gemeinsamen Bewertung der Messwerte durch Software-Entwickler, System-Designer und Manager. In den Fokus geraten solche Klassen und Systemteile, die Extremwerte bei einzelnen Metriken besitzen oder bei denen sich der Wert einer Metrik von einer Version zur anderen stark geändert hat. Die zentrale Frage ist: wird der hohe Metrikwert durch die Natur der Klasse bedingt oder zeigt er einen Problembereich auf, der zukünftig bereinigt werden muss?

Dieses Papier beschreibt die Überwachung der Weiterentwicklung (Evolution) einer Produktlinie. Hierzu sind in Abschnitt 2 der Kontext sowie die detaillierte Zielsetzung der Messungen beschrieben. In Abschnitt 3 wird der Überwachungsprozess erläutert und in Abschnitt 4 an Beispielen illustriert. Erfahrungen mit dem Ansatz sind in Abschnitt 5 beschrieben. In Abschnitt 6 werden die Ergebnisse mit anderen Arbeiten verglichen und schließlich fasst Abschnitt 7 die wesentlichen Ergebnisse zusammen.

2 Kontext und Ziele

2.1 MARKET MAKER Software AG

Die MARKET MAKER Software AG mit über 60 Mitarbeitern spezialisierte sich seit seiner Gründung im Jahr 1990 auf Börsensoftware und zugehörige Datendienste. Informationen zu den Finanzmärkten werden von zahlreichen Quellen bezogen und in Datenbanken abgelegt; die Datenbestände werden analysiert und veredelt und schließlich den Kunden für einen Datenabruf über eigene Desktop-Software oder Internet-Applikationen zur Verfügung gestellt. Im Jahr 1999 beschloss das Management den Aufbau einer neuen Produktlinie, bei der die Nutzung von Internet-Technologie ein wesentliches Element ist [CN02], [VK05]. Diese Produktlinie wurde früh durch die Anwendung der Methode PuLSE definiert [AB+00]. Die in Java 2 entwickelte Produktlinie wird heute unter dem Namen i*ProductLine beworben. Einige wenige Komponenten basieren auf C++, Delphi und PHP. Die Produktlinie besteht aus mehreren hunderttausend Zeilen Code und mehr als 1500 Klassen. Im Jahr 2000 wurden die ersten Dienste ausgeliefert und heute umfasst die Produktlinie unter anderem die folgenden Bausteine: 1) WIP, eine Plattform für kunden-spezifische Bereiche zur Integration in Web-Sites. Es werden Börsendaten und Nachrichten dargestellt, 2) INFO-AGENT, ein web-basierter Informationsdienst für Mitarbeiter in Banken zur Unterstützung in der Wertpapierberatung oder Recherche von Wertpapierstammdaten und Kursdaten,

3) XMLmarket, eine umfangreiche Schnittstelle, welche von Banken für die Bewertung von Portfolios und den Bezug von Börsendaten für Brokerage-Anwendungen genutzt wird, sowie 4) VV-Tool, ein Service zur Extraktion von Portfolio-Daten aus einem weiteren Produkt von MARKET MAKER, welche dann auf einem Web-Server verschlüsselt abgelegt werden und von berechtigten Personen per HTTPS angesehen werden können. Weitere Anwendungen, zum Teil Individuallösungen für Kunden, sind ebenfalls Produktinstanzen der i*ProductLine. Änderungen, die nicht lokal im Code adressiert werden konnten, waren zum Beispiel die Einführung des standardisierten Schlüssels ISIN (International Stock Identification Number) oder die so genannten „penny stocks“, d.h. Wertpapiere die in Einheiten von 1/10 Cent gehandelt werden können. Ferner gab es interne Maßnahmen, wie etwa ein Refactoring zur Verbesserung von Systemteilen sowie die Verwendung neuer Frameworks.

Trotz eines intensiven Scopings konnten Änderungen an der Software nicht ausgeschlossen werden. Es ist sogar so, dass bei Produktlinien durch ihren vielfältigen Einsatz Änderungen an der Software wahrscheinlicher werden sind als bei traditionellen Produkten, andererseits aber besondere Maßnahmen zur Aufrechterhaltung der Software-Qualität durch ihre Existenz als Produktlinie gerechtfertigt sind.

2.2 Ziele der Messungen

Die Überwachung der Weiterentwicklung der i*ProductLine findet aus den folgenden Gründen statt:

- Initiieren und unterstützen von Diskussionen im Team und zwischen Team und Management. Vor allem die Teile, an denen in der letzten Zeit gearbeitet wurde, d.h. Komponenten, die Änderungen unterworfen wurden, bedürfen einer Betrachtung.

- Beantworten der Frage: welche Klassen werden wahrscheinlicher als andere Probleme bereiten oder welche Probleme sind schwerer zu beheben?

- Kontrollieren der Abhängigkeiten von System-Packages.

- Kontrollieren des Erfolgs vorher vereinbarter Maßnahmen, wie zum Beispiel das Refactoring einer Komponente

2.3 Verwendete Werkzeuge

Für die Messungen wurden zwei Werkzeuge verwendet. Zum einen ist dies das M-System des FhG IESE, welches auf der Parser-Technologie FAST basiert, das wiederum ein Bestandteil der Suite Concerto2/AUDIT der SEMA Gruppe ist. Mit dem M-System werden circa 60 zum Großteil allgemein bekannte objekt-orientierte Maße berechnet. M-System lief nur auf Solaris und die Aufbereitung der Daten für die Interpretation durch die Experten dauerte mehrere Tage. Basierend auf den Erfahrungen fand die Entwicklung eines zweiten Werkzeugs mit dem Namen M-Track statt, welches nun das Parsen von Java mittels der Komponente „Bauhaus Java Extractor“ durchführt. Die

Werkzeuge besitzen Parser, die Fakten aus dem zu analysierenden Code extrahieren; auf diesen Fakten werden dann die Metriken berechnet. Die Parser mussten angepasst werden, um bestimmte „benutzt“-Beziehungen zwischen Code-Teilen zu extrahieren.

3. Der Überwachungsprozess

Die Kontrolle der Güte der Rahmenarchitektur und die der Komponenten werden bei MARKET MAKER mittlerweile durch einen Überwachungsprozess gewährleistet, der in den vergangenen Jahren initiiert und optimiert wurde. Dieser Abschnitt beschreibt die wesentlichen Elemente des aktuellen Zustands (siehe auch [VK05]).

3.1 Rollen

Für die Durchführung des Prozesses sind fünf Rollen notwendig:

- *Entwickler* und *System-Designer* sind die Fachleute für die Software; sie geben Detailauskünfte zu Fragen und diskutieren Alternativen für Änderungen.

- Ein *Produktverantwortlicher* nimmt strategische Blickwinkel ein und ist für kaufmännische Aspekte in den Diskussionen zuständig. Er wägt ab, ob Änderungen unter Berücksichtigung von Auftragslage und Budgets durchgeführt werden können.

- Ein *Repräsentant* ist ein erfahrener Entwickler mit einem guten Verständnis vom Gesamtsystem; er steht für Detailfragen im Vorfeld von Workshops zur Verfügung.

- Der *Metrik-Spezialist* ist für die fachliche Betreuung der Messungen zuständig. Dies beinhaltet auch die Definition von Metriken anhand von Qualitätsattributen, die Entwickler und Produktverantwortliche als relevant erachten. Er stellt die Datensätze inklusive grafischer Darstellungen für die Workshops zusammen.

- Der *Moderator* ist für die Vorbereitung, Durchführung und Nachbereitung der Workshops zuständig.

3.2 Prozessschritte

Der Prozess der Überwachung besteht aus sechs Schritten: Das Ziel des ersten Schrittes ist die Dokumentation der Anfangssituation des Systems (Schnappschuss) auf der Basis einer initialen Menge von Metriken. Der Schnappschuss dieses ersten Schrittes dient als Referenzpunkt für spätere Bewertungen und ermöglicht eine erste Analyse der Metriken auf Eignung für den beabsichtigten Zweck.

Die Menge von 60 möglichen Metriken ist zu groß, um sie ständig einzusetzen. Da die Metriken meistens noch auf die Programmiersprache oder die Besonderheiten des Systems angepasst werden müssen und nur begrenzte Kapazitäten dafür zur Verfügung

stehen, ist es unserer Erfahrung nach sinnvoll eine Menge von 5-10 tatsächlich zum Einsatz kommenden Metriken zu bestimmen (siehe Abschnitt 3.3 unten).

Im dritten Schritt werden die Metriken auf der aktuellen Code-Version berechnet. Der Zeitpunkt der Messwerterhebung wird durch den Produktverantwortlichen vorgegeben, da dieser Schritt mit einigem Aufwand verbunden ist. So sollten sich aktuell keine größeren Änderungen in der Umsetzung befinden und das Team der Meinung sein, dass eine Messwerterhebung Sinn macht. Der Metrik-Spezialist bestimmt die Metriken der aktuellen Version (Snapshot) sowie einen Vergleich zu früheren Versionen; diese Delta-Betrachtung beinhaltet nicht nur Aussagen in der Veränderung der Metriken sondern auch Veränderungen im Code basierend auf dem UNIX-Werkzeug *diff*.

Direkt nach der Messwerterhebung werden die vorläufigen Daten gemeinsam vom Metrik-Experten und den Repräsentanten begutachtet. Hierbei werden Ausreißer in den Daten besonders betrachtet. Diese können entweder auf Messfehlern beruhen oder Besonderheiten hervorheben. Unter Umständen kann es bei der Analyse zu neuen Fragestellungen kommen, die noch vor dem Workshop vom Metrik-Spezialisten beantwortet werden sollten. Dieser Schritt hat sich als sehr wichtig herausgestellt, um das Vertrauen des Teams in die Daten nicht zu erschüttern, einfache Fragestellungen vorweg zu nehmen und die Thematik für den nächsten Schritt einzugrenzen, so dass der Workshop effizient durchgeführt werden kann. Am Ende dieses Schrittes steht die Festlegung auf eine Menge zu diskutierender Klassen.

Der fünfte Schritt besteht aus einem Workshop mit allen Beteiligten. Hier werden Ursache und Wirkungen für Probleme bei den ausgewählten Klassen auf Basis der Messwerte identifiziert (in Abschnitt 4 wird dies genauer beschrieben). Maßnahmen für die nächsten Wochen und Monate, wie etwa das Refactoring einzelner Komponenten, werden festgelegt. Da die Qualität der Komponenten aus den Metriken ablesbar oder zumindest ableitbar ist, ist es nicht problematisch, Budgets für solche Maßnahmen zu rechtfertigen. Manchmal kann es in den Workshops dazu kommen, dass die Metriken verändert oder erweitert werden, um sie präziser zu machen.

Der sechste Schritt stellt eine Nachbereitung dar. Die Evolution der Produktlinie soll dokumentiert werden. Außerdem wird hier M-Track optimiert und die Definition der Metriken sowie die Aufbereitung der Daten für die Präsentation gelegentlich angepasst.

3.3 Auswahl einiger weniger Metriken

Wie für den zweiten Schritt beschrieben wird nach einer minimalen Menge von Metriken gesucht, die am besten die Varianzen in den Messwerten erläutert. Stehen ähnlich gute Mengen zur Verfügung, so werden auch andere Faktoren berücksichtigt, wie etwa die Einfachheit der Messung oder die Eingängigkeit der Interpretation. Die Auswahl basiert auf der „principal component analysis“ (PCA) [Dun89]. Die PCA ist ein standardisiertes Verfahren zur Bestimmung grundlegender orthogonaler Dimensionen, die Beziehungen zwischen den Variablen einer Datenmenge beschreibt. In unserem Fall gruppiert PCA Metriken zu „Cluster“ bei denen die Metriken zueinander am stärksten miteinander in Beziehung stehen, die wahrscheinlich dasselbe Phänomen beschreiben.

Der erste Cluster beschreibt am stärksten die Varianz in der Datenmenge; ein zweiter Cluster beschreibt am besten die verbliebene Varianz und so weiter. Zusätzlich wird in jedem Cluster die Teilmenge der Metriken identifiziert, die am besten orthogonal zu den anderen Clustern sind, also unterschiedliche Konzepte beschreiben. Somit wird durch Anwendung der PCA auf wenige Cluster eine kleine Menge an Metriken identifiziert, die am besten die Varianz in der Datenmenge beschreibt. Principal Components (PCs) sind Linearkombinationen von Variablen. Die Summe der Quadrate aller Koeffizienten in jeder Linearkombination ist gleich dem Wert 1. PCs werden wie folgt berechnet: Die erste PC ist die Linearkombination aller standardisierten Variablen, die ein Maximum an Varianz in der Datenmenge erläutern. Die zweite und die nachfolgenden PCs sind Linearkombinationen alle standardisierter Variablen, wobei jede neue PC orthogonal zu allen vorher berechneten PCs ist und ein Maximum an Varianz unter diesen Bedingungen beschreibt. Üblicherweise hat nur eine Teilmenge aller Variablen einen großen Koeffizienten – auch „loading“ genannt – und trägt somit signifikant zur Varianz in einer PC bei. Um diese Variablen zu identifizieren und um die PCs zu interpretieren, wurde die „varimax rotation“ angewendet.

Name	Definition	Quelle
CBO	Kopplung zwischen Klassen. Für jede gegebene Klasse C wird die Anzahl an Methodenaufrufen und die Attribute gezählt. Klassen C und D sind gekoppelt, wenn Methoden von C Methoden von D nutzen oder Attribute von D zugreifen, oder umgekehrt.	[CK94]
CBO_IL	CBO - Internal Libraries. Gleich CBO, aber die D ist Bestandteil einer internen Bibliothek	
CBO_EL	CBO - External Libraries. aber die D ist Bestandteil einer externen Bibliothek	
OMMIC	Import-Kopplung zwischen Methoden zweier Klassen. Die Klasse C ist mit der Klasse D gekoppelt, wenn C eine Methode von D aufruft oder einen Methodenparameter vom Typ D besitzt.	[BDM97]
MPC_EL	Message passing coupling - external libraries. Die Anzahl von Methodenaufrufen einer Klasse zu Methoden in externen Bibliotheken.	MPC in [LH93]
OCMEC	Export coupling through class-method interaction. Wie oft wird die Klasse C als ein Methodenparameter in anderen Klassen verwendet.	[BDM97]
OCAEC	Export coupling through class-attribute interaction. Wie oft wird die Klasse C als ein Attribut in einer anderen Klasse verwendet.	[BDM97]
IH-ICP	Inheritance information-flow-based coupling. Die Anzahl von Methodenaufrufen von geerbten Methoden, gewichtet mit der Anzahl Parameter der aufgerufenen Methode.	[LLW95]
NIMP	Anzahl der implementierten Methoden einer Klasse.	
NMINH	Anzahl der geerbten Methoden einer Klasse.	

Tabelle 1: Definitionen der verwendeten Metriken

Durch PCA wurden von den 36 damals implementierten Metriken im M-System sechs identifiziert, die am besten die Varianz in der Datenmenge beschrieben: CBO, OMMIC, MPC, OCMEC, OCAEC und IHICP (siehe Tabelle 1). Hinzu kam noch die Metrik NIMP zur Bestimmung der Größe einer betrachteten Einheit (Klasse, Code-Fragment oder Package), um Messwerte zu normalisieren. Es konnten (noch) keine Metriken ermittelt werden, die ausschließlich oder besonders gut für Produktlinien geeignet sind. Gerade bei den Metriken CBO und OMMIC erweist sich aber eine Eigenheit der Produktlinie als problematisch. Um die Software generisch zu gestalten, werden viele Schnittstellen (Interfaces) und dynamische Instantiierungen von Klassen verwendet. Java bietet mit `Class.forName(<java.util.String>)` eine Methode an, mit der ein Klassenobjekt gesucht wird, welches dann für die Instantiierung verwendet werden kann. Die effektiven Beziehungen zwischen Klassen werden also erst zur Laufzeit hergestellt und können nicht statisch analysiert werden. Die Kombination statischer und dynamischer Architekturanalyse ist zurzeit Gegenstand weiterer Forschungsarbeiten.

4. Nutzung der Metriken

Die sechs Metriken wurden durchgängig erfasst und Messwerte archiviert. In den Workshops werden zuerst Komponenten diskutiert, die bei einer oder mehreren Metriken über Ausreißer verfügen. Als Vorbereitung erstellt der Metrik-Spezialist einfache Listen, die absteigend sortiert die Metriken einzeln für alle Klassen ausweisen. In den ersten Workshops wurde schnell deutlich, dass die Werte normalisiert werden müssen. Hierzu wurde der Einfachheit halber die Größe der Klassen verwendet. Wurden Snapshots miteinander verglichen, so hilft die Klassengröße ebenfalls die Änderungen in einer Metrik zu normalisieren. Diese einfache aber leicht nachvollziehbare Betrachtung wurde dann um die folgende Vorgehensweise ergänzt: die relative Änderung einer Metrik wird in Bezug gesetzt zur Änderung der Größe, um Klassen zu identifizieren bei denen eine kleine Änderung im Code eine hohe Änderung in der Metrik nach sich gezogen hat. Für diese Fragestellung wird ein einfacher zweidimensionaler Plot-Chart verwendet, der auf der einen Achse die relative Änderung im Code und auf der anderen Achse die relative Änderung in den Metrikwerten beinhaltet.

Class (top value)	old Value	Value	delta (%)	lines modified	add	change	del.	% modif	tot Loc
Log	141	333	136,17%	20	1	18	1	3,09%	648
ServletUtils	115	210	82,61%	40	38	2	0	14,49%	276
ServiceManager	76	110	44,74%	4	0	3	1	0,35%	1159
MergerProperty	37	106	186,49%	234	156	77	1	52,47%	446
User	57	90	57,89%	998	798	200	0	88,48%	1128
MergerServlet	20	68	240,00%	592	251	315	26	96,26%	615
MergerData	37	66	78,38%	67	43	24	0	44,37%	151

Tabelle 2: Beispiel einer Datenauswertung für verschiedene Klassen mit CBO

Die Mehrheit der Datenpunkte hat einen niedrigen Wert in der Metrik. Somit erscheint es logisch, dass nur die Extremwerte betrachtet werden, solange keine detaillierten Funktionen für „problematische“ Klassen bereitstehen. Der Fokus auf Extremwerte entspricht der Intuition, dass Klassen mit hohen Werten potentiell problematisch sind. Dies ist aber nicht immer der Fall. Manchmal spiegelt ein hoher Wert exakt die Natur einer Klasse wider. Betrachten wir die Klasse „Log“ in Tabelle 2. Mit Absicht steht die Klasse mit vielen anderen Klassen in Beziehung und hat daher einen hohen Kopplungswert. In anderen Fällen, wie etwa der Klasse „User“ in Tabelle 2 ist ein hoher Kopplungswert nicht erwünscht und es ist Aufgabe des Teams die Beziehungen zu dieser Klasse zu reduzieren, um den Kopplungswert zu verringern.

Je mehr Workshops durchgeführt wurden, desto detaillierter wurden die Fragestellungen. Zum einen, weil grundsätzliche Problemkomplexe beseitigt wurden und man mehr auf die gemessenen Attribute achtete, zum anderen weil die Mächtigkeit und die Grenzen der Messungen besser erkannt wurden. Der Metrik-Spezialist begleitet neue Fragestellungen messtechnisch und fügt diese den Standard-Auswertungen hinzu. Ein Beispiel einer solchen Auswertung ist die Abhängigkeit von externen Bibliotheken (libraries). Die Identifikation von Klassen, die am meisten von Bibliotheken abhängen, sowie die Begrenzung auf einige wenige ist eine zentrale Aufgabe des Entwicklungsteams. Für diesen Zweck wurden drei weitere Metriken definiert: CBO_IL,

CBO_EL und MPC_EL (siehe Tabelle 1). Metriken mit Suffix „_EL“ messen Beziehungen zu externen, firmen-fremden Bibliotheken; Metriken mit Suffix „_IL“ messen Bibliotheken, die im Unternehmen als Asset der Produktlinie entwickelt wurden.

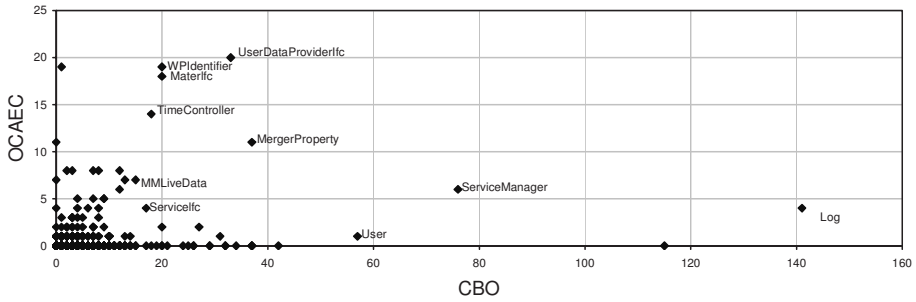

Abbildung 1: Gegenüberstellung zweier Kopplungsmaße

Jede einzelne Metrik beschreibt einen Aspekt einer Klasse. In einigen wenigen Fällen wurde später eine Kombination der Werte mehrerer Metriken betrachtet. Abbildung 1 beschreibt zum Beispiel die Stärke der Kopplung einer Klasse zum Rest des Systems (CBO auf der X-Achse) und die Anzahl der Klassen, die eine bestimmte Klasse als Attribut haben (Y-Achse). Die gleichzeitige Betrachtung der Messwerte beider Metriken erlaubt eine bessere Beurteilung der Gesamtsituation des Systems.

5. Erfahrungen

Dieser Abschnitt fasst die wesentlichen Punkte im Umgang mit den objekt-orientierten Metriken bei der Überwachung der Qualität einer Produktlinie zusammen. Wir möchten betonen, dass diese aus einem bestimmten Kontext stammen und auf einen bestimmten Zeitpunkt bezogen sind. Weder sollte man die Ergebnisse unkritisch übernehmen noch sollte man sie als statisch ansehen.

Nicht die Metriken an sich ergeben den Nutzen aus den Workshops, sondern die durch sie ausgelösten Diskussionen über einzelne Klassen. Die Metriken sind ein Werkzeug, um die Diskussionen zu fokussieren und Kandidaten zu priorisieren. Noch wichtiger ist die Etablierung einer zwischen Entwicklern, Designern und Management gemeinsamen Sicht auf die strukturelle Qualität der Produktlinie. Man unterhält sich darüber, welche Komplexität noch geduldet wird und warum bestimmte Klassen trotzdem Ausreißer darstellen dürfen. Hierdurch werden auch unerfahrene Entwickler geschult.

Die Anzahl der Metriken wurde bewusst niedrig und die Definition der Metriken einfach gestaltet. Wir erwarten erst zukünftig ein akzeptables Verhältnis zwischen Kosten und Nutzen der Datensammlung detaillierter und aufwändiger Metriken. Für den Anfang war die Verwendung einfach zu verstehender und mit geringem Aufwand zu sammelnder Metriken wichtig, anhand derer grundlegende Probleme in der Produktlinie erkennbar sind. Es haben sich drei Filter als hilfreich erwiesen: a) Ausreißer, b) Klassen mit einer hohen Änderung in einer Metrik, die zu den Top-Drei-Werten in anderen Metriken

gehörten, und c) Klassen die zu mehr als 30% geändert wurden. Diese Filter haben recht zuverlässig die Problemzonen in der Produktlinie identifiziert. Der Aufwand für die Definition und Validation von zusätzlichen Metriken ist häufig zu hoch, so dass einige wenige Fragen, die einer detaillierten Analyse bedürfen, offen bleiben müssen.

Beim Start des Metrik-Projekts gab es die Idealvorstellung, Grenzwerte für jede Metrik zu finden, ab denen eine Komponente automatisch als problematisch identifiziert wird. Schnell stellte sich heraus, dass dies nicht machbar war. Manche Ausreißer haben diesen Wert auf Grund ihrer Funktionalität zu Recht; eines der besten Beispiele dafür ist die Klasse Log. Die Unmöglichkeit der Bestimmung von Grenzwerten war schließlich auch der Grund dafür, dass es keinen automatischen Prozess der Bewertung der Qualität der Produktlinie gibt, sondern Interpretationen gemeinsam in Workshops stattfinden.

Die Qualität des Parsers zur Extraktion der Fakten ist wichtig. Kleine Fehler bei der Extraktion der Fakten können in großen Ausschlägen der Metrik resultieren, d.h. das Verfahren ist nicht robust gegenüber Parsing-Fehlern. Daher sollte gerade dieser Teil des Metrik-Systems exzessiv getestet werden. Falsche Daten können das Vertrauen der Entwickler in die Grundlagen der Bewertung zerstören. M-System und M-Track wurden durch das FhG IESE entwickelt und aufgesetzt. Die Algorithmen und Metriken mussten auf die Programmiersprache Java angepasst werden. MARKET MAKER hatte hierfür weder die Kompetenz noch die Kapazitäten. M-System konnte außerhalb des Forschungsprojekts wegen lizenzrechtlicher Bestimmungen einer Komponente nicht eingesetzt werden, was ebenfalls ein Grund für den Wechsel zu M-Track war.

Die initiale Diskussion mit einem Repräsentanten vor einem Workshop (Schritt 4 des Prozesses) ist notwendig. Der Experte kann Ergebnisse in Frage stellen, die nicht seiner Intuition entsprechen und somit Korrekturen oder Detailauswertungen auslösen. Dies führt zu einem produktiveren Workshop und bildet Vertrauen in das Messprogramm. Der Moderator soll die Definitionen jeder Metrik sowie Beispiele dazu am Anfang jeden Workshops vorstellen. Dies verhindert falsche Interpretationen.

Die verwendeten statischen Metriken führten zu keiner zufrieden stellenden Aussage, wie Klassen zu Komponenten gruppiert werden, um z.B. die Kohäsion zu berechnen.

6. Vergleichbare Forschungsarbeiten

Dieser Abschnitt fasst vergleichbare Arbeiten auf dem Gebiet objekt-orientierter Metriken insbesondere zur Bewertung von Software-Änderungen zusammen und vergleicht diese teilweise mit den vorangestellten Aussagen. Hierbei geht es nicht um einen Auseinandersetzung mit den Stärken und Schwächen der Ansätze. Alle Ansätze stellen mehr oder weniger Speziallösungen für die Kontexte, in denen sie entwickelt wurden, dar und belegen so die Notwendigkeit zur Anpassung von Metriksystemen.

Zwei voneinander unabhängige Arbeiten (Burbeck [Bur96] und Vasa und Schneider [VS03]) fokussieren auf Komplexitätsmaße auf Methoden-Ebene. Burbeck benutzt zur Darstellung der Messwerte der Komplexität von Smalltalk-Methoden eine einfache

Ampel-Metapher, die den Zustand der Methode bezüglich der gemessenen Eigenschaft anzeigt: zwei Schwellwerte unterteilen die Skala einer Metrik in drei Bereiche „grün", „gelb" und „rot". Dieses Vorgehen entspricht nicht unserer Erfahrung, dass für Ausnahmesituationen hohe Metrik-Werte keinen kritischen Zustand darstellen. Vasa und Schneider beobachteten die Veränderung der zyklischen Komplexität („cyclomatic complexity") in objekt-orientierter Software. In ihren Fallstudien beobachteten sie die Gültigkeit des ersten und fünften Gesetzes von Lehman, nämlich fortlaufende Veränderung und „conservation of familiarity", auch für objekt-orientierte Systeme. Lanza [Lan01] visualisiert die Evolution eines OO-Systems anhand einer Evolutionsmatrix. In dieser Matrix enthalten die Spalten die Version der Software und jede Zeile repräsentiert die Versionen einer Klasse. Die Elemente der Matrix sind Kästen; die Höhe und Breite der Kästen entsprechen dabei zwei Größenmaßen einer Klasse. Anhand dieser Werte werden Klassen wie folgt kategorisiert: a) Pulsar. Eine Klasse wächst und schrumpft im Laufe ihrer Lebenszeit. b) Supernova. Eine Klasse „explodiert" plötzlich. c) Stagnant. Keine Veränderungen bezüglich der Größe über einen längeren Zeitraum, und d) Persistent. Eine Klasse besteht über den gesamten Zeitraum und ist Element des Designs. Systä et al. [SY+00] berechnen Metriken für Kopplung und Kohäsion für Java-Klassen. Mechanismen für die Navigation durch Hierarchien und Darstellung von Beziehungen mit dem Ansatz Rigi [MWT94], um die Analyse von Ursachen für hohe Kopplungen und niedrige Kohäsion zu untersuchen. Durch die Nutzung dieser Informationen werden die Kandidaten für Re-Engineering bestimmt. Bansiya [Ban00] schlägt einen Ansatz zur Bewertung der strukturellen Eigenschaften eines Systems und die Stabilität von Frameworks während ihrer Evolution vor. Die Methode benutzt objekt-orientierten Entwurfs-Metriken, um einen Wert für den Aspekt „extend-of-change" zwischen zwei Versionen zu bestimmen. Aoyama [Aoy02] definiert Metriken für kontinuierliche und diskontinuierliche Evolutionen von Software. Ein System wird als Baumstruktur definiert und eine Evolution ist eine Abbildung zwischen den Strukturen zweier Versionen. Anhand von Distanz-Metriken wird bestimmt, welche Teile des Systems sich geändert haben und welche stabil geblieben sind. Godfrey und Tu [GT02] verfolgen die strukturelle Evolution mit Metriken, die darauf reagieren, ob eine Komponenten oder ein Element neu in ein System eingeführt wurde, oder ob es nur umbenannt, anders platziert oder sonst wie geändert wurde.

7. Zusammenfassung und Ausblick

Die folgenden Erfahrungen basieren auf einer mehrjährigen Fallstudie zur Überwachung der Evolution eines web-basierten Wertpapierinformationssystems, der i*ProductLine:

(1) Der wesentliche Nutzen der Überwachung sind die Diskussion über die Messwerte, die Analyse der Ursachen für Extremwerte und die Maßnahmen. Indem das Team selbst interpretiert, wird die Akzeptanz für den Messansatz gesteigert und ein Entwicklungsschwerpunkt auf die strukturellen Eigenschaften des Systems gelegt.

(2) Der Aufwand zur Bestimmung einer kleinen Menge an Metriken zur Überwachung, hier die Anwendung der „principal component analysis", ist gut investiert. Die Überwachung schaut fokussiert auf die strukturellen Eigenschaften, die als

wesentlich erkannt wurden und verliert sich nicht in einem hohen Aufwand für die Messwerterfassung.

(3) Es braucht eine gewisse Zeit, bis das Metrik-System eingeschwungen ist. Gerade die Festlegung der zu messenden strukturellen Eigenschaften bedarf einer Fokussierung durch das Entwicklungsteam und muss mit Hintergrundwissen über die Komponenten versorgt werden.

(4) Die Entwicklung von Filtern zur Identifikation von Kandidaten für die Diskussion in den Workshops ist wichtig, um die Anzahl zu reduzieren. Jeder Workshop sollte sich auf einige wesentliche Probleme konzentrieren.

(5) Die Abstände zwischen den Workshops müssen bewusst gewählt werden. Aus unserer Erfahrung ist es gut, vor und nach dem Einbringen einer größeren Funktionsmenge (z.B. Erweiterungen und Änderungen für einen wichtigen Kunden) einen Snapshot zu ziehen, um den Einfluss der Änderung auf die Architektur sichtbar zu machen.

Die Überwachung ist heute ein etablierter Prozess bei MARKET MAKER. Er wird aber ständig erweitert werden, um sich an erweitertes Wissen um die strukturellen Eigenschaften der i*ProductLine anzupassen. Die nächsten geplanten Schritte sind:

- Die Kohäsion wird bisher nicht betrachtet, da sich die Grenzen einer Komponente nicht trivial ableiten lassen, z.B. durch die Grenzen eines Java-Packages.

- Die Anwendung verschiedener Visualisierungstechniken für die Metriken soll in den Workshops erprobt werden, um schnellere Interpretation auch komplexer Zusammenhänge zu ermöglichen.

- Die statischen objekt-orientierten Metriken werden um Messmethoden dynamischer Eigenschaften erweitert. Zukünftig werden also nicht nur Beziehungen zwischen Klassen betrachtet, sondern auch solche zwischen einzelnen Objekten und solche, die erst zur Laufzeit bekannt werden. Hierzu haben Forschungsarbeiten im Kontext der i*ProductLine bereits begonnen.

Danksagung

Die Autoren danken allen Kollegen für die konstruktive und gute Mitarbeit. Die Arbeiten wurden im Rahmen des Projekts CAFÉ als Teil des Eureka Σ!2023 Programme, ITEA, Ip00004, durch das Bundesministerium für Bildung und Forschung gefördert (BMBF 01-IS-002). Den Gutachtern gilt ein besonderer Dank für die wertvollen Kommentare zur Verbesserung der Arbeit.

Literaturverzeichnis

[AB+00] Anastasopoulos, M.; Bayer, J.; Flege, O.; Gacek, C.: A Process for Product Line Architecture Creation and Evaluation – PuLSE-DSSA Version 2.0. Technical Report, No. 038.00/E, Fraunhofer IESE, Juni 2000.

[Aoy02] Aoyama, M.: Metrics and Analysis of Software Architecture Evolution with Discontinuity. In Intl. Workshop on Principles of Software Evolution, 2002.

[Ban00] Bansiya, J.: Evaluating Framework Architecture Structural Stability. In ACM Computing Surveys, Band 32, März 2000.

[BDM97] Briand, L.; Devanbu P.; Melo, W.L.: An Investigation into Coupling Measures for C++. In International Conference on Software Engineering, Boston, USA, April 1997.

[Bur96] Burbeck, S.L.: Real-time complexity metrics for Smalltalk methods. In IBM Systems Journal, Band 35, 1996.

[CK94] Chidamber, S. R.; Kemerer, C. F.: A Metrics Suite for Object Oriented Design. IEEE Transactions on Software Engineering, 20(6):476–493, Juni 1994.

[CN02] Clements, P.; Northrop L.: Software Product Lines – Practices and Patterns. SEI Series in Software Engineering, Addison Wesley, 2002.

[Dun89] Dunteman, G.: Principal Component Analysis. Sage Publication, 1989.

[FAS97] FAST Programmer's Manual, SEMA Group, France, 1997.

[GT02] Godfrey, M.; Tu, Q.: Tracking structural evolution using origin analysis. In Intl Workshop on Principles of Software Evolution, 2002.

[GV+04] Girard, J.-F.; Verlage, M.; Ganesan, D.: Monitoring the Evolution of an OO System with Metrics: An Experience from the Stock Market Software Domain. Intl. Conf. on Software Maintenance 2004, September 2004, Seiten 360-367

[Lan01] Lanza, M.: The Evolution Matrix: Recovering Software Evolution using Software Visualization Techniques. In Intl. Workshop on Principles of Software Evolution, 2001.

[LL+95] Lee, Y.-S.; Liang, B.-S.; Wu, S.-F.; Wang, F.-J.: Measuring the Coupling and Cohesion of an Object-Oriented Program Based on Information Flow. In International Conference on Software Quality, Maribor, Slowenien, 1995.

[LH93] Li, W.; Henry, S.: Object-oriented Metrics that Predict Maintainability. Journal of Systems and Software, 23(2):111–112, 1993.

[MWT94] Müller, M.; Wong, K.; Tilley, S.: Understanding software systems using reverse engineering technology. In The 62nd Congress of L'Association Canadienne Française pour l'Avancement des Sciences Proceedings, 1994.

[SV02] Schmid, K.; Verlage, M.: The Economic Impact of Product Line Adoption and Evolution. IEEE Software 19(4): 50-57 (2002)

[SY+00] Systä, T.; Yu, P.; Müller, H.: Analysing Java Software by combining Metrics and Program Visualization. In Proc. of 4th European Conference on Software Maintenance and Reengineering, 2000.

[VK05] Verlage, M.; Kiesgen, T.: Five Years of Product-Line engineering in a Small Company. Erscheint in: Proc. of the 27th International Conference on Software Engineering, St. Louis, USA, Mai 2005.

[VS03] Vasa, R.; Schneider, J.G.: Evolution of Cyclomatic Complexity in Object-Oriented Software. In 7th Workshop on quantitative approaches in object-oriented software engineering, 2003.