Über die Auswirkungen von Refactoring auf Softwaremetriken

Stefan Burger[1] & Oliver Hummel[2]

[1]Software-Engineering-Gruppe
Universität Mannheim
68131 Mannheim
sburger@mail.uni-mannheim.de

[2]Institut für Programmstrukturen und Datenorganisation
Karlsruher Institut für Technologie (KIT)
76128 Karlsruhe
hummel@kit.edu

Abstract: Softwaremetriken wurden in der Vergangenheit bereits vielfach diskutiert und auch kritisiert, werden aber in der industriellen Praxis nach wie vor als einfach zu erhebende Indikatoren für Codequalität oder gar die Qualität einer Software selbst angewendet. Gemessene Werte und Grenzwerte sind dabei oft schwierig zu interpretieren bzw. willkürlich festgelegt, so dass ihr Nutzen nach wie vor in Frage gestellt werden muss. Dieser Beitrag untermauert die bestehenden Kritiken an Softwaremetriken, indem er zunächst beispielhaft zeigt, wie Messergebnisse gängiger Metriken mit schlechter lesbarem aber funktional identischem Code künstlich „verbessert" werden können. Darauf aufbauend präsentiert er erste Analyseergebnisse (von Fowlers „Video-Store" und dem Lucene-Open-Source-Projekt), die darlegen, dass bekannte Softwaremetriken Code-Verbesserungen durch Refactoring nicht erkennen können, bzw. umgekehrt betrachtet, Refactoring die Codequalität verschlechtert. Ferner zeigen sich gängige Qualitätsmodelle ebenfalls weitgehend unempfindlich für die durchgeführten Refactorings.

1. Einleitung

Eine weit verbreitete Methode zur Messung der Wartbarkeit von Software wird unter dem Begriff statische Code-Analyse zusammengefasst [Ho08] und untersucht Quellcode mit Hilfe von Softwaremetriken, um mögliche Schwachstellen in dessen Qualität, wie beispielsweise eine hohe Komplexität, zu identifizieren. Bekannte, vor allem zur Analyse der Wartbarkeit und ihrer Untereigenschaften wie Verständlichkeit und Änderbarkeit [IO11] bereits seit Jahrzehnten genutzte Softwaremetriken sind beispielsweise McCabe's zyklomatische Komplexität [MC76], Halsteads Software Science Metrics [Ha77], oder auch jüngere objektorientierte Metriken, wie etwa die Kopplung zwischen Klassen [CK94]. Dabei gilt bisher vereinfacht gesagt meist die Grundannahme, dass „einfacherer" Code niedrigere Metrikmesswerte haben sollte (vgl. [Ha77, Mc76]), da kürzerer und somit weniger komplexer Code offensichtlich leichter zu verstehen ist [SA05, MR07]. Obwohl in der Literatur verschiedene, auf Metriken basierende

Qualitätsmodelle existieren (z.B. [PR10, AYV10]), sind direkte Nachweise einer Korrelation zwischen Metriken und Code- oder gar Softwarequalität bis dato nicht sehr zahlreich und konnten hauptsächlich für die Fehlerhäufigkeit gezeigt werden [NBZ06, SZT06]. Trotz dieser weitgehend fehlenden theoretischen Untermauerung werden in der Industrie oftmals Metrikwerte für Systeme definiert, die Auftragnehmer zu erreichen haben: zu liefernde Gewerke werden entsprechend auf die Einhaltung dieser Grenzwerte hin optimiert, in der Hoffnung damit auch die Systemqualität insgesamt zu verbessern. Es bleibt aber nach wie vor unklar, ob klassische Softwaremetriken tatsächlich in der Lage sind, gutes, d.h. verständliches und änderbares Softwaredesign zu erfassen. So fand Seng [Se07] bereits vor einigen Jahren heraus, dass eine automatisierte Code-Optimierung auf Basis von Metriken höherwertige Strukturen wie Design Patterns [GHJ10] zerstören kann. Auch vorangegangene Arbeiten der Autoren zur Analyse der Kabinensoftware eines Airbus-Flugzeugs mit Hilfe von Softwaremetriken [BH12] blieben im Hinblick auf Codequalität weitgehend aussagelos, da die verwendeten Metriken nicht in der Lage waren, hohe Domänenkomplexität von schlechter Programmierung zu unterscheiden.

Refactoring [Fo99] ist im Gegensatz zu Metriken ein recht junger Ansatz zur Verbesserung der Codequalität: sein Ziel ist es, auf Basis von funktionalitätserhaltenden Transformationen des Quellcodes, dessen Komplexität zu reduzieren und damit seine Lesbarkeit und Änderbarkeit zu verbessern. Dazu werden schlechte Programmierpraktiken, sog. „Code Smells“, durch bessere bzw. lesbarere Konstrukte ersetzt, während gleichzeitig durch Regressionstesten sichergestellt wird, dass keine neuen Fehler in die Software eingeführt wurden. Code Smells und Softwaremetriken stellen somit beide eine wichtige Indikatorfunktion für mangelhaften Code bereit, allerdings wurden die Auswirkungen von Refactoring auf Softwaremetriken und eventuelle Zusammenhänge zwischen beiden bis heute nur wenig betrachtet. Erste Arbeiten von Du Bois et al. [DDV04] und Stroggylos und Spinellis [SS07] deuten aber bereits darauf hin, dass Refactoring einen eher negativen Einfluss, auf die Messwerte gängiger objektorientierter Softwaremetriken [CK94] haben kann.

Dieser Beitrag geht daher der Frage nach, wie die Messwerte der gängigsten Softwaremetriken durch eine gezielte Veränderung der Quelltextstruktur durch Refactoring beeinflusst werden. Unter der Annahme, dass planvoll durchgeführte Refactorings tatsächlich die Codequalität erhöhen, eignen sie sich offensichtlich ideal für einen Vorher-Nachher-Vergleich von Metriken, der neue Aufschlüsse über deren Aussagekraft bzgl. Codequalität geben könnte. In den folgenden Kapiteln wird sich auf Basis der dort vorgestellten Untersuchungen, die weit über bisherige Arbeiten hinausgehen, herausstellen, dass die von Fowler propagierten Refactorings Messwerte bekannter Softwaremetriken oftmals negativ beeinflussen bzw. einen überwiegend vernachlässigbaren Einfluss auf darauf aufbauende Qualitätsmodelle ausüben. Das folgende Kapitel liefert zunächst einen vertieften Einblick zu Softwaremetriken und Refactorings, bevor Kapitel 3 die von Fowler vorgeschlagenen Refactorings mit Hilfe von über 20 bekannten Metriken vermisst. Kapitel 4 erläutert die bei der Vermessung zweier Fallstudien (Fowlers „Video Store“-Beispiel und das quelloffene Lucene-Framework) gewonnenen Erkenntnisse, die in Kapitel 5 diskutiert und in Kapitel 6 zusammengefasst werden.

2. Forschungsstand zu Metriken und Refactorings

Um die vorwiegend kritische Haltung vieler Autoren gegenüber Softwaremetriken noch einmal zu unterstreichen, gibt dieses Kapitel zunächst einen kurzen Abriss vor allem von Arbeiten, die die Aussagekraft bekannter Softwaremetriken in Frage stellen. Danach werden wichtige Grundlagen des Refactoring vorgestellt und diskutiert, bevor eine Gegenüberstellung beider Ansätze die Kernidee dieses Beitrags, nämlich ein besseres Verständnis für die Beeinflussung von Metriken durch Refactorings, illustriert.

2.1 Softwaremetriken

Kritische Berichte zu Softwaremetriken sind in der Literatur mittlerweile recht zahlreich. Vor einigen Jahren entdeckten z.B. van der Meulen und Revilla [MR07] eine Abhängigkeit von McCabes zyklomatischer Komplexität (CC): sie konnten in ihren Untersuchungen nachweisen, dass diese stark von der Länge des betrachteten Quelltexts, also den Lines of Code (LOC) beeinflusst wird (mit einer Korrelation von 0,95), die Codekomplexität also offenbar mit der Länge des Programmcodes wächst. Kaur et al. [Ka09] untersuchten in ihrer Analyse sowohl McCabes CC als auch diverse Halstead-Metriken und kamen zu dem Schluss, dass die jeweiligen Definitionen einige schwerwiegende Probleme beinhalten. Beispielhaft genannt seien hier die nur vage Definition von Operanden und Operatoren bei Halstead oder abermals erkannte Abhängigkeiten der zyklomatischen Komplexität von der Programmlänge. Des Weiteren sollen an dieser Stelle Jones et al. [JC94] genannt werden, die in ihrer Arbeit versuchten, die Stärken und Schwächen verschiedener Metriken zu identifizieren. Auch in dieser Veröffentlichung wird auf Lücken in der Definition mehrerer Metriken hingewiesen, beispielhaft genannt seien sowohl die oft ungenaue Definition der LOC als auch die Einflüsse verschiedener Programmiersprachen – bzw. Programmierstile auf gängige Metriken mit LOC-Bezug.

Stamelos et al. [ST02] liefern eine ausführliche Fallstudie zur Anwendung von Softwaremetriken auf über 100 in C geschriebene Applikationen (z.B. mail) aus der SUSE 6.0 Linux-Distribution. Doch auch der Erkenntnisgewinn dieses Beitrags bleibt ernüchternd: zwar konnten die Metriken mit bekannten Grenzwerten (z.B. aus [Mc76, Ha77]) verglichen werden, doch eindeutige Qualitätsaussagen waren auf Basis dieses einfachen Modells nicht möglich. Ferner fanden sich bei Stamelos et al. verschiedene Ausreißer, die weit über den Grenzwerten lagen, ohne dass dafür konkrete Ursachen in Form von „schlechtem“ Code festzustellen waren, ähnlich wie das bei der zuvor bereits angesprochenen Untersuchung der Autoren der Fall war [BH12]. Wie bereits in der Einleitung angedeutet, konnten Nagappan et al. [NBZ06] und Schröter et al. [Sc06], jüngst immerhin einen Zusammenhang zwischen Komplexitätsmetriken und der Anzahl an Fehlern in Softwaresystemen empirisch nachweisen. Ein weiterer Nachweis der Anhängigkeit zwischen Coupling-Metriken und Softwarefehlern wurde von Binkley et al. [BS09] geführt. Es bleibt allerdings nach wie vor unklar, wie sich hohe Werte bei Komplexitätsmetriken auf andere Qualitätsfaktoren wie beispielsweise die Verständlichkeit des Quelltextes im Detail auswirken, auch wenn als erste Vermutung natürlich

ein Zusammenhang zwischen hohen Komplexitätsmesswerten und schlechter Verständlichkeit naheliegend scheint.

2.2 Refactoring

Eine der ersten Arbeiten im Bereich Refactoring wurde 1992 von Opdyke [Op92] veröffentlicht. In seiner Dissertation werden u.a. erste Refactoring-Patterns zum Erzeugen von Super- und Unterklassen beschrieben. Die Grundidee von Refactoring ist es, bestehenden Code ohne Veränderung der Funktionalität neu zu strukturieren, um dadurch die Codekomplexität zu reduzieren und spätere Anpassungen und Erweiterungen einfacher durchführen zu können sowie dadurch Zeit und Kosten für Weiterentwicklungen zu reduzieren [Al01]. In seinem bekannten Buch aus dem Jahre 1999 [Fo99] fasst Fowler eine Reihe häufig in der Praxis genutzter Refactoring-Patterns und Anwendungsfälle zusammen. Eine Übersicht über jüngere Forschung im Bereich Refactoring liefern beispielsweise Mens et al. [MT04], die auch mögliche positive Auswirkungen von Refactoring auf einzelne Qualitätsindikatoren, wie Wiederverwendbarkeit oder Komplexität, beschreiben. Eine erste Analyse der Auswirkungen von Refactorings auf Softwarewartungen durch Wilking et al. [WKK7], zeigt immerhin eine leichte Verbesserung der Wartbarkeit restrukturierter Software, die allerdings mit einem zusätzlichen Aufwand für das Refactoring erkauft werden muss. Weitere empirisch gesicherte Erkenntnisse über die Auswirkungen von Refactoring auf die Code- oder gar Softwarequalität sind den Autoren bis dato nicht bekannt. Nichtsdestotrotz ist natürlich eine Hoffnung der Refactoring-Befürworter, dass eine regelmäßige Überarbeitung der Codestruktur zumindest einer Komplexitätserhöhung des Quelltexts entgegen wirkt.

2.3 Metriken und Refactoring

Wie bereits in der Einleitung angesprochen, sind den Autoren bisher nur drei Arbeiten geläufig, die objektorientierte Softwaremetriken vor und nach einem Refactoring vergleichen. Alshayeb [Al09] untersuchte die Auswirkungen von Refactoring auf fünf Software-Qualitätsindikatoren (adaptability, maintainability, understandability, reusability und testability). Dazu wurden die Auswirkungen von Refactorings auf einzelne Methoden dreier ausgewählter Programme mit Hilfe von neun Metriken (DIT, NOC, CBO, RFC, FOUT, WMC, NOM, LOC und LCOM [CK94]), vgl. Tabelle 3) analysiert und auf die Indikatoren übertragen. Die Untersuchung kam zu dem wenig greifbaren Ergebnis, dass das Zusammenspiel zwischen Refactorings und Qualitätsmerkmalen sehr komplex zu sein scheint und künftig genauer untersucht werden sollte.

In Du Bois [DM03] Untersuchungen eines einfach LAN-Simulator-Systems erhöhten (+) sich Messwerte der fünf genutzten objektorientierten Metriken nach einem Refactoring bei 50% der Ergebnisse (s. Tabelle 1). Bei 30% blieb das Ergebnis unverändert (o) und bei nur rund 20% sanken (-) die Messwerte.

Tabelle 1. Überblick über die Ergebnisse von Du Bois et al. [DM03]

Refactoring	Number of Methods	Number of Children	Coupling betw. Obj.	Response for a Class	Lack of Cohesion
EncapsulateField	+	o	o	+	+
PullUpMethod subclass	-	o	-	-	-
PullUpMethod superclass	+	o	+	+	+
ExtractMethod	+	o	o	+	+

Auch Stroggylos und Spinellis [SS07] kommen in ihren Stichproben auf teilweise deutliche Erhöhungen der Metrikergebnisse von bis zu 50 Prozent. Ihre Arbeit untersuchte ebenfalls nur objektorientierte Metriken (wie z.B. die in Tabelle 1 ebenfalls gezeigten) und kleinere, unabhängige Stichproben von Programmen wie Apache Log4j, MySQL Connector/J und JBoss und bleibt somit abermals von begrenzter Aussagekraft.

Zusammenfassend bleibt festzuhalten: bisherige Untersuchungen basierten zumeist auf wenigen Refactoring-Patterns und analysierten nur eine geringe Anzahl von Metriken (die objektorientierten nach Chidamber und Kemerer [CK94]). Ferner ist den Autoren bis dato keine Untersuchung bekannt, die die Auswirkungen von Refactorings beispielsweise auf die Halstead-Metriken [Ha77] oder über mehrere konsekutive Refactoring-Schritte hinweg analysiert hätte. Des Weiteren existiert noch keine detaillierte Analyse über den Einfluss aller von Fowler gelisteten Refactorings auf gängige Softwaremetriken, so dass nach wie vor unklar bleibt, in wie weit Beiden ein ähnliches Verständnis der Codekomplexität zugrunde liegt. Diese Arbeit leistet einen ersten Beitrag zum Schließen dieser Lücke, indem sie alle Fowler-Refactorings mit den in Literatur und Praxis gängigsten Softwaremetriken analysiert und zudem zwei Case Studies entsprechend untersucht.

2.4 Zur „Optimierung“ von Metriken

In der Industrie ist es („unter der Hand“) eine durchaus gängige Praxis, Softwaresysteme so zu „optimieren“, dass sie zuvor vereinbarte Grenzwerte bei Metriken einhalten können. Implizit einher geht damit der Wunsch, dass dadurch die Codekomplexität und auch die Softwarequalität verbessert werden mögen. Wie im Folgenden beispielhaft gezeigt, ist eine solche Verbesserung der Messwerte mit etwas Hintergrundwissen über den Aufbau der Metriken sehr leicht machbar, führt aber im folgenden Beispiel zu offensichtlichen Verschlechterungen in der Lesbarkeit des Quelltexts bzw. umgekehrt würde ein entsprechendes funktionserhaltendes und sinnvolles Refactoring des Codes zu schlechteren Metrikwerten führen. Alle im weiteren Verlauf vorgestellten Messergebnisse wurden mit dem kommerziellen Analysewerkzeug Understand[1] (Version 3.0) von Scientific Tools Inc. erhoben, so dass Beeinflussungen der Messungen durch eine unterschiedliche Interpretation der Metriken ausgeschlossen werden können.

Vorrangiges Ziel des folgenden einfachen Versuchs ist es, die Auswirkungen eines Refactorings auf Halsteads Softwaremetriken [Ha77] zu demonstrieren. Dazu wurde eine Berechnung der „Mitternachts-Formel“ zur Lösung quadratischer Gleichungen vereinfacht, indem die eigentliche Gleichung auseinander gezogen und mit Zwische-

[1] http://www.scitools.com (Letzter Zugriff 21.07.2012)

nergebnissen versehen wurde. Ein solches Vorgehen ermöglicht es Entwicklern üblicherweise die einzelnen Schritte leichter zu erfassen und zu verstehen und entspricht dem Refactoring-Pattern „Introduce Explaining Variable“ [Fo99]). Hier wurde die Formel in drei Teile aufgespalten und jedes Teil erhielt einen beschreibenden Namen (endResult für das Endergebnis oder qudEquPos für die positive Lösung der quadratischen Gleichung; die Abkürzungen wurden rein aus Platzgründen verwendet, in der Praxis sollten die Begriffe natürlich vollständig ausgeschrieben werden).

Tabelle 2. Funktional identische Code-Snippets mit unterschiedlichen Metrikwerten.

Snippet 1a	Snippet 1b

int result= -(b+sqrt(pow(b,2)-4*a*c)/2*a)
-(b-sqrt(pow(b,2)-4*a*c)/2*a);
 	
if(getCondition()<5)
if(getBill() != getCondition())
if(!(getEnable() != OCCUPIED))
condition = true;
	int qudEquPos = -(b+sqrt(pow(b,2)-4*a*c)/2*a);
	int qudEquNeg = -(b-sqrt(pow(b,2)-4*a*c)/2*a);
	int endResult = qudEquPos - qudEquNeg;
	if(getCondition()<5) {
	if(getBill() != getCondition()) {
	if(!(getEnable() != OCCUPIED)) {
	condition = true;

Die verbesserte Lesbarkeit des Quellcodes wird also insbesondere durch das Einfügen zusätzlicher Variablen erreicht. Aus der Sicht der Halstead-Metriken benötigt der Code nun deutlich mehr Operatoren (z.B. +, =) und Operanden (z.B. qudEquPos), was automatisch zu schlechteren Metrikwerten führt (bspw. verschlechtert sich das Halstead Volume von 413 auf 493). Die zusätzlich eingefügten Variablen haben aber nicht nur einen negativen Effekt auf die Halstead-Metriken, durch das Refactoring evtl. überflüssig gewordene und entfernte Kommentare würden ferner den Kommentaranteil des Codes reduzieren, der oftmals ebenfalls als Qualitätsmerkmal [KWR10] angesehen wird. Bereits in diesem einfachen Beispiel ist die Robustheit der Metriken gegenüber einem einfachen Refactoring also offenbar nicht gegeben.

3. Analyse der Fowler-Patterns

Die eben beispielhaft gezeigten Einflüsse des Patterns „Introduce Explaining Variable“ auf bekannte Softwaremetriken werfen die Frage auf, ob auch weitere Refactorings, die die Lesbarkeit und Verständlichkeit von Sourcecode erhöhen sollen, von gängigen Softwaremetriken ebenfalls negativ bewertet werden würden? Fowler [Fo99] beschreibt in seinem bekannten Buch und auf seiner Webseite [Fo12] immerhin mehr als 50 solcher Patterns. Um die grundlegenden Wechselwirkungen zwischen Refactoring und Softwaremetriken zu analysieren, wurden für alle Refactorings jeweils die von Fowler gegebenen Beispiele mit den heute gängigsten Softwaremetriken (s. Tabelle 3) vermessen: einmal mit dem entsprechendem „Code Smell“ vor dem Refactoring und einmal nach dem Anwenden des Refactorings. „Inverse“ Patterns, die ihre Wirkung gegenseitig aufheben, z.B. „*Pull Up Method*“ und „*Push Down Method*“, wurden jeweils nur einmal untersucht. Außerdem wurden fortgeschrittene Enterprise-Patterns, wie z.B. das Pattern „*Wrap entities with session*“[2], nicht näher betrachtet.

[2] http://www.refactoring.com/catalog/wrapEntitiesWithSession.html (letzter Zugriff 31.08.2012)

Tabelle 3. Übersicht der genutzten Metriken

1) Halstead-Metriken [Ha77]	*Basismesswerte: unterschiedliche Operatoren (n1) und Operanden (n2), Anzahl Operatoren (N1) und Operanden (N2)* Metriken: Vocabulary (Voc), Volume (Vol), Difficulty (Dif), Effort (Eff), Length (Len)
2) LOC- Metriken	Lines of Code (LOC), Commented Lines of Code (CLOC), Statements (Stat), Declarations (Decl).
3) Komplexitätsmetriken	Maximale zyklomatische Komplexität (Max.CC) [MC76], Durchschnittliche zyklomatische Komplexität (CC) [MC76], Maximales Nesting (Max.Nest.)
4) objektorientierte Metriken [CK94]	Lack of Cohesion (LCOM), Depth of Inheritance Tree (DIT), Count of Base Classes (CBC), Count of Coupled Classes (CBO), Count of Derived Classes (NOC), Count of All Methods (RFC), Count of Instance Methods (NIM), Count of Instance Variables (NIV), Count of Methods (WMC)

Insgesamt wurden im Zuge dieser Untersuchung Ergebnisse von 21 bekannten und häufig genutzten Metriken, sowie den 4 Halstead-Basismesswerten (s. Tabelle 3), die zur Berechnung der eigentlichen Halstead-Metriken benötigt werden, in Fowlers 50 Beispielprogrammen erhoben. Entsprechend wurde 1250 Mal der ursprüngliche Code mit der restrukturierten Version verglichen. Ein positiver Wert (> 0) zeigt einen Anstieg der Metrikergebnisse nach dem Refactoring, was bei den allen verwendeten Metriken eine Verschlechterung der Qualität indiziert (vgl. [Ha77, Mc76, CK94]. Ist das Ergebnis negativ, so hat sich der Metrikmesswert verringert. Nach dem Refactoring waren 38% (469) der Metriken höher als zuvor, nur bei 13% (163) der Messungen ergaben sich Verbesserungen, während die restlichen 49% (609) unverändert blieben. Die folgende Tabelle 4 fasst die ermittelten Ergebnisse auf einen Blick zusammen.

Tabelle 4. Übersicht der Veränderung bei den Patterns.

Verschlechterung	469	38%
Unverändert	618	49%
Verbesserung	163	13%
Gesamt	1250	100%

Tabelle 5 und Tabelle 6 schlüsseln diese Ergebnisse für die einzelnen Metriken auf und verdeutlichen, dass bei 14 von 21 Metriken die überwiegende Anzahl von Refactorings zu einer Verschlechterung führt, während nur bei vier die Mehrzahl bei „unverändert“ liegt. Keines der angewandten Refactorings führt zu einer überwiegenden Verbesserung der Metrikwerte und damit zu einer Verringerung der dadurch implizierten Komplexität.

Tabelle 5. Veränderungen klassischer Metriken bei den 50 „Fowler-Refactorings“.

	Len	**Voc**	**Vol**	**Max CC**	**Eff**	**Dif**	**LOC**	**Decl**	**Stat**	**CLOC**	**CC**	**Max Nest**
Verschlecht.	27	28	29	2	29	29	31	26	27	4	0	0
Unverändert	8	11	6	44	5	10	10	21	13	44	50	50
Verbesserung	15	11	15	4	16	11	9	3	10	2	0	0

Tabelle 6. Veränderungen der OO-Metrikwerte bei 50 Refactorings.

	LCOM	**DIT**	**CBC**	**CBO**	**NOC**	**RFC**	**NIM**	**NIV**	**WMC**
Verschlecht.	17	12	12	12	7	21	18	9	19
Unverändert	31	36	36	36	41	25	27	37	27
Verbesserung	1	1	1	1	1	3	4	3	3

Refactorings, die das Volumen des Sourcecodes vergrößern, z. B. durch Hinzufügen von zusätzlichen Klassen oder Methoden, haben per definitionem einen negativen Einfluss auf Halstead- und LOC-Metriken. Zu diesen Refactorings zählen unter anderem *Extract Method*, *Extract Class* oder *Extract Interface*. Entsprechende Ergebnisse werden in Tabelle 7 an Hand konkreter Messergebnisse illustriert. Die Werte zeigen die Veränderung pro Metrik vor einem Refactoring im Vergleich zu einer Messung danach. Beispielsweise hat sich bei *Extract Class* das Halstead-Volume um 30 Metrikpunkte erhöht. Negative Werte in der Tabelle beschreiben eine Verbesserung, bei einem positiven Wert erhöht sich entsprechend die von den Metriken bewertete Codekomplexität.

Tabelle 7. Auszug aus den Differenzen der Halstead-Messungen für verschiedene Refactorings.

Pattern Name	***n1***	***n2***	***N1***	***N2***	**Len**	**Voc**	**Vol**	**Dif.**	**Eff**
Add Parameter	*0*	*2*	*0*	*2*	2	2	7	0	7
Extract Class	*5*	*3*	*6*	*4*	10	8	30	3	90
Extract Interface	*8*	*2*	*8*	*2*	10	10	22	4	44
Extract Method	*5*	*4*	*6*	*5*	11	9	35	3	176
Extract Subclass	*11*	*3*	*12*	*3*	15	14	42	5	108
Extract Superclass	*11*	*3*	*12*	*3*	15	14	42	5	108

Bei „Add Parameter“ wird beispielsweise ein Übergabeparameter zur Methode hinzugefügt. Der Parameter und sein Typ werden in Understand als zwei neue Operanden gezählt und erhöhen somit n2 und N2 und die davon abhängenden Metriken entsprechend.

Die in diesem Kapitel vorgestellten Ergebnisse geben zwar einen ersten Einblick in die Auswirkungen von Refactorings auf Softwaremetriken, sind aber natürlich nicht repräsentativ für die Verteilung der Refactorings, da sie von einer Gleichverteilung in der Praxis ausgehen. Um praxisnähere Werte für die erhobenen Metriken zu erhalten, werden im folgenden Kapitel weitere Ergebnisse aus der Untersuchung zweier exemplarisch ausgewählter Systeme vorgestellt.

4. Fallstudien

Als Untersuchungsobjekte dienten das „Video Store Example“ (ca. 110 bis 230 LOC) aus Fowlers Buch als ein einfaches, aber klar nachvollziehbares „Spielzeugbeispiel“ und das Open-Source-Projekt Lucene (über 140 KLOC) als ein reales, weltweit im Einsatz befindliches System. Wie zuvor wurden die Metriken jeweils vor und nach einem Refactoring erfasst und miteinander verglichen.

4.1 Analyse von Fowlers „Video Store“

Mit Fowlers „Video Store“ [Fo99] wurde zunächst ein beispielhaftes System untersucht, das sieben klar definierte und gut nachvollziehbare Refactoring-Iterationen durchlaufen hat und dadurch, intuitiv nachvollziehbar, besser verständlich geworden ist. Die von Fowler durchgeführten Refactorings wurden Schritt für Schritt nachvollzogen und nach jedem Schritt vermessen. Dazu wurden immer zwei Messreihen durchgeführt, nämlich

einerseits der Vergleich der Revision vor dem Refactoring mit der Revision danach und andererseits die Veränderung zwischen der jeweiligen Revision und der ursprünglichen Version (vgl. Tabelle 9). Insgesamt wurden somit 2 Mal in jedem Refactoring-Schritt 21 Metriken und die 4 Halstead-Basiswerte (vgl. Tabelle 3) vermessen, was 350 Messwerte ergab. Im Ergebnis vergrößerten sich bei 57% aller Messungen die Metrikwerte, während sie sich bei nur 9% verringerten. Die entsprechenden Veränderungen sind in Tabelle 8 zusammengefasst.

Tabelle 8. Übersicht Messergebnisse für Fowlers Video-Store.

	LOC	Halstead Basis	Halstead-Metriken	OO	Gesamt	%
Verschlechterung	50	45	49	56	200	57%
Unverändert	41	4	4	69	118	34%
Verbesserung	7	7	17	1	32	9%
Gesamt	98	56	70	126	350	100%

Besonders interessant sind die Auswirkungen des finalen Schritts, in welchem mehrere Unterklassen eingeführt werden, um die verschiedenen Filmgenres voneinander zu trennen. Dadurch wurden vor allem bei den Halstead-Metriken deutliche Zuwachsraten von teilweise über 100% im Vergleich zur Ursprungsversion (Tabelle 9) gemessen.

Tabelle 9. Halstead-Metriken des finalen Video-Store im Vergleich zur ursprünglichen Version.

Length	Vocabulary	Volume	Difficulty	Effort
+77%	+121%	+63%	+110%	+18%

4.2 Analyse des Open-Source-Projekts Lucene

Um die bisher gefunden Resultate an Hand eines realen System zu überprüfen und einen Eindruck über die realen Auswirkungen von Refactorings zu erhalten, wurde abschließend das Open-Source-Projekt Apache Lucene vermessen. Dafür wurden zwölf verschiedene Revisionen aus dem SVN-Repository des Projekts ausgewählt, die ausschließlich mit dem Kommentar „Refactored“ oder „Refactoring“ gekennzeichnet sind. D.h. es wurden – sofern die Kommentare im SVN korrekt gesetzt worden sind – nur Revisionen verwendet, bei denen ein reines Refactoring und keinerlei Veränderung der Funktionalität stattgefunden hat. Für die Messungen wurden jeweils die mit „refactored“ gekennzeichnete Revision und die vorhergehende Revision aus dem Repository geladen; für diese Messungen wurden wiederum die in Tabelle 3 genannten 21 Metriken und 4 Basiswerte erhoben, was für insgesamt 300 Vergleiche ergab: auch bei Lucene ließ sich nur in 21% der Fälle ein verbesserter Messwert nachweisen, eine Verschlechterung aber in 54%; die folgende Tabelle 10 fasst die Ergebnisse zusammen.

Tabelle 10. Übersicht Metrikveränderungen nach Refactoring bei Apache Lucene.

	Absolut	Relativ
Verschlechterung	170	57%
Unverändert	63	21%
Verbesserung	67	22%
Gesamt	300	100%

In der folgenden Tabelle 11 wurden die verschiedenen Metrik-Arten in fünf Blöcke unterteilt: der erste Block beinhaltet alle LOC-Metriken, der zweite Block alle Halstead-Metriken, der dritte alle OO-Metriken und der letzte Block alle Komplexitätswerte nach McCabe (vgl. wiederum Tabelle 3). Auch hier zeigt sich deutlich, dass in jedem Block die Anzahl der Ergebnisse, die sich nach dem Refactoring verschlechtert haben, am größten ist. Besonders eklatant sind die Werte für die objektorientierten Metriken, bei denen es nur in acht Prozent aller untersuchten Fälle zu einer Verbesserung kommt.

Tabelle 11. Übersicht der Verteilung nach Metrik-Arten.

	LOC		Halstead Basis		Halstead Metriken		OO-Metriken		McCabe	
Verschlechterung	35	58%	32	67%	37	62%	53	49%	13	54%
Unverändert	9	15%	2	4%	0	0%	46	43%	6	25%
Verbesserung	16	27%	14	29%	23	38%	9	8%	5	21%
Gesamt	60	100%	48	100%	60	100%	108	100%	24	100%

Um die Auswirkungen eines einzelnen Refactoring-Schritts besser zu illustrieren, soll an dieser Stelle beispielhaft die Revision 604870 von Apache Lucene genauer betrachtet werden. In dieser wurden im Vergleich zur vorherigen Revision zwei von 7188 Dateien verändert sowie drei neue Dateien hinzugefügt. Laut Revisionskommentar diente das Refactoring dazu, eine gemeinsame Helfer-Klasse zu schaffen ("Refactored to have a common PayloadHelper class"). Von den 21 gemessenen Metriken und 4 Basiswerten steigen 22 zum Teil deutlich an. Zum Beispiel wächst die Anzahl der LOC in den zwei veränderten Dateien um 22%, die durchschnittliche CC um 50%. Nur die Ergebnisse von Halsteads Effort und Lack of Cohesion (LCOM) fallen.

4.3 Fortgeschrittene Qualitätsmodelle

In der Vergangenheit wurde verschiedentlich daran gearbeitet, mehrere Metriken in einen Zusammenhang zu bringen, um aussagekräftige Qualitätsmodelle bzw. -indikatoren für Software zu entwickeln. Ein Beispiel ist der sogenannte Maintainability Index (MI, [Co94]). Dieser basiert auf einem einfachen Stufenmodell, bei dem alle Funktionen über einem bestimmten Schwellwert als qualitativ hochwertig angesehen werden. In den vorgestellten Fallstudien lag keine der vermessenen Methoden unter diesem Wert und auch alle durchgeführten Refactorings ergaben keinerlei Veränderung der Einstufung. Ähnliche Ergebnisse liefert das Modell der Software Improvement Group (SIG) [AYV10] aus den Niederlanden. Auch dort wurden über 90% der Methoden bereits vor dem Refactoring als sehr gut klassifiziert und keine der Einstufungen veränderte sich durch die Refactorings.

In eine ähnliche Richtung gehen auch die Ergebnisse des USUS-Modells [PR10], das in der Lage ist, mehrere Revisionen einer Software miteinander zu vergleichen und sich daraus ergebende Trends zu bestimmen. USUS erkannte nach den Refactorings in jedem der drei vermessenen Projekte einen Trend hin zu schlechterer Qualität und zudem zwei neue Problembereiche (sog. Hotspots) bei den Fowler- Refactorings sowie einen bei Lucene. Tabelle 12 illustriert die Ergebnisse der USUS-Analyse, neben der Veränderung der Hotsports (HS) zeigt sie die Veränderung des Durchschnitts der genutzten Metrik (D) und jeweils die Elemente, die der genutzten Metrik zugrunde liegen.

Tabelle 12. Übersicht der USUS Ergebnisse, negative Trends sind kursiv dargestellt.

		Fowler Patterns		**Video Store**		**Lucene**	
Metrik	**Element**	**D**	**HS**	**D**	**HS**	**D**	**HS**
Average component dependency	Klasse	+0,1	0	*-28,1*	*0*	0	0
Class size	Klasse	0	0	0	0	*0*	*0*
Cyclomatic Complexity	Methode	*-0,4*	*0*	-4,4	0	*0*	*0*
Method length	Methode	0	0	-2,3	0	*0*	*+1*
No. non-static/final public fields	Klasse	*+1,4*	*+2*	0	0	*0*	*0*
Packages w. cyclic dependencies	Package	0	0	0	0	0	0

Auch dieses Qualitätsmodell zeigt also meist nur marginale Veränderungen nach den Refactorings, die zudem oftmals noch negativ interpretiert werden und ist damit nicht in der Lage, die Auswirkungen der Refactorings sinnvoll zu interpretieren.

5. Diskussion

Im vorherigen Kapitel wurden drei aufeinander aufbauende Messreihen mit insgesamt 1900 Vergleichen der bekanntesten Softwaremetriken aus der Literatur vorgestellt. Deren Resultate ähneln sich bei allen drei vermessenen Beispielen – trotz ihrer Verschiedenheit – sehr stark: zumeist steigt die Mehrzahl (insgesamt 47%) der Metrik-Ergebnisse nach einem Refactoring deutlich an, nur bei 15% aller durchgeführten Refactorings verringert sie sich, was auf Grund des großen Umfangs von möglichen Kombinationen stichprobenartig auch analytisch an Hand des Aufbaus der Refactorings und Metriken nachvollzogen werden konnte. Die Ergebnisse sind in Tabelle 13 nochmals auf einen Blick zusammengefasst und zeigen zudem eine große Ähnlichkeit mit den weniger umfangreichen Ergebnissen früherer Untersuchungen ([Du06, SS07]).

Tabelle 13. Übersicht der Auswirkungen aller Messungen im Vergleich zu früheren Veröffentlichungen.

	Ergebnisse dieses Beitrags		[SS07]		[DM03]	
Verschlechterung	983	52%	30	59%	10	50%
Unverändert	848	45%	0	0%	4	20%
Verbesserung	293	15%	21	41%	6	30%
Gesamt	1900	100%	51	100%	20	100%

Erstmals stellt sich mit der in diesem Beitrag durchgeführten Untersuchung heraus, dass Veränderungen nach Refactorings weitgehend unabhängig von der Art der Metrik zu sein scheinen, sowohl die älteren Halstead-Metriken (Verschlechterung von 61%), die

bis dato im Zusammenhang mit Refactorings noch nicht untersucht worden waren, als auch jüngere OO-Metriken (Verschlechterung von 56%) verzeichnen deutliche Komplexitätsanstiege. Gleiches gilt auch für die Art des durchgeführten Refactorings, während bisherige Untersuchungen nur einige wenige Refactorings untersucht haben, konnte im Rahmen dieses Beitrags gezeigt werden, dass die 50 bekanntesten Fowler-Refactorings zumeist ähnliche Auswirkungen auf die gemessenen Metriken haben. Es gab allerdings keine Metrik, die sich bei allen Refactorings durchgängig gleich verhalten hätte, so dass belastbare statistische Aussagen noch nicht getroffen werden können.

Interessant sind auch die Erkenntnisse aus der Anwendung fortgeschrittener Qualitätsmodelle, die die Einflüsse der Refactorings sowohl in den einfachen Demonstrationsbeispielen, als auch in einem realen System, weitgehend überhaupt nicht abbilden können. Andererseits verdeutlicht die durchgeführte Analyse des Lucene-Frameworks, wie selbst kleine Änderungen oft großen Einfluss auf Metrikwerte haben können: Im Beispiel aus Abschnitt 4.2 wurden nur insgesamt fünf von ca. 18.000 Dateien durch Refactoring verändert, was nichtsdestotrotz einen merklichen Einfluss auf alle Metriken hatte (s. ebendort). Da alle durchgeführten Messungen nur öffentlich verfügbaren Code verwendet haben, können sie aber jederzeit gut nachvollzogen und auch analysiert werden. Ferner wurde auch für alle Messungen das gleiche Analysewerkzeug („Understand“) verwendet, so dass Verfälschungen der Messergebnisse durch unterschiedliche Metrikinterpretationen ausgeschlossen werden können.

Auf Grund des noch immer recht geringen Umfangs der vorgestellten Untersuchung erscheint eine Verallgemeinerung der Ergebnisse zwar nach wie vor nicht angeraten, es liegen nun aber deutliche Verdachtsmomente vor, dass Softwaremetriken und Refactorings eine gegensätzliche Interpretation von Codequalität aufzuweisen scheinen, so dass weitergehende Forschung in diesem Bereich angeraten erscheint.

6. Fazit und weitere Schritte

Auch nach mehreren Jahrzehnten intensiver Forschung fehlt es den aktuell gängigen Softwaremetriken offensichtlich noch deutlich an „Ausdrucksstärke“, da sie nicht einmal in der Lage scheinen, die Komplexität eines Sourcecodes korrekt zu erfassen, also inhärente Domänenkomplexität von schlechter Programmierung zu unterscheiden. Die in jüngster Zeit populär gewordenen Refactorings zielen genau auf die Behebung schlechter Programmierpraktiken ab und bieten sich entsprechend als eine einfache Möglichkeit für einen Vorher-Nachher-Vergleich von Softwaremetriken an. Die wenigen, bisher veröffentlichten Untersuchungen dieser Art begründeten jedoch den Verdacht, dass gängige Metriken nicht hinreichend in der Lage sind, die vermutlich positiven Einflüsse von Refactorings auf die Quelltextqualität abzubilden. Sie müssen allerdings sowohl im Hinblick auf die verwendeten Metriken, als auch auf die untersuchten Refactorings als wenig umfassend bezeichnet werden. Sie werden von den in diesem Beitrag vorgestellten Untersuchungen um ein Vielfaches übertroffen: neben einer systematischen Vermessung der Fowler-Refactorings wurden hierfür erstmals mehrere Überarbeitungen eines großen Open-Source-System mit gängigen Softwaremetriken untersucht. Die gefundenen Ergebnisse erhärten somit den Verdacht, dass die Metriken nicht in der Lage

sind, die durch die Refactorings erreichte Komplexitätsreduktion nachzuvollziehen, wo sie überhaupt eine Veränderung erkennen, ist es meist eine Zunahme der Komplexität. Aus Sicht der Softwaremetriken lässt sich natürlich umgekehrt die Vermutung aufstellen, dass Refactorings die Codequalität negativ beeinflussen könnten; welche Perspektive Recht behalten wird, ist bis dato noch nicht abschließend zu klären. Die Autoren planen daher, in naher Zukunft weitere Open-Source-Projekte zu untersuchen, um die hier vorgestellten Ergebnisse weiter zu untermauern (oder ggf. zu widerlegen). Als zentrales Ergebnis dieses Beitrags bleibt letztlich festzuhalten, dass gängige Softwaremetriken und Refactorings ein weitgehend gegensätzliches „Verständnis“ von Codekomplexität aufzuweisen scheinen.

Bei einer stichprobenartigen Betrachtung des Aufbaus einiger Metriken ist dieser Widerspruch zwar nachvollziehbar, genau verstanden sind die Einflüsse von Refactoring auf Metriken aber bei weitem noch nicht. Eine detaillierte analytische Untersuchung der Auswirkungen von Refactorings auf Softwaremetriken ist daher für die Zukunft sicher wünschenswert, aber mit einem nicht unerheblichem Aufwand verbunden. Allein die Kombination der in diesem Beitrag verwendeten 21 Metriken und 4 Basiswerten mit 50 Refactorings ergibt 1250 zu betrachtende Kombinationen, so dass mittelfristig eine weitgehend automatisierbare Quellcode-Vermessung als die ökonomischere Analysemöglichkeit erscheint. Dennoch wollen die Autoren ausgewählte Refactoring-Patterns zukünftig genauer untersuchen, um besser verstehen zu können, welchen Einfluss sie auf den Sourcecode und daraus abgeleitete Metriken haben. Eventuell lassen sich so einzelne Metriken, die Verbesserungen im Code durch Refactorings recht zuverlässig erkennen können, identifizieren und zu einem verbesserten Qualitätsmodell zusammenfassen.

Eine weitere naheliegende Aufgabe ist es, die Auswirkungen anderer Ansätze, die ebenfalls die Codequalität erhöhen sollen, wie z.B. Design oder Architectural Patterns oder auch Coding Guidelines [SA05], auf Softwaremetriken in ähnlicher Weise zu untersuchen. Aus den insgesamt gewonnenen Erkenntnissen lassen sich letztlich eventuell neuartige Metriken (z.B. basierend auf einer Zählung von Code Smells oder Patterns) mit besserer Aussagekraft ableiten oder möglicherweise sogar die erkannten Effekte aus bestehenden Metriken herauszurechnen, um deren Aussagekraft zu verbessern. Der Verdacht, dass heute verfügbare Softwaremetriken und Qualitätsmodelle anerkannte Praktiken guten Softwaredesigns (d.h. Refactorings und Design Patterns, vgl. [Se07]) sowie guter Software-Architektur nicht erfassen können, dürfte jedenfalls immer schwerer von der Hand zu weisen sein.

Literaturverzeichnis

[Al09] Alshayeb, M.: Empirical investigation of refactoring effect on software quality, Elsevier Information and software technology, Vol. 51, Nr. 9, S.1319-1326, 2009

[Al01] Alshayeb, M.; Li, W; Graves, S.; An empirical study of refactoring, new design, and error-fix efforts in extreme programming, 5th World Multiconference on Systemics, Cybernetics and Informatics, 2001

[AYV10] Alves, T.L.; Ypma, C. Visser, J.; Deriving metric thresholds from benchmark data, 2010 IEEE International Conference on Software Maintenance (ICSM), 2010

[BH12] Burger, S.; Hummel, O: Applying Maintainability Oriented Software Metrics to Cabin Software of a Commercial Airliner, 16th European Conference on Software Maintenance and Reengineering, Szeged, 2012
[BS98] Binkley, A.B.; Schach, S.R.: Validation of the coupling dependency metric as a predictor of run-time failures and maintenance measures, Intern. Conf. on Software Eng., 1998
[Co94] Coleman, D.; Ash, D.; Lowther, B.; Oman, P: Using metrics to evaluate software system maintainability Computer, IEEE, Vol., 27, S. 44-49, 1994
[CK94] Chidamber S.R.; Kemerer C.K.: A Metrics Suite for Object Oriented Design, IEEE Transactions on Software Engineering, Vol. 20. Nr. 6, 1994
[DDV04] Du Bois, B., Demeyer, S., Verelst, J; Refactoring-improving coupling and cohesion of existing code; Working Conference on Reverse Engineering, 2004
[DM03] Du Bois, B., Mens, T: Describing the impact of refactoring on internal program quality, Intern. Workshop on Evolution of Large-scale Industrial Software Applications, 2003
[Ha77] Halstead. M.H: Elements of Software Science (Operating and Programming Systems Series), Elsevier Science Inc., 1977
[Ho08] Hoffmann, D. W: Software-Qualität, Springer-Verlag New York Inc, 2008
[Ka09] Kaur, K.; Minhas, K., Mehan, N., Kakkar, N: Static and Dynamic Complexity Analysis of Software Metrics, Engineering and Technology, Citeseer, Vol. 56, 2009
[JC94] Jones, C: Software metrics: Good, bad and missing Computer, IEEE, Vol. 27, 1994
[Fo99] Fowler, M: Refactoring: improving the design of existing code, Addison-Wesley, 1999
[Fo12] Fowler, M: http://www.refactoring.com/, zuletzt besucht am 21.08.2012
[IO11] ISO/IEC: ISO/IEC 25010: 2011, Systems and software engineering-Systems and software Quality Requirements and Evaluation (SQuaRE)-System and software quality models, Switzerland, International Organization For Standardization, 2011
[KWR10] Khamis, N.; Witte, R.; Rilling, J.: Automatic quality assessment of source code comments: the JavadocMiner, Springer Natural Language Processing and Information Systems, S. 68-79, 2010
[Mc76] McCabe, T.J: A complexity measure, IEEE Transactions on Software Engineering, Vol. 2, Nr. 4, S. 308, 1976
[MR07] van der Meulen, M.J.P.; Revilla, M.A.: Correlations between internal software metrics and software dependability in a large population of small C/C++ programs, International Symposium on Software Reliability, 2007
[MT04] Mens, T.; Tourwé, T.: A survey of software refactoring, IEEE Transactions on Software Engineering, Vol. 30, Nr. 2, S. 126-139, 2004
[NBZ06] Nagappan, N.; Ball, T.; Zeller, A.: Mining metrics to predict component failures, International Conference on Software Engineering, 2006
[Op92] Opdyke W.F.; Refactoring Object-Oriented Frameworks, PhD Thesis, University of Illinois, 1992
[PR10] Philipp, M.; Rauch, N.; Einsicht und Handeln mit Usus, Eclipse Magazin 6.10, 2010
[SA05] Sutter, H.; Alexandrescu, A.:C++ Coding Standards: 101 Rules, Guidelines, and Best Practices, Addison-Wesley Professional, 2005
[St02] Stamelos, I.; Angelis, L.; Oikonomou, A.; Bleris, G. L: Code quality analysis in open source software development, Information Systems Journal, Blackwell Science Ltd, Vol. 12, S. 43-60, 2002
[Se07] Seng, O.; Suchbasierte Strukturverbesserung objektorientierter Systeme, Univ.-Verlag Karlsruhe, 2007
[SS07] Stroggylos, K.; Spinellis, D: Refactoring--Does It Improve Software Quality? Fifth International Workshop on Software Quality, 2007
[SZT06] Schröter A.; Zimmermann, T.; Zeller, A.: Predicting component failures at design time. ACM/IEEE International symposium on empirical software engineering, 2006
[WKK07] Wilking, D.; Khan, U.; Kowalewski, S.: An Empirical Evaluation of Refactoring, e-Informatica Software Engineering Journal, Vol. 1, Nr. 1. S. 27-42, 2007