Seminar

Ursachen und Vermeidung von Fehlern in der Softwareentwicklung

Defektabschätzung

Malgorzata Wojciechowska
Institut für Informatik
Freie Universität Berlin
malgorzata@gmx.net

Zusammenfassung:

Um in der rasant voranschreitenden Welt der Software-Entwicklung konkurrenzfähig zu bleiben, müssen Manager großer Projekte die Entstehung von Defekten verhindern, ihre schnelle Entdeckung ermöglichen und eine sofortige Reparatur sicherstellen. Auf diese Weise kann die Software weiter entwickelt werden, um neue Ansprüche der Kunden zu erfüllen.
Um eine statistisch erfassbare Prozesskontrolle für die Softwareentwicklung zu ermöglichen, ist es wichtig, neben der üblichen Qualitätskontrolle, die Defektdaten während aller unterschiedlichen Testphasen zu sammeln. Dabei werden Defektprofile erstellt, die es ermöglichen, Defekte und Versagen genau vorherzusagen. In dieser Arbeit werden ausführliche Beispiele zur Sammlung von Daten über Defekte und Versagen genutzt, um eine Reihe von bekannten Hypothesen aus der Softwaretechnik zu testen.

Inhaltsverzeichnis

1. Einleitung

In der Welt der Softwaretechnik fehlt es an veröffentlichten empirischen Daten über große industrielle Systeme. Das ist einer der Grunde warum diese Disziplin bis jetzt keine korrekte wissenschaftliche Grundlage aufbauen konnte. Im Bereich der Defektabschätzung etwa dominieren Hypothesen, die letztendlich unbewiesen sind, weil es an methodisch überzeugenden Projekten mangelt. Diese Forschung ist jedoch wichtig nicht nur aus Sicht der Wissenschaft, sondern auch für das Management der Softwareindustrie. Deren Manager müssen den Einsatz ihrer begrenzten Betriebsmittel optimieren, um Qualitätsprodukte rechtzeitig und innerhalb des Etats zu liefern. In dieser Arbeit werden gängigen Hypothesen überprüft.

„Als eine **Hypothese** (altgriechisch **υπόθεσις** - *hypóthesis - die unbewiesene Annahme*) bezeichnet man in der Wissenschaft laut Duden *von Widersprüchen freie, aber zunächst unbewiesene Aussage, Annahme als Hilfsmittel für wissenschaftliche Erkenntnisse*“.

Als Grundlage dieser Arbeit dienen zwei Forschungsstudien. Bei der ersten Studie handelt es sich um eine quantitative Studie von Norman und Ohlsson [1], die Defekte und Versage in zwei Versionen eines großen kommerziellen Systems darstellt. In der zweiten Studie von Hassan und Holt [2] wurden sechs große Open Source Softwaresysteme auf Defektanfälligkeit hin geprüft. Bei den zu bestätigenden oder zu widerlegenden Hypothesen aus dem Bereich der Softwaretechnik handelt es sich um:

- das Pareto-Prinzip von Verteilung von Defekten und Versagen,
- den Gebrauch von früheren Defektdaten um spätere Defekt und Versagensdaten vorherzusagen,
- die Häufigkeit und Zeitspanne, in der Veränderungen der Systeme vorgenommen werden,
- Größenmaße in der Defektvorhersage,
- Metriken zur Defektvorhersage und Benchmarking-Defektdaten.

Die Reichweite der hier vorgestellten Ergebnisse muss jedoch als begrenzt gelten. Es handelt sich nicht um allgemeingültige Gesetze der Softwaretechnik, weil sie einige grundlegende Anmerkungen, besondere Testanstrengungen und den betrieblichen Gebrauch nicht berücksichtigen.

2. Methoden und untersuchte Software Projekte

In dieser Arbeit wurden zwei Arten von Untersuchungen durchgeführt. Die erste Untersuchung bezieht sich auf die Hypothesen 1,2,3,4,7,8,9,10 und die zweite Untersuchung bezieht sich auf die Hypothesen 5 und 6.

Erste Untersuchung

Die hier präsentierten Daten basieren auf zwei großen, aufeinander folgenden Veröffentlichungen eines Projektes, das auf der Entwicklung eines Systems zur Telekommunikationsvermittlung beruht. Die frühere Veröffentlichung wird im Folgenden als *Veröffentlichung n* und die spätere Veröffentlichung als *Veröffentlichung n+1* bezeichnet. Für diese Studie wurden jeweils 140 und 246 Module von Veröffentlichung n und Veröffentlichung n+1 zufällig ausgewählt. In dieser Analyse benutzte man eine Menge von Modulen, die entweder unverändert sind oder bereits modifiziert wurden. Die Module variierten in der Größe zwischen 1000 bis 6000 LOC. Beide Veröffentlichungen hatten eine identische Gesamtgröße.

LOC	Release n	Release n+1
<1000	23	26
1001-2000	58	85
2001-3000	37	73
3001-4000	15	38
4001-5000	6	16
5001-6000	0	6
>6000	1	2
Total	140	246

Tabelle 1: Verteilung der Module nach Größe[1]

Die Defektdaten wurden während vier unterschiedlicher Phasen gesammelt:

- Funktionstest (FT)
- Systemtest (ST)
- die ersten 26 Wochen mit einer bestimmten Anzahl an Betriebstests(SI)
- das erste Jahr des Betriebs (OP)

Durch die gesamte Studie hindurch wird die Kombination von FT und ST Defekten in der Gesamtheit als *Testdefekte* und die Kombination von SI und OP Defekten als *Betriebsdefekte* bezeichnet. Gelegentlich wird auch die Bezeichnung *Versagen* auftauchen. Formal gesehen ist ein Versagen eine Abweichung von dem gewünschten oder geplanten Verhalten in einem laufenden System. Alle Versagensfälle können zu einem eindeutigen Defekt in einem Modul zurückverfolgt werden. Unterschiedliche Versagensfälle, die auf denselben Defekt zurückgeführt werden können, werden nicht separat gezählt. Das bedeutet, wenn z.B. 20 OP Defekte für das Modul x gezählt werden, dann sind es diese 20 eindeutigen Defekte, die für alle beobachteten Versagensfälle (und die auf Defekte in Modul x zurückverfolgt werden können) während des ersten Betriebsjahrs verantwortlich sind.
Jeder gefundene Defekt wurde in einer Phase wie folgt klassifiziert:

a) der Defekt wurde bereits behoben;

[1] Tabelle 1 bei Norman E. Fenton, Niclas Ohlsson, 2000

b) der Defekt soll behoben werden;
c) der Defekt verlangt kein Eingreifen (bzw. wird nicht als Defekt behandelt);
d) der Defekt ist auf Installationsprobleme zurückzuführen;

In dieser Studie werden nur Defekte betrachtet, die unter Punkt b) zu klassifizieren sind. Interne Untersuchungen haben gezeigt, dass die Aufzeichnung von Defekten und ihrer Klassifikation nach den Oben aufgeführten Kategorien als zuverlässig angesehen werden kann.
Tabelle 2 zeigt eine Zusammenfassung der entdeckten Defekte in jeder Testphase und für jede Veröffentlichung:

LOC	Release n	Release n+1
<1000	23	26
1001-2000	58	85
2001-3000	37	73
3001-4000	15	38
4001-5000	6	16
5001-6000	0	6
>6000	1	2
Total	140	246

Tabelle 2: Verteilung der Defekte während verschiedener Testphasen[2]

Zweite Untersuchung

In dieser Studie wurden sechs große Open Source Softwaresysteme geprüft: Postgres (DBMS), KDE (K Desktop Environment), KOffice (Office Produktivity Suite), FeeBSD (Operating System), OpenBSD (Operating System), NetBSD (Operating System). Die Tabelle 3 fasst die Details für diese Open Source Softwaresysteme zusammen. Das älteste System ist über 10 und das jüngste System ist fünf Jahre alt. Für jedes System wird die Zahl der Subsysteme genannt und die Zahl der Defekte die in diesen Subsystemen aufgetreten sind. Z.B. enthält des Datenbanksystem Postgres 104 Subsysteme und 1401 entdeckte Defekte.

Application Name	Application Type	Start Date	Subsys. Count	Faults	Prog. Lang.
NetBSD	OS	21 March 1993	393	2451	C
FreeBSD	OS	12 June 1993	182	3264	C
OpenBSD	OS	18 Oct 1995	401	1015	C
Postgres	DBMS	9 July 1996	104	1401	C
KDE	Windowing System	13 April 1997	167	6665	C++
Koffice	Productivity Suite	18 April 1998	259	5223	C++

*Tabelle 3:*Summe der untersuchten Systeme[3]

Hier wird das Konzept der Trefferquote (Hit Rate) präsentiert, welches häufig benutzt wird, um die Leistung unterschiedlicher Hypothesen zu messen. Bei dieser Untersuchung haben die Autoren eine Top Ten Liste erstellt. In dieser Liste wurden alle Subsysteme mit Defekten gezählt. Die Art der Aufzählung basiert auf demselben Prinzip wie Hit Rate beim Cache, d.h.,

[2] Tabelle 2 bei Norman E. Fenton, Niclas Ohlsson, 2000
[3] Tabelle1 bei Ahmed E. Hassan and Richard C. Holt, 2005

jedes defekte Subsystem wird in einer Top Ten Liste aufgenommen. Die Hit Rate besagt, ob ein Subsystem mit einem Defekt in der Liste registriert wurde, oder nicht. Die Leistung der in dieser Studie vorgeschlagenen Hypothesen wird gemessen, in dem man für jedes Software System die Versionsverwaltung (CVS) automatisch ohne das Eingreifen eines Nutzers analysiert. Aufgrund der speziellen Eigenschaften bei der Code-Entwicklung wird das erste Jahr dieses Projektes ignoriert und erst die nachfolgenden 4 Jahre genutzt, um die Leistung zu messen. Für beide Hypothesen wurde die Hit Rate in Bezug auf die fixierten Defekte über einen Zeitraum von 4 Jahren aufgezeichnet.

3. Die getesteten Hypothesen und Ergebnisse

Da die Daten rückwirkend gesammelt und analysiert wurden, gab es keine Möglichkeit, kontrollierte Experimente durchzuführen. Trotzdem war der Umfang und die Qualität der Daten so groß, dass man sie für die Tests von einigen bekannten Softwaretechnik Hypothesen in Bezug auf die Verteilung und Vorhersage von Defekten und Versagen nutzen konnten. In diesem Kapitel werden die Hypothesen in fünf Kategorien gruppiert.

In Abschnitt 3.1 werden Hypothesen bezüglich des Pareto-Prinzips betrachtet. Es wird oft angenommen, dass eine kleine Anzahl von Modulen in einem System die Mehrheit der gesamten Systemdefekte beinhaltet. Die Annahme des Pareto-Prinzips für Defekte veranlasst viele Experten dazu, nach Methoden zu suchen, um defektanfällige Module in der Entwicklung- und Testphase so früh wie möglich vorhersagen zu können. Diese Methoden kannt man in zwei Kategorien einteilen: a) Nutzung von früheren Defektdaten zur Vorhersage späterer Defekt- und Versagensdaten; b) Nutzung von Produktmaßen zur Vorhersage von Defekt- und Versagensdaten. Um das Pareto-Prinzip zu untermauern, wurde eine Anzahl von Hypothesen getestet, die sich auf diese Methoden der frühen Vorhersage von defektanfälligen Modulen beziehen. Im Abschnitt 3.2 werden die Hypothesen, die sich auf Punkt 1) beziehen getestet, während in Abschnitt 3.4 Hypothesen getestet werden, die sich auf Punkt 2 beziehen. Der Abschnitt 3.3 untersucht die Hypothese in Bezug auf Häufigkeit und Zeitspanne der Veränderung der Systeme. Schließlich werden in Abschnitt 3.5 einige Hypothesen bezüglich Benchmarking Defekt-Daten getestet.

3.1 Hypothesen in Bezug auf das Pareto-Prinzip über Defekt - und Versagensverteilung

Das Pareto-Prinzip besagt allgemein, dass 20% aller möglichen Ursachen 80% der gesamten Wirkung hervor bringen. Man bezeichnet dies häufig als „20-80 Regel“, in dem Sinne, dass 80% der Defekte in 20% der Module zu finden sind. Es ist ein Erfahrungswert, dass die letzte Perfektion stets den größten Aufwand erfordert. Dies gilt gleichermaßen für die Zahl der Schreibfehler in einem Text wie für das Versagensrisiko einer technischen Anlage. Das 80-20-Prinzip kann auch so verstanden werden, dass mit 20% des Aufwandes bereits 80% der Leistung erbracht werden und für die letzten 20% der Leistung 80% des Aufwandes erforderlich sind. Grafisch wird dies durch eine anfangs sehr steile und zum Ende hin sehr flache S-Kurve im Aufwand-Ergebnis-Diagramm dargestellt.

Hypothese 1a: Die überwiegende Mehrheit der Defekte, die während der Vor-Veröffentlichungstests entdeckt werden, konzentriert sich in einigen wenigen Modulen.

Diese Hypothese wurde bestätigt. Während der Testsphase konnte man feststellen, dass 60% der gefundenen Defekte in nur 20% der Module zu finden waren. Das zeigt die Abbildung 1.

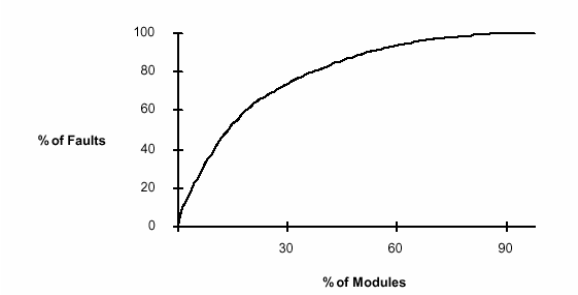


Abbildung 1: Pareto-Diagram, Prozentsatz von Modulen / Prozentsatz von Defekten[4]

Hypothese 1b: Die Module, in denen sich die Defekte, die während der Vor-Veröffentlichungstests entdeckt werden, konzentrieren, machen gleichzeitig den größten Teil des gesamten Codes aus.

Weil die Hypothese 1a bewiesen wurde, ist es wichtig die Hypothese 1b zu testen. Man glaubt, dass die Hypothese 1a einfach zu erklären ist, auf Grund dessen, dass ein kleiner Anteil von Modulen alle Defekte beinhaltet, die tatsächlich den größten Teil des Systems ausmachen. In dieser Studie konnte diese Hypothese 1b nicht bewiesen werden. Man erhielt folgende Ergebnisse: 20% von Modulen die 60% von Defekten beinhalteten, machten nur 30% der Systemgröße aus.

Hypothese 2a: Eine kleine Anzahl von Modulen enthält die meisten Betriebsdefekte (gemeint sind Versagen wie man sie in Phasen SI und OP beobachtet hat).

Die Ergebnisse zeigten, dass 10% der Module für 100% des Versagens verantwortlich sind. Somit wurde diese Hypothese bestätigt.

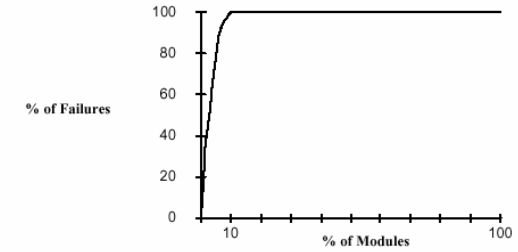


Abbildung 2: Pareto-Diagram, Prozentsatz von Modulen / Prozentsatz des Versagens[5]

Hypothese 2b: Eine kleine Anzahl von Modulen enthält die meisten Betriebsdefekte, weil diese Module den größten Teil des Codes ausmachen.

Hier kommt die Untersuchung zu einem Ergebnis, das dem zur Hypothese 1a vergleichbar ist. Man würde erwarten, dass die Bestätigung der Hypothese 2a die Tatsache erklärt, dass ein kleiner Anteil von Modulen alle Versagen beinhaltet, der tatsächlich den größten Teil des Systems ausmacht. Für diesen Fakt wurde keine Bestätigung gefunden. Im Gegenteil, es wurde sogar eine Antithese bestätigt:

„Die meisten Betriebsdefekte wurden durch Defekte in kleinen Teilen des Codes verursacht“.

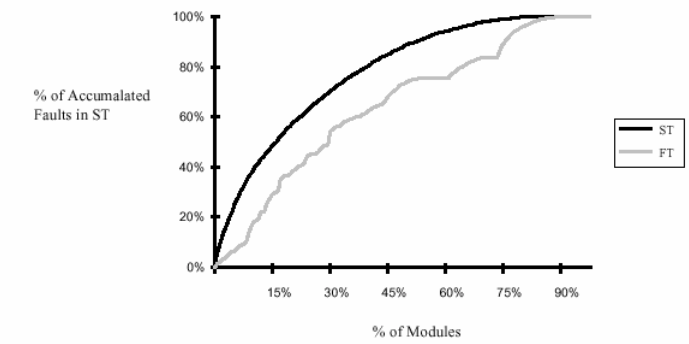
[4] Abbildung 1 bei Norman E. Fenton, Niclas Ohlsson, 2000
[5] Abbildung 2 bei Norman E. Fenton, Niclas Ohlsson, 2000

100% der Betriebsdefekte befanden sich in Modulen mit nur 12% der Systemgröße. Damit wurde bewiesen, dass die Größe in keiner signifikanten Weise die Anzahl der Defekte erklären kann. Häufig aber wurde als Grund dafür, dass eine kleine Anzahl von Modulen für die meisten Defekte verantwortlich ist, angenommen, dass diese Defektanfälligen Module disproportional groß sind und somit den größten Teil des Systems ausmachen. In dieser Studie wurde nachgewiesen, dass diese Annahme falsch ist.

3.2 Hypothesen in Bezug auf den Gebrauch von früheren Defektdaten dienen der Vorhersage von Defekten und Versagen (im Modullevel)

Hypothese 3: Ein häufigeres Auftreten von Defekten in Funktionstests (FT) ist verknüpft mit einem häufigeren Auftreten von Defekten in Systemtests (ST).

Die Ergebnisse, die man bei der Untersuchung der Hypothese 3 bekommen hat, geben keine klare Bestätigung dieser Hypothese.
In bestimmten Modulen treten 50% der Defekte während des Systemstests auf. Diese Module sind auch für 37% der Defekte während des Funktionstests verantwortlich. Schwachpunkte der Hypothese resultieren aus folgenden Ergebnissen: 10% von meist defektanfälligen Modulen während Systemtests sind verantwortlich für 38% von Defekten in Systemtests. Aber 10% meist defektanfälliger Module während Funktionstests sind verantwortlich für 17% von Defekten in Systemtests. Der Fall tritt in 75% von Modulen auf. Das bedeutet, dass fasst 29% der Defekte während der Systemstests auf andere Art und Weise erklärt werden müssen.



Abbidung 3: Akkumulierter Prozentsatz der absoluten Defektzahl, während des Systemtests, wenn die Module nach Zahl der Defekte in Systemtest und Funktionstests geordnet würden.[6]

Hypothese 4: Ein häufigeres Auftreten von Defekten während der Vor-Veröffentlichungstests ist verknüpft mit einem häufigeren Versagen im Betrieb.

Diese Hypothese besagt, dass relativ kleine Teile von Systemmodulen mit meisten Defekten die Defektanfälligkeit in Vor- und Nach-Veröffentlichungen nachweisen.

- In Veröffentlichung n treten 93% der während Vor-Veröffentlichungstests entdeckten Defekte in solchen Modulen auf, die später keine Betriebsdefekte aufweisen. Das bedeutet andererseits, 100% der später im Betrieb festgestellten Defekte treten in jenen Modulen auf, die in den Vor-Veröffentlichungstests lediglich 7% der Defekte beinhalteten.

[6] Abbildung 3 bei Norman E. Fenton, Niclas Ohlsson, 2000

- In der Veröffentlichung n+1 wurde eine viel größere Zahl von Betriebsdefekten beobachtet. Gleichwohl stellte sich ein ähnliches Phänomen ein, wie in Veröffentlichung n. 77% der Vor-Veröffentlichungsdefekte traten in Modulen auf, die später keine Betriebsdefekte aufwiesen. Dies bedeutet wiederum im Umkehrschluss, dass 23% aller Vor-Veröffentlichungsdefekte in Modulen auftraten die später auch Betriebsdefekte aufwiesen.

Als spezifisch erscheint, dass Module mit hoher Defektdichte in Vor-Veröffentlichungs-Untersuchungen eine niedrige Defektdichte in der Nach-Veröffentlichungsphase aufweisen und umgekehrt. Diese Ansicht basiert zumeist auf der Annahme, dass einige Module „wirklich schwierig" sind und dies auch während der ganzen Test- und Betriebsphase bleiben werden.
Dieses Resultat verändert die Perspektive für die allgemein verwendeten Begriffe der Softwaremaße und der Defektdichte. Wenn man die Module finden will, die während des Betriebes defektanfällig werden, sollte man alle Module ignorieren, die während der Tests defekt-anfällig waren. In Wirklichkeit besteht die Gefahr in der Annahme, dass die vorhandenen Daten einen Beweis für eine kausale Beziehung geben könnten. Die Daten, die hier zu beobachten sind, können dadurch erklärt werden, dass Module in denen während der Tests nur wenige Defekte entdeckt wurden, einfach nicht gründlich getestet wurden. Die Module, die während der Tests eine große Zahl an Defekten zeigten, wurden wahrscheinlich extrem intensiv durchgetestet. Die Erklärung dieses Falles liegt in den Testanstrengungen begründet.

Die Ergebnisse zu Hypothese 4 stellen die gesamte Art und Weise in Frage, in der Software-Komplexitäts-Matrizen genutzt und validiert werden. Das ultimative Ziel von Komplexitäts-Matrizen ist die Vorhersage von defektanfälligen Modulen in der Nach-Veröffentlichung. Die meisten Validations-Studien von Komplexitäts-Matrizen betrachten eine Matrize als gültig, wenn sie mit der Defektdichte in der Vor-Veröffentlichung übereinstimmt. Die Resultate der hier verwendeten Studie zeigen, dass gültige Matrizen im Grunde schlecht in der Hervorsage für das sind, was sie vorhersagen sollten. Die Ergebnisse dieser Hypothese lassen es deutlich als Gefahr erscheinen, die Defektdichte als eine Messeinheit für Softwarequalität zu nehmen. Wenn die Defektdichte im Sinne von Vor-Veröffentlichungsdefekten (was sehr üblich ist) gemessen wird, dann wird diese Messung auf dem Modullevel nichts über die Qualität der Module aussagen. Ein hoher Wert ist dann eher ein Indikator für intensives Testen als für schlechte Qualität.

3.3 Hypothesen in Bezug auf Häufigkeit und Zeitspanne, in der Veränderung der Subsysteme vorgenommen werden

Hypothese 5: Je öfter die Subsysteme verändert wurden, desto höher ist die Wahrscheinlichkeit, dass sie Defekte beinhalten.

Diese Hypothese beschäftigt sich mit Subsystemen, die seit Anfang des Projektes am häufigsten verändert worden sind. Sie besagt, dass die am meisten veränderten Teilsysteme dazu tendieren, komplex zu werden.

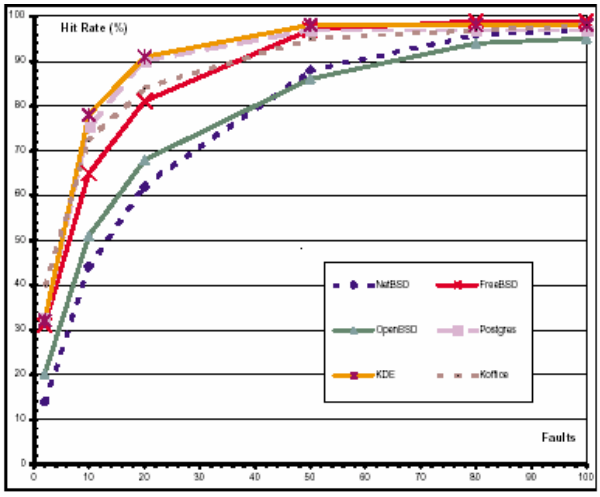


Abbildung 4: Prozentsatz von Hit Rate und Defekten für die Hypothese 5[7]

Abbildung 4 zeigt das Wachstum der Hit Rate bei Variation der Größe der Top List. Je öfter die Subsysteme verändert worden sind, desto mehr Defekte beinhalten sie. Das ist ein Beweis der Hypothese 5.

Hypothese 6: Erst vor kurzem veränderte Subsysteme weisen eine erhöhte Defektanfälligkeit auf.

Diese Hypothese prüft die Subsysteme, die erst vor kurzem verändert worden sind. Der Gedanke, der hinter dieser Hypothese steht, besagt, dass die erst vor kurzem veränderten Subsysteme meistens defektanfällig sind. In den Subsystemen, die seit längerer Zeit nicht mehr verändert worden sind, ist es unwahrscheinlich, Defekte zu finden.

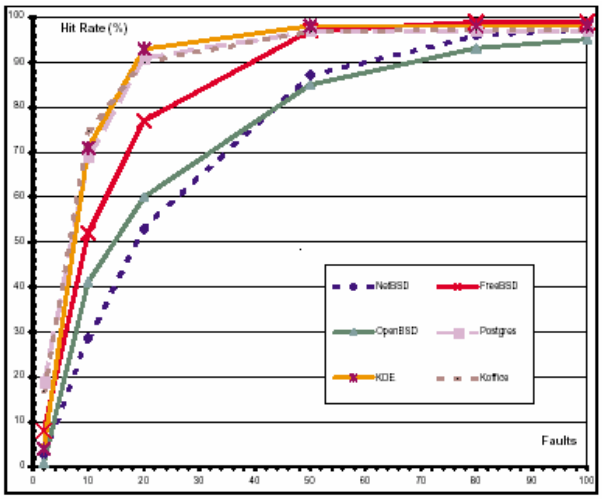


Abbildung 5: Prozentsatz von Hit Rate und Defekten für Hypothese 6[8]

Die Ergebnisse des Testes zu Hypothese 6 zeigen, dass die Hit Rate ein logarithmisches Wachstum aufweist, wenn die Größe der Top List zunimmt. Das ist ein Zeichen dafür, dass die Systeme, die erst vor kurzem verändert worden sind, tatsächlich defektanfälliger sind.

[7] Abbildung 2 bei Ahmed E. Hassan and Richard C. Holt, 2005
[8] Abbildung 3 bei Ahmed E. Hassan and Richard C. Holt, 2005

3.4 Hypothesen über Größenmaße in der Defektvorhersage

Hypothese 7: Aus einem einfachen Größenmaß (LOC) des Moduls lässt sich eine Vorhersage über die Defektanfälligkeit ableiten.

Hypothese 7a: Kleine Module sind weniger defektanfällig als größere Module.

Hypothese 7b: Ein Größenmaß eines Moduls (LOC) ist ein guter Indikator für die Anzahl von Defekten, die in Vor (Nach)- Veröffentlichungstests entdeckt werden.

Hypothese 7c: Ein Größenmaß (LOC) ist ein guter Indikator für die Defektdichte eines Moduls, die im Rahmen einer Vor (Nach)- Veröffentlichungsstudie festgestellt wird.

LOC – „Lines of Code“ - dient zur Beschreibung der Größenmaße von Programmen und wird in der Regel auf den Quellcode angewendet. Messwert ist die Anzahl von Programmzeilen, die eventuell durch das Weglassen von leeren Zeilen und Kommentarzeilen, eventuell auch Deklarationszeilen, modifiziert werden kann. Die Programmgröße ist eine wesentliche Grundlage für viele Vorhersagen, unter anderen über Zeit, Personal, Kosten, Speicherplatz etc.. Sie dient auch zur Berechnung zahlreicher indirekter Maße, wie Defektdichte, Strukturiertheitsgrad etc..
Hypothese 5a untermauert die strukturierte und Objektorientierte Programmierung. Sie beinhaltet die Idee, dass kleinere Module besser zu entwickeln, testen und warten sein sollten. Das ist der Grund, warum sie weniger Betriebsdefekte aufweisen. Andererseits ist auch bekannt, dass das Ziel, sehr kleine Module herzustellen, die Gesamtkomplexität nicht reduziert, sondern lediglich verlagert in Richtung Interface/Kommunikation.
Eine kleine Anzahl von relevanten empirischen Studien, die bis her erschienen sind, brachten keine eingängigen Ergebnisse zu Beziehungen zwischen Maßen und Defektdichte. Die Forschungen von Basil und Pericone (1984) weisen darauf hin, dass die Defektdichte mit den Modulmaßen immer geringer wird. Ihre Erklärung war, dass die große Anzahl von Interfacedefekten sich über alle Module ausdehnt. In Widerspruch zu den Erkenntnissen von Basil und Pericone (1984) wurden in der hier vorgenommenen Studie keine relevanten Trends aufgedeckt.
Um die Differenz näher zu prüfen, werden die Ergebnisse der Studie von Basil und Pericone einer Eingehendären Untersuchung unterzogen. Aus dieser Studie stammt die Tabelle 3, die die Anzahl von Modulen mit einer bestimmten Anzahl von Defekten angibt. Zudem zeigt diese Tabelle unterschiedliche Typen von Modulen und die Häufigkeit ihres jeweiligen Vorkommens. Bei der Analyse von Norman und Ohlsson [1] ist im Vergleich zu der Analyse von Basil und Pericone (1984) zu beobachten, dass bei ersteren der Anteil der Module mit Defekten und der Anteil von neuen Modulen größer ist. Im nächsten Schritt der Analyse werden die neuen Module bei Norman und Ohlsson [1] ausgeschlossen. Diese Module sind generell größer als jene aus der Studie von Basil und Pericone (1984), dieses Phänomen haben Norman und Ohlsson [1] näher betrachtet.

	Release n			Release n+1			
Fault	Mod	New	Percent modified modules	Mod	New	Splitted	Percent modified modules
0	9	0	7	15	3	0	7
1	5	3	4	16	1	0	7
2	12	0	9	18	2	0	8
3	10	0	8	13	0	0	6
4	8	0	6	12	1	0	5
5	12	1	9	7	0	0	3
6	3	1	2	14	1	0	6
7	4	0	3	5	0	0	2
8	7	0	5	5	0	1	2
9	8	2	6	13	0	1	6
10	5	0	4	6	2	0	3
11 to 15	17	1	13	24	1	0	11
16 to 20	4	0	3	14	2	3	6
21 to 25	3	0	2	21	0	0	9
26 to 30	7	0	5	9	0	1	4
31 to 35	5	0	4	8	0	1	4
36 to 40	2	0	2	6	0	0	3
>40	9	2	7	18	0	2	8

Tabelle 4: Anzahl der von Defekten betroffenen Modulen in Veröffentlichung n (140 Module, 1815 Defekten) und Veröffentlichung n+1 (246 Module, 3795 Defekten). [9]

Der Vergleich zwischen LOC und der Anzahl der Defekte in der Vor- und Nach-Veröffentlichungsphase lässt keinen eindeutigen Hinweis für diesen in der Veröffentlichung n+1 behaupteten Trend zu. Als Basili und Pericone keinen Trend beobachten konnten, kalkulierten sie die Anzahl von Defekten pro 1000 ausführbare LOC. Tabelle 5 zeigt die Ergebnisse dieser Studie.

	Release n		Release n+1	
Module size	Frequency	Faults/1000 Lines	Frequency	Faults/1000 Lines
500	3	1.45	6	13
1000	15	4.77	17	6
1500	32	5.24	35	5
2000	24	6.32	41	7
2500	14	5.88	34	5
3000	22	5.74	37	5
3500	11	7.83	18	7
>3500	9	7.38	42	8

Tabelle 5: Defekte/1000 Lines of Code in Veröffentlichung n und n+1[10]

Oberflächlich gesehen, scheinen die Resultate in Tabelle 5 für Veröffentlichung n+1 das Ergebnis von Basili und Percone zu bestätigen. In der Veröffentlichung n+1 wird deutlich, dass die kleinsten Module die höchste Defektdichte besitzen. Allerdings ist die Defektdichte auch anderer Gruppen vergleichbar groß. In Veröffentlichung n verhalten sich die Ergebnisse genau umgekehrt zu jenen aus der Studie von Basil und Pericone (1984). Der Ansatz zur Gruppierung von Daten, der dort unternommen wurde, war irreführend. Was Basili und Percone nicht zeigen konnten, war ein einfaches Schema für die Beziehung zwischen

[9] Tabelle 3 bei Norman E. Fenton, Niclas Ohlsson, 2000
[10] Tabelle 4 bei Norman E. Fenton, Niclas Ohlsson, 2000

Defektdichte und Modulmaß. Obwohl die gesammelten Daten dieser Veröffentlichung die Resultate von Basili und Perricone zu unterstützen scheinen, kann die Abbildung nur eine sehr hohe Variation für die kleinen Module nachweisen, aber keinen Beweis dafür, dass die Modulgröße signifikanten Einfluss auf die Defektdichte hat.
Die Fähigkeit LOC zu klassifizieren, wurde in Abbildung 6 dargestellt. Das Diagram besagt, dass, obwohl frühere Analysen keine Vorhersagbarkeit anzeigten, das LOC ziemlich gut die defektanfälligen Module klassifiziert, und für die meisten defektanfällige Module (20%) viel wirksamer ist als andere Methoden.

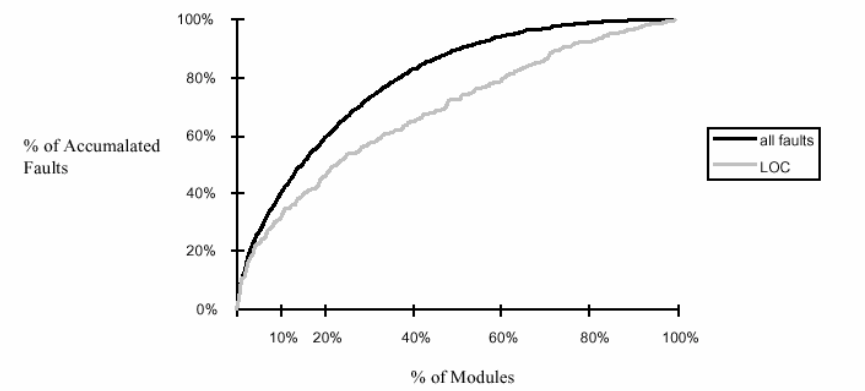


Abbildung 6: Prozentsatz der absoluten Defektzahl vs. Ordnung der Module nach LOC-Größe für Veröffentlichung n+1[11]

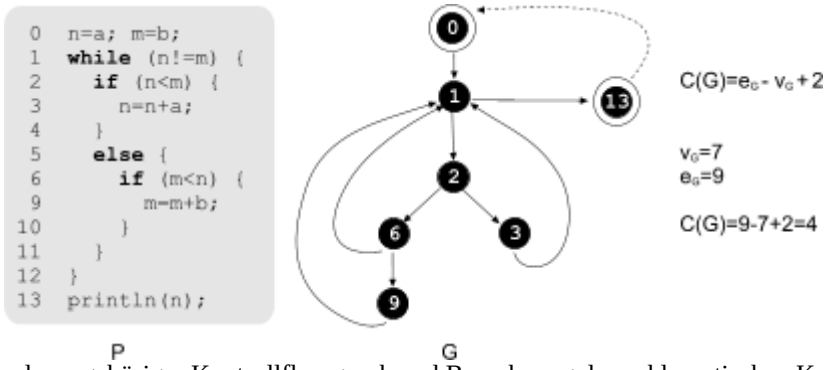
Hypothese 8: Ein Komplexitätsmaß ist ein besserer Indikator für defekt- und versagensanfällige Module als ein einfaches Größenmaß.

Beim Komplexitätsmaß handelt es sich um eine zunächst irreführende Bezeichnung, die verwendet wird, um eine Klasse von „Maßregeln“ zu beschreiben, die direkt vom Quellprogramm extrahiert werden kann. Gelegentlich können Komplexitätsmaße extrahiert werden, bevor ein Code produziert wird, z.B. wenn genaue Entwurfe für die Entwicklung eines Codes in einer graphischen Sprache SDL dargestellt werden. Die Hypothese 8 beinhaltet die Annahme, dass Komplexitätsmaße nützlich sind, weil sie anzeigen, wo die Defekte im System liegen.

Kurze Erklärung zur zyklomatischen Komplexität nach McCabe

Die zyklomatische Komplexität von McCabe ist ein klassisches Maß, das mit dem Begriff der strukturellen Komplexität assoziiert wird. Das Maß legt als Modell den Kontrollflussgraphen einer einzelnen Funktion zu Grunde und gibt die Anzahl unabhängiger Pfade durch ein Programm an. Für einen Graphen G ist die McCabe-Zahl C(G) gleich der Anzahl von Schleifen im Kontrollflussgraphen, bei dem der Endknoten "künstlich" mit dem Startknoten verbunden wird. Für Programme kann der Wert jedoch sehr leicht durch Zählen der Entscheidungen gebildet werden. Die folgende Abbildung illustriert die Berechnung der McCabe-Zahl. Mit v_G ist die Anzahl von Knoten des Graphen, mit e_G die Anzahl der Kanten gemeint.

[11] Abbildung 9 bei Norman E. Fenton, Niclas Ohlsson, 2000



Programmcode, zugehöriger Kontrollflussgraph und Berechnung der zyklomatischen Komplexität[12]

Der Kontrollfluss wird betrachtet, um Aussagen darüber machen zu können, ob und inwiefern die Software-Module zu testen und zu warten sind. Ein Modul ist gut zu testen, wenn es wenige, gut überschaubare Testfälle gibt. Es ist gut zu warten, wenn seine Funktionsweise leicht verstanden werden kann. Beides wird durch "einfache" Strukturen im Kontrollfluss begünstigt. Insofern bezieht man sich hier auf die Verständlichkeit. Insofern ist die McCabe-Zahl tatsächlich ein gültiger Prädiktor für die Testbarkeit oder Wartbarkeit. Bei der Aufzeichnung der zyklomatischen Komplexität sowie der Vor- und Nach-Veröffentlichungsdefekte für die Veröffentlichung n+1 ist eine Reihe interessanter Trends zu beobachten. Die komplexesten Module scheinen defektanfälliger in der Vor-Veröffentlichungs-Phase zu sein, weisen dann in der Nach-Veröffentlichungs-Phase jedoch kaum noch Defekte auf. Die meisten defektanfälligen Module in der Nach-Veröffentlichungs-Phase scheinen die weniger komplexen Module zu sein. Verantwortlich dafür ist die Art der Verteilung der Testanstrengungen über die Module. Die Module, die komplexer zu sein scheinen, werden mit größerer Vorsicht behandelt, als die einfacheren Module. Die vergleichende Betrachtung einerseits der zyklomatischen Komplexität andererseits der Gesamtheit aller Defekte zeigt, dass es sich bei der zyklomatischen Komplexität tatsächlich um einen Indikator zu vorhersage von Defekten handelt. Die Ergebnisse offenbarten, dass sehr defektanfällige Module während der Vor-Veröffentlichungs-Phase fast keine Nach-Veröffentlichungsdefekte aufweisen. Somit kommt man zu dem Schluss, dass die „Komplexität“ an sich das defektanfällige Verhalten in diesem Fall aber nicht erklären kann. Tatsächlich ist Komplexität kein bedeutend besserer Indikator für die Vorhersage von defektanfälligen und versagensanfälligen Modulen, als einfache Größenmessungen.

Während LOC (und somit auch die Komplexitäts-Matrizen) sinnvolle Vorhersage-Indikatoren für die Gesamtzahl von Defekten sind, gestatten sie nur unzulängliche Vorhersagen für die Defektdichte, nach denen man eigentlich sucht. Aber das Komplexitätsmaß ist ein guter Indikator zur Vorhersage von Modulattributen wie Verständlichkeit und Wartungs-freundlichkeit.

[12] Entnommen aus http://www.software-kompetenz.de

3.5 Hypothesen zur Beziehung zwischen Defektdichte, Qualität und Benchmarking-Daten

Trotz der beachtlichen Menge an Software weltweit, gibt es keine übereinstimmenden Festlegung darüber, was eine gute, schlechte oder durchschnittliche Defektdichte - unter bestimmten, festgelegten Bedingungen - ist. Es spricht jedoch einiges für die Annahme, dass dafür hinreichende Informationen vorliegen, zum Beispiel für kommerzielle C-Programme, für die Defekte als Betriebsdefekte definiert werden, die während der ersten 12 Monate des Gebrauches durch einen typischen Benutzer auftreten. Es wurde fast nichts darüber veröffentlicht, ob einzelne Firmen diese Art von Daten für ihre eigenen Systeme sammeln. In der Literatur wurden jedoch erste Versuche unternommen, um eine Gliederung für dieses Problem zu erstellen. Für das erwähnte Beispiel der Defektdichte in den ersten 12 Monaten des Gebrauchs in einem typischen Betriebseinsatz etwa folgendermaßen:

- kleiner als ein Defekt pro KLOC bekommt die Note seht gut (gewöhnlich nur erzielt durch Firmen mit state-of-the-art Entwicklung und Testmethoden)
- zwischen 4 bis 8 Defekte pro KLOC bekommt die Note typisch
- größer als 12 Defekte pro KLOC bekommt die Note schlecht

Wenn nur die Vor-Veröffentlichungsdefekte betrachtet werden, sind 10-30 Defekte pro KLOC typisch für die kombinierten Funktions-, System- und Integrationstests. Aus Gründen, die bereits besprochen wurden, sind die hohen Werte der Vor-Veröffentlichungsdefektdichte kein Anzeichen von geringer Qualität. Deswegen ist es problematisch, von schlechter oder guter Defektdichte zu sprechen.

Hypothese 9: Es ist davon auszugehen, dass die Defektdichte in den Testphasen und der nachfolgenden Betriebsphase bei Hauptversionen eines Softwaresystems nahezu konstant bleibt.

	FT	ST	SI	OP
Rel n	3.49	2.60	0.07	0.20
Rel n+1	4.15	1.82	0.43	0.20

Tabelle 6: Defektdichte in vier Test- und Betriebsphasen[13]

Die Tabelle 6 zeigt, dass beide Untersuchungen die Hypothese bestätigen, dass die Defektdichte über alle Testphasen hinweg stabil bleibt. Die einzige Abweichung ist in der OP-Phase zu beobachten. Dieses Ergebnis suggeriert, dass der Entwicklungsprozess in Bezug auf die Defektdichte stabil und reproduzierbar ist. Diese Schlussfolgerung ist von Bedeutung für den Softwareoptimierungsprozess, der von CMM verkörpert wird. Denn eine allgemeine Annahme des *Capability Maturity Model* (CMM) ist, dass ein einheitlicher und wiederholbarer Prozess eine notwendige Vorbedingung für die Verbesserung des Prozesses ist. Damit wurde diese Hypothese bestätigt.

[13] Tabelle 5 bei Norman E. Fenton, Niclas Ohlsson, 2000

Hypothese 10: Softwaresysteme, die unter vergleichbaren Bedingungen produziert werden, weisen in der Regel eine etwa gleiche Defektdichte sowohl in der Test- als auch in der Betriebsphase auf.

Um die Hypothese zu testen, wurden die Ergebnisse dieser Fallstudie mit anderen veröffentlichten Daten verglichen. Um den Test einfach zu halten, beschränkte man die Analyse auf 2 unterschiedliche Phasen mit:

1. der Vor-Veröffentlichungsfehlerdichte und
2. der Nach-Veröffentlichungsfehlerdichte

Die Ergebnisse der beiden veröffentlichten Fallstudien werden einander in Tabelle 7 gegenüber gestellt:

	Pre-release	Post-release	All
Rel n	6.09	0.27	6.36
Rel n+1	5.97	0.63	6.60

Tabelle 7: Defektdichte während der Vor-Veröffentlichungs- und der Nach-Veröffentlichungs-Phase[14]

Interessant ist der Unterschied zwischen Vor-Veröffentlichungsdefektdichte und Nach-Veröffentlichungsdefektdichte. In beiden Veröffentlichungen ist die Größenordnung der Vor-Veröffentlichungsdefektdichte höher als die der Nach-Veröffentlichungsdefektdichte. Die wenigen erschienenen Studien, die den Unterschied zwischen Vor- und Nach-Veröffentlichungsdefektdichte deutlich machen, zeigen, dass über 10-mal mehr Defekte in Vor-Veröffentlichungsphase auftauchen.
Kitchenham et al. (1986) berichten über ein höheres Verhältnis von Vor- und Nach-Veröffentlichung. Laut dieser Studie deckt die Kombination von kontrollierten und nicht kontrollierten Codes eine Vor-Veröffentlichungsdefektdichte von ca. 16 pro KLOC (thousands of lines of code) auf und eine Nach-Veröffentlichungsdefektdichte von ungefähr 0,3 pro KLOC.
Aus diesen Ergebnissen lässt sich schlussfolgern, dass 10 bis 30 mal so viele Defekte in der Vor-Veröffentlichungs-Phase auftreten wie in der Nach-Veröffentlichungs-Phase.

[14] Tabelle 6 bei Norman E. Fenton, Niclas Ohlsson, 2000

4. Schlussfolgerungen

In den hier vorgestellten Fallstudien wird die differenzierte Datensammlung als Teil des regelmäßigen Konfigurationsmanagements und der Qualitätssicherung betrachtet. Diese Daten wurden benutzt, um Licht auf jene Themen zu werfen, denen eine zentrale Bedeutung in der Softwaretechnik zukommt. Über die bisher gängige Praxis der Qualitätskontrolle hinaus, hat es sich als sinnvoll erwiesen, die Defektdaten während unterschiedlicher Testphasen zu sammeln und auszuwerten, um eine effizientere statistische Prozesskontrolle für die Softwareentwicklung zu ermöglichen. Dies entspricht beispielsweise auch der Vorgehensweise der Firma Hitachi. So lassen sich Defektprofile erstellen, die eine genauere Vorhersage von Defekten und Versagen ermöglichen.

Ein weiterer wichtiger Grund dafür, die unterschiedlichen Defektdaten zu sammeln, ist es, die Effektivität verschiedener Teststrategien abschätzen zu können. In dieser Arbeit wurden ausführliche Beispiele von Defekt- und Versagensuntersuchungen genutzt, um eine Reihe von bekannten Hypothesen aus der Softwaretechnik zu überprüfen. Da jedes Unternehmen aus der Softwareindustrie sich um die beste Qualität unter den gegebenen ökonomischen Bedingungen bemüht, ist das Wissen über Defektabschätzung (was viel Arbeit, Zeit und Geld einsparen hilft) für die Führungskräfte von großen Bedeutung. Einen konkreten Hinweis, wie das Budget verteilt werden könnte, vermitteln die Untersuchungsergebnisse zu den vorgestellten Hypothesen.

Der am wenigsten überraschende Nachweis gelingt dieser Studie mit der Untermauerung der beiden Pareto-Prinzipien (1a und 2a). Bestätigt wurde auch die These, dass die Häufigkeit der Veränderung der Module Einfluss auf die Defektanfälligkeit der Module hat. Das bedeutet, dass erst vor kurzem veränderte Module defektanfälliger sind. Ein weiterer sicherer Indikator für die Anzahl der Defekte ist ein einfache Größenmaß (LOC) des Moduls.

Allerdings ist es nicht der Fall, dass die Größe in signifikanter Weise die Anzahl der Defekte erklärt. Auch ist es nicht der Fall, dass Komplexität der Matrix das defektfällige Verhalten erklären könnte. Eine Gruppe von Modulen, die besonders in der Vor-Veröffentlichungsphase defektanfällig ist, wird nicht die gleiche Gruppe von Modulen sein, die sich in der Nach-Veröffentlichungsphase als defektanfällig erweisen wird.

In der Untersuchung von Fenton und Ohlsson [1] wird gezeigt, dass es immer möglich ist, ein System in einer Umgebung zu konstruieren, welche den aufgestellten Regeln widerspricht. Als gutes Beispiel kann hier die Hypothese 7 gelten, die besagt, dass größere Module eine geringere Defektdichte haben, als kleinere. Abgesehen davon, dass man keinen eindeutigen Beweis dafür findet und auch Schwächen in der Studie entdeckt, ist es unangebracht, daraus eine Regel für die Defektabschätzung abzuleiten. Allein die Erhöhung der Anzahl der Tests wurde zu einer Widerlegung dieser Regel führen. Wird ein Modul nicht getestet oder benutzt, wird darin auch kein Defekte oder Versagensfall entdecken.

Um es noch einmal zu betonen: Bei der Verbindung zwischen Größe und Defektdichte handelt es sich nicht um eine kausale Beziehung. Aus diesem Grunde werden vollständigere Modelle empfohlen, die die empirischen Beobachtungen um weitere erklärende Faktoren erweitern; insbesondere um die Testintensität und den Betriebsgebrauch.

Zur Reichweite der hier vorgelegten Arbeit lässt sich feststellen, dass damit kein exakter Pfad zur optimalen Defekt- und Versagensvorhersage vorgelegt werden kann. Gleichwohl können

mit den Ergebnissen Fehler und Resoursenverschwendung in einem begrenzten Umfang vermieden werden. Die verifizierten Hypothesen erlauben eine rationale und pragmatisch zu handhabende Annäherung an Standart für eine optimierte Defektabschätzung im Bereich der Softwareentwicklung.

5. Literaturgrundlage

[1] Norman E. Fenton, Niclas Ohlsson, “Quantitative Analysis of Faults and Failures in a Complex Software System”, IEEE Transactions of Software Engineering, (Vol. 26, No. 8), pp. 797-814, August 2000.

[2] Ahmed E. Hassan and Richard C. Holt, "The Top Ten List: Dynamic Fault Prediction" Software Architecture Group (SWAG), School of Computer Science, University of Waterloo,Canada, 2005

[3] Basili VR und Perricone BT, “Software Errors and Compexity: An Empirical Investigation”, Communications of the ACM 27(1), pp.42-52, 1984.

[4] http://www.software-kompetenz.de

[5] Kitchenham BA, Kitchenham AP, Feelows JP, “The effects of inspections on software quality and productivity”, ICL Tech J, 112-122, May 1986.

[6] Peter Liggesmeyer, “Software-Qualität: Testen, Analysieren und Verifizieren von Software”, Spektrum Akademischer Verlag, Heidelberg, Berlin, 2002