Eine Untersuchung über Korrekturkosten von Software-Fehlern

Tilmann Hampp, Markus Knauß
Abteilung Software Engineering, Institut für Softwaretechnologie
Universität Stuttgart
www.iste.uni-stuttgart.de/se

1 Einführung und Motivation

Verbesserungen in der Software-Entwicklung basieren auf der Annahme, dass Fehler günstiger zu beheben sind, wenn sie möglichst früh entdeckt werden [Boe76,Boe87]. Erfahrungsberichte bestätigen diesen Zusammenhang [Bas84, Hum95, Kan03, Shu02]. Aktuelle oder detaillierte Zahlen liegen aber nicht vor. Darum ist zum Beispiel unklar, ob und wie sich ein objektorientiertes Vorgehen auswirkt und welche anderen Faktoren den Korrekturaufwand bestimmen.

Darum untersuchen wir den Korrekturaufwand einzelner Fehler. Wir prüfen, ob die Korrektur eines Fehlers aufwändiger wird, je länger der Fehler unentdeckt bleibt. Die Dauer, über die der Fehler unentdeckt bleibt, bezeichnen wir als Latenzzeit. Fehler werden bei bestimmten Tätigkeiten gemacht, beispielsweise entstehen Spezifikationsfehler bei der Analyse und Spezifikation: Die Tätigkeit bestimmt die Abstraktionsebene des Fehlers. Prüfungen entdecken nur Fehler, die auf einer bestimmten Abstraktionsebene oder darunter gemacht wurden [Frü06]: Spezifikationsfehler können nur durch ein Review der Spezifikation oder wieder ab dem Systemtest entdeckt werden; im Gegensatz dazu werden Codefehler bereits im Unittest entdeckt. Wir vermuten als weiteren Einfluss die Fehlerschwere und die Zahl der zur Korrektur betrachteten oder geänderten Software-Einheiten. Aus diesen Überlegungen haben wir die folgenden Hypothesen abgeleitet:

H1: Prüfungen entdecken nur Fehler, die auf der gleichen oder einer tieferen Abstraktionsebene der Entwicklung gemacht wurden.

H2: Je länger ein Fehler unentdeckt bleibt, desto aufwändiger ist seine Korrektur.

H3: Je schwerwiegender ein Fehler ist, desto aufwändiger ist seine Korrektur.

H4: Je mehr Software-Einheiten zur Korrektur eines Fehlers betrachtet und geändert werden müssen, desto aufwändiger ist seine Korrektur.

Diese Hypothesen untersuchen wir mit Fehlerdaten aus studentischen Projekten. Im Folgenden beschreiben wir das Software-Praktikum, aus dem die Projekte stammen (Abschnitt 2), und die Untersuchung (Abschnitt 3), denen wir diese Hypothesen unterziehen. Die Ergebnisse sind in Abschnitt 4 beschrieben, Abschnitt 5 enthält die Bewertung und unser Fazit.

2 Software-Praktikum

Im Software-Praktikum des Studiengangs Softwaretechnik an der Universität Stuttgart [Lud01] führen alle Studierenden im 3. und 4. Semester ein vollständiges Softwareprojekt in Teams mit jeweils 3 Teilnehmern durch. Das Praktikum beginnt typisch Mitte Februar und endet im Juli. Das Software-Praktikum wird von zwei Mitarbeitern betreut. Ein weiterer Mitarbeiter übernimmt die Rolle des Kunden. Falls ein hochschulexterner Kunde beteiligt ist, übernimmt er die Rolle eines internen Kundenvertreters.

Vorgegeben ist, welche Entwicklungsphasen durchlaufen, welche Dokumente erstellt und welche Prüfungen durchgeführt werden müssen. Vorgegebene Entwicklungsphasen sind Planung, Spezifikation, Entwurf, Implementierung und Test. Die in den Phasen entstehenden Dokumente sind der Projektplan, die Spezifikation, der Entwurf, die Implementierung, die Testdokumentation mit Testfällen und die Protokolle der durchgeführten Prüfungen. Durchzuführende Prüfungen sind das Review der Spezifikation, ein Walkthrough durch den Entwurf und der Test der Implementierung sowie ein Abnahmetest mit den Betreuern und dem Kunden.

Die Betreuer geben Meilensteine mit Termin für jede Phase vor. Der Meilenstein ist erreicht, wenn der Betreuer die Abgabe abnimmt. Die Terminvorgaben sind notwendig, um zum Beispiel Reviews in einem gegebenen Zeitraum durchzuführen und das Software-Praktikum zu einem definierten Zeitpunkt zu beenden. Die Phasen werden streng sequentiell durchschritten. Iterationen gibt es im Software-Praktikum nicht.

Die Aufgabe im Software-Praktikum 2007 war die Entwicklung eines Werkzeugs für die Verwaltung, Ausführung und Dokumentation von Testfällen in Java. An diesem Software-Praktikum haben 73 Studierende teilgenommen, die in 25 Teams eingeteilt waren. 21 dieser Teams haben die Anforderungen des Software-Praktikums erfüllt und eine Lösung der ge-

stellten Aufgabe geliefert. Eine dieser Lösungen ist das Werkzeug „Justus“, das auf der Webseite [Jus08] zu finden ist.

3 Untersuchung

Wir prüfen die Hypothesen aus Abschnitt 1 mit Daten aus dem Software-Praktikum 2007. Dazu mussten die 73 teilnehmenden Studierenden detaillierte Daten über Art und Schwere von Fehlern erheben, die sie bei der Softwareentwicklung gemacht, entdeckt und korrigiert haben. Für die Fehlersammlung wurde den Studenten ein Werkzeug zur Verfügung gestellt, die Sammlung erfolgte pro Team.

Damit die Teams die Fehlerdaten möglichst einheitlich erheben, haben wir eine Anleitung zur Identifikation von Fehlern und zur Klassifikation der Fehlerart und Fehlerschwere [Ham07] entwickelt. Um die Erhebung für die Teilnehmer zu vereinfachen, haben wir das Werkzeug JDefectCollector implementiert.

3.1 Klassifikationen

Fehler sind als Abweichung von den Anforderungen definiert. Prüfungen können Fehler sichtbar machen. Im Software-Praktikum durchgeführte Prüfungen sind in Tabelle 1 aufgelistet. Um den Entdeckungszeitpunkt eines Fehlers zu erheben, wurden die Prüfungen der jeweiligen Phase zugeordnet. Als Entdeckungszeitpunkte ergaben sich Spezifikationsreview, Spezifikationsabnahme, Entwurfsreview, Entwurfsabnahme, Handbuchreview, Handbuchabnahme, Codeabnahme, Unittest, Unittest-Abnahme, Integrations- und Systemtest, Systemtest-Abnahme und Abnahme. Tabelle 2 zeigt die Klassifikation nach Fehlerschwere und Tabelle 3 die Klassifikation nach Fehlerart.

Prüfung	**Definition und Erhebung**
Review	Jeder Befund ist ein Fehler.
Test	Jede Abweichung von einem Soll-Resultat ist ein Fehler.
Meilenstein-abnahme	Jede Anmerkung des Betreuers oder des Kunden ist ein Fehler.
Abnahme	Jede zu korrigierende Anmerkung ist ein Fehler.

Tabelle 1: Erfassungvorschrift für Fehler

Fehler-schwere	**Definition**
Kritischer Fehler	Der Prüfling ist für den vorgesehenen Zweck unbrauchbar.
Hauptfehler	Die Nutzbarkeit des Prüflings ist merklich beeinträchtigt.
Nebenfehler	Die Nutzbarkeit ist kaum beeinträchtigt.

Tabelle 2: Klassifikation nach Fehlerschwere

Fehlerart	**Definition**
Spezifikationsfehler	Kundenanforderungen wurden falsch, inkonsistent, unklar oder gar nicht spezifiziert.
Entwurfsfehler	In der Spezifikation geforderte Daten werden nicht oder falsch bearbeitet, gespeichert oder ausgegeben. Schnittstellen sind falsch entworfen oder nicht realisierbar.
Implementierungsfehler	Falsch oder nicht implementierte Teile des Entwurfs oder Fehler in der Implementierung.
Handbuchfehler	Produktmerkmale, Bedienungen oder Installation sind im Vergleich zur Spezifikation falsch, nicht oder unvollständig beschrieben.
Darstellungsfehler	Abweichungen von Richtlinien, Zeichensetzung, Grammatikregeln, Numerierungsfehler, . . .
Andere	Begründung erforderlich

Tabelle 3: Klassifikation nach Fehlerart

3.2 Werkzeuge

Die Fehlerdaten mussten von den Studenten kontinuierlich erfasst werden. Für die Erfassung der Fehlerdaten wurde den Teams die Anwendung JDefectCollector zur Verfügung gestellt. Abbildung 1 zeigt einen Screenshot der Anwendung. Um die erfassten Fehler der Teams möglichst vergleichbar zu machen, wurde im Erfassungsdialog ständig eine kontextsensitive Anleitung zur Fehler-Klassifizierung und -Dokumentation angezeigt.

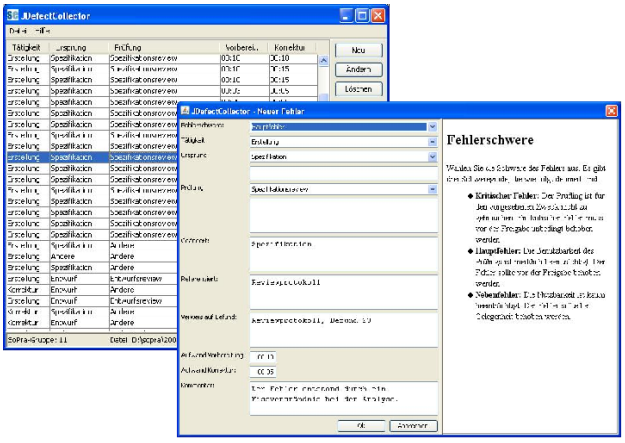

Abbildung 1: Die Anwendung JDefectCollector für die Erfassung der Fehlerdaten

Beim Erreichen externer Meilensteine am Ende der vorgeschriebenen Phasen musste jedes Team seine gesammelten Fehlerdaten abgeben. Dazu wird mit dem Werkzeug eine DDZ-Datei (Defect Data Zipped) erzeugt. In der Datei werden die Fehlerdaten im Klartext als kommaseparierte Werte gespeichert und an-

schließend gepackt. Das hat den Vorteil, dass die Daten einfach auswertbar sind und die Studenten die übermittelten Daten prüfen konnten. So konnten die Studenten sicher sein, dass nur die Fehlerdaten und keine persönlichen Daten außer der Teamnummer übertragen werden.

Für die Auswertung der gesammelten Daten wurde das Werkzeug JDefectMerger eingesetzt. Das Werkzeug sammelt alle vorliegenden DDZ-Dateien und sortiert die Fehlerdaten nach Teams und nach Meilensteinen. Außerdem werden Duplikate in den Fehlerdaten entfernt. Duplikate entstehen, weil die Fehlerdaten einer Phase bei deren Beendigung abgegeben werden müssen. Da die Fehlerdaten aber über das gesamte Projekt gesammelt werden, sind in den Fehlerdaten zu einer Phase auch die Fehlerdaten vorangegangener Phasen enthalten. Ergebnis von JDefectMerger sind Dateien im CSV-Format.

4 Ergebnisse

19 der 21 Teams haben insgesamt 1271 Fehler dokumentiert. Nach Prüfungen und nach Abnahme wurden 1103 Fehler dokumentiert (Abbildung 2). Die übrigen 168 Fehler wurden bei anderen Aktivitäten oder in den Meilensteinabgaben von den Betreuern entdeckt. Wir werten nur Spezifikations-, Entwurfs- und Codefehler aus, weil nur für diese Fehler ein Einfluss der Latenzzeit und der Abstraktionsebene vermutet wird. Andere Fehler werden nicht betrachtet, für die Auswertung bleiben 1066 Fehler (Tabelle 4).

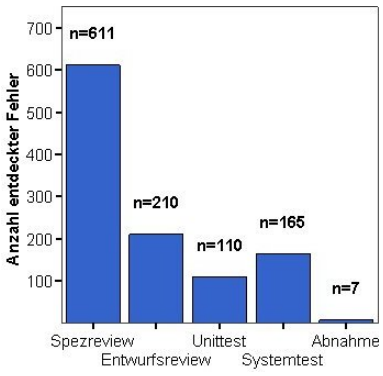

Abbildung 2: Entdeckte Fehler pro Prüfung

Wir konnten keinen Einfluss des Vorbereitungsaufwands feststellen und zeigen darum im Folgenden die Auswertung des Korrekturaufwands einschließlich Vorbereitungsaufwand.

4.1 Datenvalidierung

Weil das Werkzeug JDefectCollector zu jedem Fehler einen Zeitstempel ablegt, konnten ungewöhnliche Kombinationen aus Prüfung und Fehlerart identifiziert und korrigiert werden. Beispielsweise wurde ein Codefehler fälschlicherweise dem Entwurfsreview zugeordnet, obwohl der Codefehler zur Zeit des Unittests entdeckt wurde.

Eine erste Darstellung des Korrekturaufwands als Boxplot zeigt viele Ausreißer und Extremwerte. Von den 30 aufwändigsten Fehlern wurden 6 ausgeschlossen: In 5 Fällen ist unklar, ob die richtige Einheit (Minuten oder Stunden) verwendet wurde. In einem Fall ist die Gruppe offensichtlich nicht mit der Implementierung fertig geworden: Wesentliche Funktionen wurden erst nach der Abnahme implementiert und diese Implementierung als ein einzelner Fehler dokumentiert.

4.2 Prüfungen und Abstraktionsebene

In jeder Prüfung werden vorrangig Fehler einer bestimmten Art entdeckt (Tabelle 4).

Prüfung	Spezifikationsfehler	Entwurfsfehler	Codefehler
Spezifikationsreview	572	0	0
Entwurfsreview	1	209	0
Unittest	0	4	107
Systemtest	13	3	150
Abnahme	0	0	7

Tabelle 4: Entdeckte Fehler in Prüfungen

Spezifikationsfehler werden vorrangig im Spezifikationsreview und dann erst wieder im Systemtest entdeckt. Dies stützt die Hypothese H1. Wir erklären die wenigen spät entdeckten Spezifikationsfehler durch die intensiven Reviews. Die Teilnehmer begutachten sich gegenseitig, damit haben alle Gutachter die gleichen Anforderungen analysiert und spezifiziert – sie werden zu Experten der Anforderungen im Software-Praktikum.

Entwurfsfehler werden hauptsächlich im Entwurfsreview entdeckt. Nur wenige Entwurfsfehler sind in den Tests dokumentiert. Unsere Erfahrung ist aber, dass der Entwurf die Teilnehmer (über-)fordert. Wir erwarten darum eher viele Entwurfsfehler. Im Software-Praktikum gibt es aber keinen formalen Integrationsschritt und keinen formalen Integrationstest. Die Teilnehmer integrieren kontinuierlich während der Implementierung. Wir vermuten, dass während der kontinuierlichen Integration viele Entwurfsfehler entdeckt, aber nicht dokumentiert werden. Der Unittest enthält einen (verkappten) Integrationstest. Damit werden zwar Entwurfsfehler entdeckt, aber nur teilweise; ein Teil wird erst im Systemtest endeckt. Dies stützt die Hypothese H1.

Wir verzichten auf eine statistische Prüfung der Hypothese, weil beinahe alle Spezifikations- und Entwurfsfehler früh entdeckt werden.

4.3 Latenzzeit und Korrekturaufwand

Abbildung 3 zeigt den Korrekturaufwand pro Fehler für die verschiedenen Prüfungen, links mit, rechts ohne Extremwerte. Der Anstieg des Korrekturaufwands im Verlauf des Projekts wird deutlich.

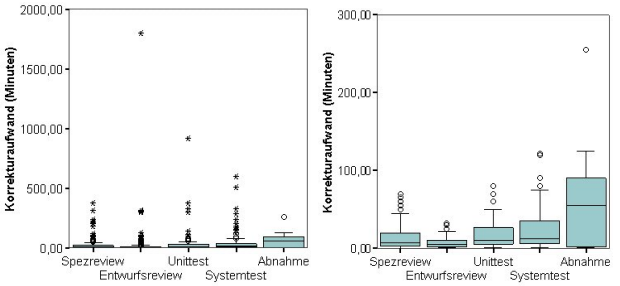

Abbildung 3: Korrekturaufwand pro Fehler

Aufwand (in Min.)	Spez.-review	Entw.-review	Unit-test	Syst.-test	Ab-nahme
Min.	0.0	1.0	1.0	1.0	1.0
1. Quartil	3.0	2.0	5.0	6.0	1.5
Median	7.0	5.0	10.0	12.0	55.0
Mittelwert	17.1	22.9	36.7	40.8	70.6
3. Quartil	20.0	10.0	26.5	35.0	90.0
Max.	375.0	1800.0	918.0	601.0	255.0
Std.-Abw.	34.6	129.4	101.9	78.4	93.0

Tabelle 5: Korrekturaufwand pro Fehler

Tabelle 5 zeigt Lage und Streuung der Korrekturaufwände. Median und Mittelwert nehmen im Verlauf des Projekts zu. Der Mittelwert ist jeweils deutlich größer als der Median. Dies zeigt den Einfluss der Ausreißer und Extremwerte.

Zur Prüfung der Hypothese H2 werden Spezifikations- und Entwurfsfehler zu den frühen Fehlern zusammengefasst, weil nur wenige dieser frühen Fehler spät entdeckt werden. Codefehler werden als späte Fehler bezeichnet. Spezifikations- und Entwurfsreview werden zusammengefasst (frühe Fehlerentdeckung im Review), Unittest und Systemtest werden gemeinsam betrachtet (späte Fehlerentdeckung im Test). Für den Hypothesentest wird die Hypothese H2 umformuliert: Es soll gezeigt werden, dass die Korrektur der frühen Fehler mit der Latenzzeit teurer wird – teurer als bei früher Entdeckung und teurer als die späten Fehler:

H2a: Die späte Korrektur früher Fehler ist aufwändiger als die frühe Korrektur.

H2b: Die späte Korrektur früher Fehler ist aufwändiger als die Korrektur später Fehler.

Tabelle 6 zeigt Lage und Streuung der Daten. Wir verwenden für den Hypothesentest den Wilcoxon-Rangsummen-Test, weil wir annehmen, dass die Korrekturaufwände nicht normalverteilt sind. Der Unterschied zwischen Median und Mittelwert und die große Streuung stützen diese Annahme. Als Signifikanzniveau wählen wir 0,05.

Aufwand (in Min.)	Frühe Fehler, entdeckt in Reviews	Frühe Fehler, entdeckt in Tests	Späte Fehler
Minimum	0.0	4.0	1.0
1. Quartil	3.0	10.0	5.0
Median	6.0	37.5	10.0
Mittelwert	18.7	131.8	32.0
3. Quartil	15.0	150.2	30.0
Maximum	1800.0	918.0	601.0
Std.-Abw.	73.2	222.7	63.4

Tabelle 6: Korrektur früher und später Fehler

Der Korrekturaufwand der frühen Fehler unterscheidet sich deutlich zwischen den früh in Reviews und den spät in Tests entdeckten Fehlern. Median und Mittelwert zeigen, dass die späte Korrektur etwa 6 mal so aufwändig ist wie die frühe Korrektur. Der Unterschied ist signifikant ($W = 3124.5, p < 0.000$). Die frühen Fehler sind im Test auch deutlich teurer zu korrigieren als die späten Fehler, etwa um den Faktor 4. Der Unterschied des Medians ist signifikant ($W = 1574, p = 0.004$). Die Nullhypothesen der Hypothesen H2a und H2b – es gibt keinen Unterschied – können verworfen werden, beide Hypothesen werden bestätigt. Die Hypothesentests bestätigen den subjektiven Eindruck (Abbildung 3) und die Erfahrungsberichte [Boe76, Bas84, Boe87, Hum95, Shu02, Kan03].

Eine detaillierte Analyse des Korrekturaufwands für verschiedene Prüfungen und Fehlerarten zeigt, dass es zwischen Codefehlern nach Unittest und nach Systemtest keinen signifikanten Unterschied gibt ($W = 7362, p = 0.259$). Wir können keine Aussage zum Korrekturaufwand nach Spezifikationsreview und Entwurfsreview treffen, weil der Mittelwert größer, der Median aber kleiner wird (Tabelle 5).

4.4 Fehlerschwere und Korrekturaufwand

Abbildung 4 und Tabelle 7 zeigen den Korrekturaufwand pro Fehler getrennt für Nebenfehler, Hauptfehler und kritische Fehler. Extremwerte sind in der Abbildung nicht dargestellt. Deutlich wird, dass die Korrektur schwerer Fehler aufwändiger ist.

Die Signifikanz des Unterschieds im Median für die unterschiedliche Fehlerschwere prüfen wir wieder mit dem Wilcoxon-Rangsummen-Test und 0,05-Signifikanzniveau für die folgenden Hypothesen:

H3a: Hauptfehler sind aufwändiger zu korrigieren als Nebenfehler.

H3b: Kritische Fehler sind aufwändiger zu korrigieren als Nebenfehler.

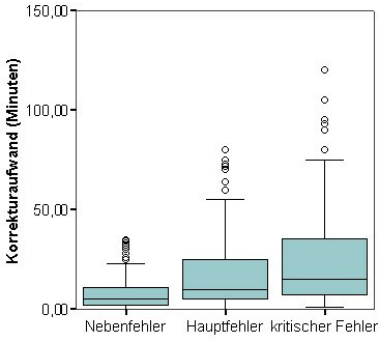

Abbildung 4: Korrekturaufwand pro Fehler

Aufwand (in Min.)	Neben-fehler	Haupt-fehler	kritischer Fehler
Minimum	0.0	0.0	1.0
1. Quartil	2.0	5.0	7.0
Median	5.0	10.0	15.0
Mittelwert	11.5	36.4	41.4
3. Quartil	11.0	25.0	35.5
Maximum	240.0	1800.0	510.0
Std.-Abw.	21.1	120.0	72.7

Tabelle 7: Korrekturaufwand pro Fehler

H3c: Kritische Fehler sind aufwändiger zu korrigieren als Hauptfehler.

Die Nullhypothese – es gibt keinen Unterschied im Korrekturaufwand – kann jeweils verworfen werden (Tabelle 8), die Hypothese wird bestätigt.

Hypothese	W	p
H3a	70659	< 0.000
H3b	21285.5	< 0.000
H3c	22162.5	< 0.000

Tabelle 8: Hypothesentest zur Fehlerschwere

Weil die Prüfung und die Fehlerart mit dem Korrekturaufwand zusammenhängen, werden die Hypothesen für jede Prüfung mit den dabei typisch entdeckten Fehlern getrennt geprüft (Tabelle 9). Die Abnahme wird nicht betrachtet, weil nur wenige Fehler entdeckt und korrigiert werden. Signifikante Ergebnisse mit $p < 0{,}05$ sind unterstrichen.

Die Nullhypothese – es gibt keinen Unterschied – kann nur in den Reviews abgewiesen werden, aber bereits im Entwurfsreview kann kein Unterschied mehr zwischen Hauptfehlern und kritischen Fehlern gezeigt werden. In den Tests gibt es eine Ausnahme: Im Unittest ist der Unterschied zwischen kritischen Fehlern und Nebenfehlern signifikant, der p-Wert ist aber relativ hoch.

Als Grund für den Einfluss der Fehlerschwere in den Reviews vermuteten wir zuerst, dass die Gutachter den erwarteten Korrekturaufwand in die Gewichtung eines Fehlers einfließen lassen. Eine Durchsicht der Kommentare zu den Fehlern konnte die Vermutung aber nicht bestätigen. Dass es keinen unterschiedlichen Korrekturaufwand in den Tests gibt, widerspricht den Erfahrungen in der Industrie [Kan03].

Hypothese für Prüfung		W	p
Spez.-review und Spezifikationsfehler	H3a	22692.5	< 0.000
	H3b	4492.5	< 0.000
	H3c	4369	0.001
Entwurfsreview und Entwurfsfehler	H3a	2494	0.016
	H3b	703.5	< 0.000
	H3c	308	0.079
Unittest und Codefehler	H3a	553	0.487
	H3b	442.5	0.021
	H3c	507	0.114
Systemtest und Codefehler	H3a	1437	0.076
	H3b	619.5	0.575
	H3c	1162.5	0.344

Tabelle 9: Hypothesentest zur Fehlerschwere nach Prüfungen

4.5 Software-Einheiten und Korrekturaufwand

Die Hypothese H4 können wir nicht prüfen. Die Anforderungen an die Datenerhebung waren ungenau, so dass die Teilnehmer nicht konsistent dokumentiert haben, welche Einheiten referenziert oder geändert wurden. Weil die Definition der Fehlerart aber enthält, welches Software-Artefakt betroffen ist, weisen die Ergebnisse auf eine Bestätigung hin: Entwurfsfehler werden teurer als Codefehler, weil der Entwurf und mehrere Codeklassen zur Korrektur überarbeitet werden müssen. Spezifikationsfehler werden noch teurer, weil zur Korrektur die Spezifikation, meist der Entwurf und mehrere Codeklassen überarbeitet werden müssen.

5 Bewertung und Fazit

Mit den Daten aus dem Software-Praktikum kann gezeigt werden:

Frühe Fehler werden entweder sofort durch ein Review oder erst am Ende der Entwicklung im Systemtest entdeckt (Hypothese H1). Durch die kontinuierliche Integration wird aber bereits im Unittest ein Teil der Entwurfsfehler entdeckt.

Die Korrektur eines Fehlers ist nach Tests teurer als nach Reviews. Wenn frühe Fehler spät entdeckt werden, dann sind sie teuer zu korrigieren. Die Korrektur ist dann auch teurer als die Korrektur später Fehler. Die Hypothese H2 kann auch in einem studentischen Projekt gezeigt werden.

Die Fehlerschwere spielt bei der Korrektur nach den Reviews eine Rolle (Hypothese H3). Im Test konnten wir diesen Zusammenhang nicht zeigen.

Die Korrekturaufwände streuen stark. Ausreißer und Extremwerte sind typisch – einzelne Fehler können sehr viel teurer werden als erwartet.

Die Untersuchung ist kein kontrolliertes Experiment. Weil wir aber Hypothesen überprüfen, betrachten wir die interne und externe Validität: Die interne Validität ist durch die unkontrollierten Variablen der Untersuchung eingeschränkt. Die Teilnehmer zeigen unterschiedliche Leistungen. Sie haben aber die gleichen Lehrveranstaltungen besucht und Reviews und Tests geübt; außerdem konnten wir auf umfangreiche Daten vieler Teams zurückgreifen. Die Anleitung lässt Spielraum bei der Identifikation eines Fehlers, der Zuordnung der Fehlerart und der Fehlerschwere, schränkt aber den Freiraum bei der Datenerhebung deutlich ein.

Wir können nicht beurteilen, ob alle Teilnehmer die Fehler vollständig und korrekt dokumentiert haben. Es wurden nur wenige Spezifikations- und Entwurfsfehler spät entdeckt. Die Unterschiede im Korrekturaufwand der Fehler könnten auch durch die nicht kontrollierten Variablen verursacht werden. Die Ergebnisse bestätigen aber bisherige Beobachtungen.

Die externe Validität ist durch die geringe Projektgröße, das kleine und einfache Produkt, den vorgegebenen Prozess mit intensiven Reviews und die stabilen, hart vorgegebenen Anforderungen eingeschränkt. Die Teilnehmer sind Studenten und keine Praktiker und führen das Projekt betreut durch.

Die Ergebnisse entsprechen aber weitgehend den berichteten Praxiserfahrungen und bestätigen bekannte Zusammenhänge. Als typisch für die Praxis bewerten wir auch die kontinuierliche Integration in kleinen Projekten und die Schwierigkeiten mit der Fehlererhebung [Gra87].

6 Danksagung

Wir bedanken uns bei den Teilnehmern des Software-Praktikums für die Metrikerhebung und bei Sebastian Schumm für die Implementierung des Werkzeugs JDefectCollector und die Unterstützung bei der Datenerhebung und -auswertung.

Literatur

[Bas84] Basili, V. R.; Perricone, B. T.: *Software Errors and Complexity: An Empirical Investigation.* Com. of the ACM, 27(1) 1984.

[Boe76] Boehm, B. W.: *Software Engineering.* IEEE Transactions on Computers, C-25(12), 1976.

[Boe87] Boehm, B. W.: *Industrial Software Metrics Top 10 List.* IEEE Software 4(5), 1987.

[Frü06] Frühauf, K.; Ludewig, J.; Sandmayr, H.: *Software-Prüfung. Eine Anleitung zum Test und zur Inspektion.* 6. Aufl., Vdf, 2006.

[Gra87] Grady, R. B.; Caswell, D. L.: *Software Metrics: Establishing a Company-Wide Program.* Prentice-Hall, 1987.

[Ham07] Hampp, T.: *Klassifikationen für die Erhebung von Fehlerzahlen.* Internes Dokument, 2007.

[Hum95] Humphrey, W. S.: *A Discipline for Software Engineering.* Addison-Wesley, 1995.

[Jus08] http://www.iste.uni-stuttgart.de/se/research/projects/justus/index.html 4. Februar 2008.

[Kan03] Kan, S. H.: *Metrics and Models in Software Quality Engineering.* Addison-Wesley, 2nd Ed., 2003.

[Lud01] Ludewig, J. (Hrsg.): *Praktische Lehrveranstaltungen im Studiengang Softwaretechnik: Programmierkurs, Software-Praktikum, Studienprojekte, Fachstudie.* Mit Beiträgen von: S. Krauß, J. Ludewig, P. Mandl-Striegnitz, R. Melchisedech, R. Reißing. Bericht der Fakultät Informatik, Universität Stuttgart, 2. Aufl., 2001.

[Shu02] Shull, F.; Basili, V.; Boehm, B.; Brown, A. W.; Costa, P.; Lindvall, M.; Port, D.; Rus, I.; Tesoriero, R.; Zelkowitz, M.: *What We Have Learned about Fighting Defects.* Proc. of the 8th IEEE Symposium on Software Metrics (METRICS'02), 2002.