# Prüfen von C–Code und Statemate/Matlab–Spezifikationen: Ein Experiment

Frank Houdek[1], Dietmar Ernst[2], Thilo Schwinn[1]

[1]Daimler-Chrysler AG
Forschung und Technologie
Abt. Prozeßgestaltung
Postfach 23 60
89013 Ulm
{houdek,schwinn}@dbag.ulm.daimlerbenz.com

[2]Universität Ulm
Fakultät für Informatik
Abt. Programmiermethodik
und Compilerbau
89069 Ulm
dietmar@informatik.uni-ulm.de

**Zusammenfassung**

Die Softwareprüfung spielt bei der Softwareentwicklung eine wesentliche Rolle. Daher wurden hierzu in den letzten Jahren auch eine Vielzahl von Techniken entwickelt. Allerdings gibt es bisher nur wenig *empirisch* gesicherte Aussagen zu der Wirkungsweise der verschiedenen Techniken in verschiedenen Umfeldern.

In diesem Bericht beschreiben wir eine experimentelle Untersuchung, in der verschiedene, gängige Verfahren der Softwareprüfung auf zwei Arten von Artefakte — Programmcode und ausführbare Spezifikationen — im Bereich der eingebetteten Systeme angewandt wurden.

Bei der Untersuchung konnten wir ein signifikant besseres Abschneiden statischer Prüfverfahren (Inspektionen) feststellen und auch die bereits von Myers [Mye78] publizierte Beobachtung disjunkter Fehlerklassen bestätigen.

## 1 Einleitung

Die Prüfung von Software spielt in deren Lebenszyklus eine wesentliche Rolle. So werden bei größeren oder sicherheitskritischen Systemen oft mehr als 40 % des Projektbudgets für Softwareprüfaktivitäten aufgewendet [LRRA98]. Um diese Aufwände zu reduzieren, ist es notwendig, die zur Verfügung stehenden Techniken der Softwareprüfung gezielt einzusetzen oder so zu kombinieren, daß diese ihre bestmögliche Wirkung entfalten können. Dazu ist es jedoch unabdingbar, die Stärken und Schwächen der jeweiligen Techniken verstehen und auch quantifizieren zu können. Dies erfordert eine empirische Herangehensweise, bei der gezielt die Wirkungsweise einzelner Techniken beobachtet werden. In diesem Bericht präsentieren wir den Aufbau und die Ergebnisse einer solchen Untersuchung.

Da im täglichen Projektleben derartige Studien nur schwer durchführbar sind (ein paralleles Ausprobieren verschiedener Techniken verbietet sich zumeist schon aus Kostengründen, und der unmittelbare Einsatz neuer Techniken ist mit unkalkulierbaren Risiken verbunden), haben wir diese Untersuchung im Rahmen eines Praktikums an der Universität Ulm durchgeführt.

In dem Experiment haben wir versucht, statische und dynamische Verfahren der Softwareprüfung vergleichend zu betrachten. Dabei haben wir uns auf frühere Arbeiten, wie

beispielsweise [Mye91], [BS87], [KL95b], [PVB95] oder [EHSS97] gestützt. Diesen Untersuchungen ist gemeinsam, daß diese meist Programmcode in den Mittelpunkt der Betrachtung stellen, da sich dieser sowohl statisch (z. B. durch Inspektionen) als auch dynamisch (z. B. durch funktionalen Test) prüfen läßt.

Nun gewinnen, insbesondere in dem hier betrachteten Umfeld eingebetteter reaktiver Systeme, formale Spezifikationswerkzeuge zunehmend an Bedeutung. Exemplarisch seien hier *Statemate* und *Rhapsody* von *iLogix*, *Matlab* von *The MathWorks Company* oder *MatrixX* von *Integrated Systems Inc.* genannt.

Die mit diesen Werkzeugen erstellten Spezifikationsdokumente haben, ebenso wie Programmcode, die Eigenschaft ausführbar zu sein. (Viele der angesprochenen Werkzeuge bieten sogar die Möglichkeit der automatischen Code–Erzeugung an). Einem Einsatz dynamischer Techniken steht also prinzipiell nichts im Weg. Da die möglichst frühzeitige Entdeckung von Fehlern im Software–Entwicklungsprozeß unbestritten ist [Boe81], haben wir versucht, statische und dynamische Prüfverfahren auch für diese Art von Artefakten einzusetzen. Konkret haben wir dabei die Werkzeuge Statemate und Matlab eingesetzt.

Die Ergebnisse, die wir in Hinblick auf Programmcode gewonnen haben, decken sich mit den in der Literatur beschriebenen Beobachtungen. So konnten auch wir ein besseres Abschneiden statischer Prüfverfahren sowohl in Hinblick auf Effektivität (d. h. Anteil der gefundenen Mängel) als auch Effizienz (d. h. entdeckte Mängel pro Zeit) feststellen.

Bezogen auf die ausführbaren Spezifikationen ergab sich hier ein weniger scharfes Bild. Auffällig war jedoch, daß die Fehlerklassen, die mit den verschiedenen Techniken gefunden wurden, relativ wenig Überlappungen aufwiesen. Diese Beobachtung stützt diesbezügliche Untersuchungen bei Programmcode (siehe z. B. [Mye78]).

### 1.1 Aufbau des Berichts

Im nächsten Abschnitt beschreiben wir den Aufbau und die Durchführung des Experiments sowie die Hypothesen, die mit der Untersuchung verbunden waren. In Abschnitt 3 präsentieren wir die Ergebnisse aus dem Experimentteil, in dem wir C–Code betrachtet haben. Abschnitt 4 stellt die Ergebnisse, die bei der Betrachtung ausführbarer Spezifikationen gewonnen wurden, vor. In Abschnitt 5 wenden wir uns einigen speziellen Fragen in Hinblick auf Inspektionen, wie Nutzen der Inspektionssitzung oder Güte von Restfehlerschätzungen, zu. Eine kritische Betrachtung der Ergebnisse (Abschnitt 6) und eine kurze Zusammenfassung (Abschnitt 7) beenden diesen Bericht.

## 2 Aufbau und Durchführung des Experiments

Die experimentelle Untersuchung wurde im Rahmen eines Praktikums im Informatik–Hauptstudium an der Universität Ulm durchgeführt. Diese Vorgabe implizierte eine Reihe von Faktoren, die im experimentellen Design (siehe z. B. [Pfl95, JSK91, FP96]) berücksichtigt werden mußten. Im einzelnen waren dies:

- *Dauer:* Ein Praktikum sollte ein Semester dauern. Verglichen mit anderen Laborexperimenten (z. B. [KL95a]) stand uns damit eine relativ lange Dauer zur Verfügung, die sinnvoll genutzt werden sollte.
- *Aufwand:* Ein Praktikum ist als Veranstaltung mit vier Semesterwochenstunden angesetzt. Die wöchentliche Gesamtbelastung der Teilnehmer sollte daher acht Stunden nicht überschreiten.
- *Teilnehmerzahl:* Da das Angebot an Praktika in Ulm relativ groß ist, kann nur mit vier bis acht Teilnehmern gerechnet werden. (Am aktuellen Praktikum nahmen insgesamt sechs Studenten teil).

Um trotz der geringen Teilnehmerzahl zu vernünftige Aussagen zu gelangen, ist daher eine interne Replikation (d. h. mehrfaches Untersuchen derselben Fragestellung) angebracht. Um aber das Praktikum für die Teilnehmer interessant zu halten, ist aber auch hinreichend viel Abwechslung notwendig. Wir wählten daher den in Abbildung 2.1 dargestellten Ablauf.

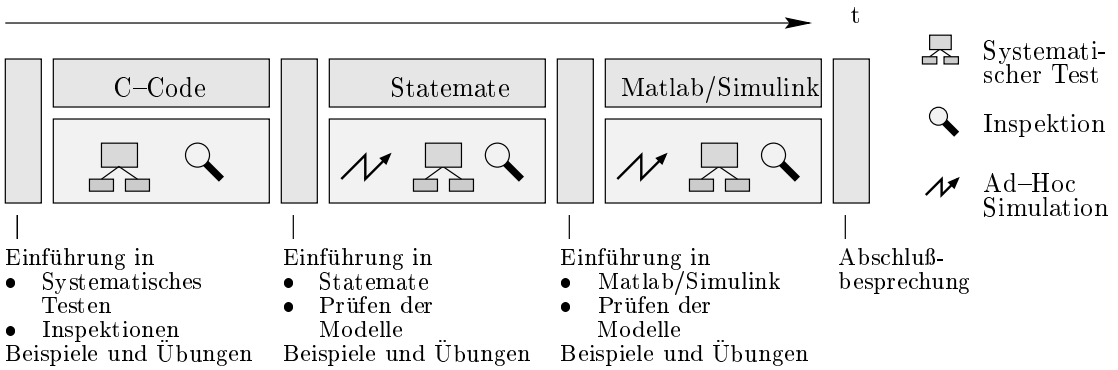



**Abb. 2.1:** Ablauf der experimentellen Untersuchung im Rahmen eines einsemestrigen Praktikums.

Zu Beginn der Veranstaltung fanden einführende Vorlesungen in die Softwareprüfung im allgemeinen und deren Umsetzung im Rahmen der Untersuchung im besonderen statt. Die eingesetzten Techniken beschreiben wir in Abschnitt 2.1 näher. Übungen an kleineren Beispielen beendeten diesen einführenden Teil.

Als erster großer Experimentblock wurden dann vorbereitete C–Programme (siehe Abschnitt 2.2) mittels Inspektion und systematischem Test näher betrachtet. Abbildung 2.2 beschreibt den detaillierten Ablauf dieses Blocks. Dieser Block ist die Replikation einer früheren Untersuchung, die wir im Wintersemester 1996/97 durchgeführt haben. In [EHSS97] sind der Ablauf und die Ergebnisse dieser Untersuchung dokumentiert.

Anschließend führten wir die teilnehmenden Studenten an das Werkzeug Statemate und die dahinterliegenden Konzepte — Statecharts [HLN+90, HP96] und Simulation — heran. Hier hatten die Studenten, mit Ausnahme von Grundkenntnissen aus der Automatentheorie, keinerlei Vorkenntnisse. Nach einführenden Übungen und der Vorstellung der

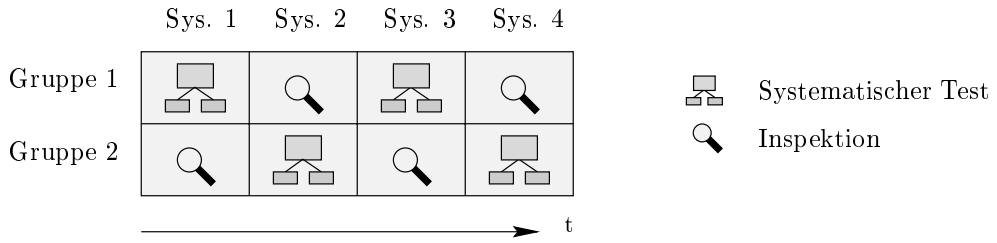



**Abb. 2.2:** Experimentelles Design zum Vergleich von Inspektion und Test bei C–Programmen. Die Studenten wurden in zwei Gruppen eingeteilt. Jede Gruppe prüfte in vier aufeinanderfolgenden Wochen je ein anderes System. Im Fall der statischen Prüfung bereiteten sich die 2-3 Teilnehmer individuell auf eine gemeinsame Sitzung vor. Bei der dynamischen Prüfung erstellte die Gruppe bis zum Ende der Woche ein gemeinsames Testprotokoll.

adaptierten Prüfverfahren folgte der zweite große Experimentblock, in dem Statemate–Analysemodelle geprüft wurden. In Abbildung 2.3 ist der Detailablauf graphisch aufbereitet.

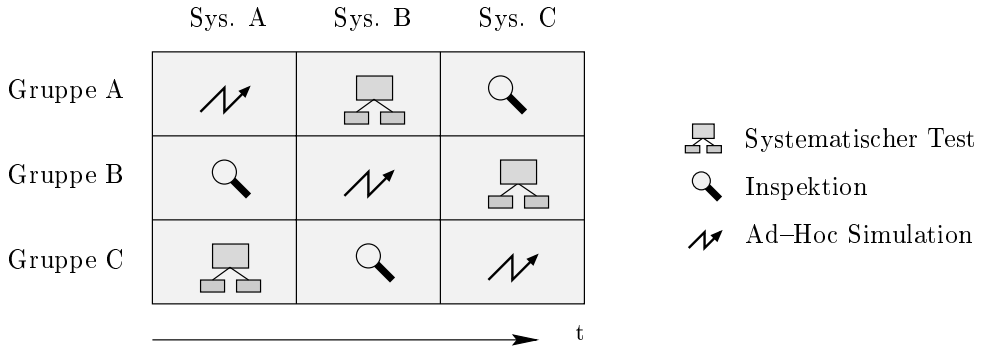



**Abb. 2.3:** Experimentelles Design zum Vergleich von Inspektion, Ad–Hoc Simulation und Test für Statemate (und auch Matlab/Simulink) Dokumente. Die Studenten wurden in drei Gruppen eingeteilt. In drei aufeinanderfolgenden Wochen wurde je ein System mit je einer Technik geprüft. Im Falle der Inspektion bereiteten sich die zwei Gruppenmitglieder individuell auf eine gemeinsame Sitzung vor. Bei Ad–Hoc Simulation und Test arbeiteten die beiden Gruppenmitglieder für sich und gaben dann auch je zwei Prüf–Protokolle ab.

Im letzten großen Block betrachteten wir Matlab/Simulink. Strukturell war der Ablauf in diesem Block identisch mit dem vorangegangenen Statemate–Block (vgl. Abbildung 2.3). Auch hier hatten die Studenten (fast) keinerlei Vorkenntnisse.

Am Ende des Praktikums fand eine Abschlußbesprechung statt, in der die Erfahrungen der Teilnehmer zusammengetragen wurden. In der Zusammenfassung finden sich einige der dort geäußerten Eindrücke.

## 2.1 Eingesetzte Prüftechniken

Im Rahmen der Untersuchung betrachteten wir Inspektionen und Test (bei C–Code) bzw. Inspektionen, Test und Ad–Hoc Simulation (bei ausführbaren Spezifikationen). In den folgenden Abschnitten wollen wir kurz diese Techniken beschreiben und insbesondere auf deren Implementierung im Rahmen des Praktikums eingehen.

### 2.1.1 Inspektionen

Inspektionen gehören zur Klasse der statischen Prüftechniken. Diesen Techniken liegt die Idee zugrunde, daß vier Augen mehr sehen als zwei. Im Rahmen einer Inspektionssitzung wird der Prüfling von mehreren Personen begutachtet und bewertet. Da der Prüfling bei der Prüfung nicht „ausgeführt“wird, lassen sich diese Techniken auf alle Artefakte anwenden, die sinnvoll gelesen werden können (z. B. Verträge, Analysedokumente, Entwürfe, Programmcode oder Testpläne, nicht aber Hex–Listings o. ä.). Die verschiedenen statischen Prüftechniken unterscheiden sich in der Zielsetzung der Prüfung (z. B. Aufdecken von Mängeln oder Überprüfen der Übereinstimmung mit der Planung), den beteiligten Personen (z. B. mit Moderator und Protokollant) und dem zugrundeliegenden Prozeß (z. B. kurzfristig angesetzte Inspektionssitzung ohne Vorbereitung oder formaler Prozeß).

Nach IEEE 610.12-1990 ist eine Inspektion eine *„statische Analysetechnik, die auf visueller Betrachtung entwickelter Produkte beruht, um (in diesen) Fehler, Verletzungen von Entwicklungsstandards sowie weitere Probleme zu finden.“* [1] (vgl. auch IEEE 1028). Die Zielsetzung einer Inspektion ist also die Identifikation von Mängeln. In Tabelle 2.1 sind die einzelnen Schritte des Inspektionsprozesses beschrieben.

Die Rollenverteilung sah in unserer Untersuchung wie folgt aus: Die Studenten übernahmen den Part der Inspektoren. Die Moderation und Protokollierung sowie die Autorenschaft wurde von uns als Praktikumsbetreuern wahrgenommen.

Die Überblickssitzung gestaltete sich in der Regel recht kurz. An Hand einer Übersichtsskizze wurde das System kurz vorgestellt, und anschließend wurden noch Besonderheiten in der jeweiligen Modellierung angesprochen.

Zur Vorbereitung hatten die Studenten eine Woche Zeit. Die Mängel, die sie dabei fanden, wurden auf einem Formblatt zusammen mit der Vorbereitungszeit notiert. Allgemeine Prüffragen in Form einer Checkliste unterstützten sie dabei. Die individuelle Vorbereitungszeit lag typischerweise zwischen 60 und 180 Minuten.

In der Inspektionssitzung wurden die individuellen Mängel zusammengetragen. Da die Zahl der Mängel, die die einzelnen Inspektoren gefunden hatten, nie sehr groß war (bedingt auch durch die Größe der Dokumente), bestand keine Notwendigkeit, die Inspektionssitzung besonders rigide zu moderieren. Lediglich die Versuche einer Lösungsfindung wurden vom Moderator unterbunden. Im Mittel dauerten diese Sitzungen 30 Minuten.

Diese Vorgehensweise wurde für alle drei Arten von Dokumenten so beibehalten.

[1] Originaldefinition: *A static analysis technique that relies on visual examination of development products to detect errors, violations of development standards, and other problems.*

| *Schritt* | *Aktivitäten* | *Wer? (Rolle)* |
|---|---|---|
| Planung | Startpunkt für eine Inspektion ist ein zu überprüfendes Artefakt. Für dieses wird festgelegt, wann und wer dieses zu inspizieren hat. Die betroffenen Personen werden informiert. | Moderator |
| Einführungssitzung | Der Autor oder Moderator stellt den Prüfling den Inspektoren vor, teilen die benötigten Dokumente (z. B. Formblätter oder Referenzunterlagen) aus und erläutern die genaue Zielsetzung der Inspektion. | Moderator, Autor |
| Vorbereitung | Die Inspektoren bereiten sich *individuell* auf die Inspektionssitzung vor und versuchen dabei, Mängel und Unklarheiten im Prüfling zu entdecken. Die Vorbereitung kann strukturiert oder unstrukturiert erfolgen. Im ersten Fall kann durch Checklisten, Szenarien oder Perspektivenbeschreibungen eine fokusiertere „Lesetechnik“ vorgegeben werden. | Inspektor |
| Inspektionssitzung | In der Inspektionssitzung führt der Moderator durch den Prüfling. Dabei merken die Inspektoren ihre Mängel an und stellen Verständnisfragen. Der Autor darf sich in dieser Sitzung nicht verteidigen. Seine Anwesenheit dient (1) der Klärung von Fragen, (2) zum Ausräumen von Mißverständnissen und (3) zur besseren Information des Autors. Lösungsansätze sollten im Rahmen einer Inspektionssitzung nicht diskutiert werden. | Inspektor, Moderator, Autor, Protokollant |
| Korrektur* | Anhand der protokollierten Mängel versucht der Autor, den Prüfling zu verbessern. | Autor |
| Überprüfung* | Der Moderator überprüft die Ergebnisse der Nacharbeit und entscheidet (ggf. auch nach Beratung mit den Inspektoren), ob eine Re–Inspektion notwendig ist oder nicht. | Moderator |

**Tab. 2.1:** Inspektionsprozeß. Die mit * markierten Schritte wurden im Rahmen dieses Praktikums nicht durchgeführt.

### 2.1.2 Systematischer Test

Während bei Inspektionen bewußt auf die Ausführung des Prüflings verzichtet wird, so ist dies das zentrale Element aller Testaktivitäten. Die wesentliche Idee hinter dem Testen ist, die Korrektheit eines Systems dadurch zu belegen, indem man zeigt, daß es für alle

möglichen Eingaben die korrekten Antworten liefert. Bei zeitkritischen Systemen impliziert die korrekte Antwort nicht nur das (zahlenmäßig) richtige Ergebnis, sondern auch den erwarteten Zeitpunkt, zu dem das Ergebnis zur Verfügung steht. In der Regel ist aber eine vollständige Überprüfung aller Eingaben (ggf. zusammen mit ihren Zeitabhängigkeiten) nicht möglich. Man kann nur eine Repräsentantenmenge aller Eingabedaten einsetzen. Von der Güte dieser Repräsentantenmenge hängt es ab, ob ggf. noch im System vorhandene Mängel entdeckt werden können oder nicht. Durch die Beschränkung auf eine Repräsentantenmenge kann der Test aber nicht mehr verwendet werden, um die Korrektheit eines Systems zu zeigen, sondern nur, um Mängel aufzudecken. Bevor das System mit den Elementen der Repräsentantenmenge (den sogenannten Testfällen) ausgeführt wird, muß noch festgelegt werden, welche Ausgabe für diese Eingaben erwartet wird. Nur so ist eine unbeeinflußte Bewertung der Ausführungsergebnisse gewährleistet.

Die verschiedenen Testtechniken unterscheiden sich nun darin, wie die Repräsentantenmenge gebildet wird. Beim *funktionalen Test* wird diese Menge ausgehend von der Systemspezifikation gebildet. Es wird zuerst geprüft, welche Funktionalitäten das System zur Verfügung stellen soll. Dann werden Eingabedaten konstruiert, die diese Funktionalitäten nutzen. Ein Hilfsmittel zur Konstruktion dieser Eingabedaten ist die Klassifikationsbaummethode [GW95, Mye91].

Beim *strukturellen Test* wird die Repräsentantenmenge auf Basis der Systemstruktur gebildet. Durch eine Analyse der Programmstrukturen wird festgestellt, welche Eingabedaten nötig sind, damit jeder Ausführungspfad mindestens einmal durchlaufen wird. Da auch dies in der Regel nicht möglich ist (z. B. können Schleifen beliebig oft durchlaufen werden), wird dieses Kriterium weiter abgeschwächt. In unserer Untersuchung lag eine „optimale Überdeckung“ (von Programmcode) dann vor, wenn

- jede Anweisung mindestens einmal ausgeführt wurde,
- in jeder Fallunterscheidung der `then` und `else` Zweig benutzt wurden,
- jede Schleife null mal, einmal, und mindestens zwei mal durchlaufen wurde sowie
- in jeder logischen Verknüpfung jeder Teilterm einmal wahr und falsch war.

Solche Überprüfungen sind manuell sehr aufwendig. Aus diesem Grund haben wir den Instrumentierer GCT (Generic Coverage Tool, [Mar91, Mar92]) eingesetzt. Dieses Werkzeug modifiziert den Prüfling in der Weise, daß an allen relevanten Stellen Zähler eingebaut werden. Eine summarische Auswertung am Ende eines Testlaufs weist dann auf eine mangelnde Überdeckung hin.

Im Rahmen unserer Untersuchung haben wir eine Kombination von funktionalem und strukturellem Test eingesetzt. Zuerst erstellten die Studenten einen Klassifikationsbaum und leiteten aus diesem die Testfälle für den funktionalen Test ab. Nach dem Ausführen der Testfälle und der Analyse der Ergebnisse wurde mit Hilfe des GCT (nur bei C–Code) überprüft, ob die Überdeckung durch diese Testfälle bereits optimal war. Wenn nicht, wurde versucht, weitere Testfälle zu finden bzw. zu begründen, warum eine vollständige Überdeckung nicht möglich ist.

Während der funktionale Test sowohl bei C-Code wie auch bei den ausführbaren Spezifi-

kationsdokumenten eingesetzt werden kann, ergaben sich beim strukturellen Test Unterschiede in Abhängigkeit der betrachteten Dokumente.

Im Falle der Statemate–Modelle forderten wir, daß eine optimale Überdeckung dann vorliegt, wenn jeder Zustand und jeder Zustandsübergang mindestens einmal benutzt wurde. Das Werkzeug Statemate stellt hierzu ein Protokoll zur Verfügung, welches beschreibt, welcher Zustand und welcher Übergang wann benutzt worden sind. Dieses Protokoll ist allerdings sehr schwer lesbar (siehe auch Abschnitt 6.2.1).

Bei Matlab/Simulink–Modellen sahen die Überdeckungskriterien vor, daß jedes Schalt–Element (Schalter, Multischalter) einmal in jeder Stellung arbeitete und jede logische Verknüpfung einmal wahr und falsch liefert. Das Werkzeug selbst stellt hierzu keinerlei Unterstützung zur Verfügung.

Das Ergebnis des Testprozesses war in jedem Fall (C–Code, Statemate, Matlab/Simulink) ein Testprotokoll, das folgende Bestandteile hatte.

- Klassifikationsbaum
- Die daraus abgeleiteten Testfälle
- Die erwarteten Ergebnisse
- Die bei der Ausführung tatsächlich beobachteten Ergebnisse
- Eine Liste der beim funktionalen Test entdeckten Fehlverhalten
- Weitere Testfälle zur Optimierung der Überdeckung
- Begründung, falls eine optimale Überdeckung nicht erreicht werden kann
- Erwartete und tatsächliche Ergebnisse der weiteren Testfälle
- Mängelliste aufgrund der weiteren Testfälle
- Liste der Ursachen für die entdeckten Mängel

### 2.1.3 Ad–Hoc Simulation

Die Ad–Hoc Simulation ist eine abgespeckte Variante des systematischen Tests. Gemeinsam ist die Idee, Mängel dadurch zu entdecken, indem das System mit typischen Eingabedaten ausgeführt wird. Diese Eingaben resultieren aber nicht aus einer systematischen Vorgehensweise, sondern basieren auf der Intuition und Erfahrung des jeweiligen Prüfers.

Als Ergebnis einer Ad–Hoc Simulation forderten wir von den Studenten einen Bericht, in dem alle Mängel, die ihnen bei der Prüfung aufgefallen sind, dokumentiert sind. Zur Ausführung der Systeme wurden die Simulationsmöglichkeiten der betroffenen Werkzeuge herangezogen. Insbesondere Statemate bietet durch die sogenannte Panels eine elegante Möglichkeit, die Schnittstelle der Systeme nach „außen“ zu modellieren und interaktiv erlebbar zu machen.

## 2.2 Verwendete Systeme und Materialien

Zu jedem der eingesetzten Systeme, die als C–Code, Statemate–Modell oder Matlab/Simulink–Modell vorlagen (s. u.), lag eine *Systemspezifikation* vor. Dieses Dokument diente als Referenzdokument für die durchgeführten Prüfungen.

In der Systemspezifikation werden umgangssprachlich die Anforderungen an das jeweils betrachtete System beschrieben. Wo sinnvoll, findet sich auch eine Übersichtsskizze oder erläuterndes Diagramm. Eine Beschreibung der Signale, mit denen das jeweilige System mit der Außenwelt interagieren kann, ist ebenfalls Bestandteil der Systemspezifikation. Die Dokumentgröße betrug zwischen zwei und sechs Seiten.

### 2.2.1 C–Systeme

In Tabelle 2.2 haben wir die vier eingesetzten C–Systeme kurz charakterisiert. Die Größe (gemessen in Lines of Code, inkl. Kommentare) bezieht sich jeweils auf den Teil, der im Rahmen der Prüfaktivitäten untersucht wurde. Darüberhinausgehende Teile (wie z. B. die Hardwaresimulation, s. u.) wurden nicht mitgezählt.

Die Angabe $x + y$ an enthaltenen Mängeln bedeutet, daß $x$ Mängel sowohl durch Testaktivitäten als auch durch Inspektionen entdeckt werden können, während $y$ Mängel nur mit Hilfe von Inspektionen entdeckt werden können. In diese Gruppe fallen Mängel, die vor allem struktureller Natur sind, wie schlechte Modularisierung oder Fehler in Schnittstellen, soweit sie sich nicht auf die Funktionalität auswirken. Marginale Mängel, wie Schreibfehler oder die Verwendung von konstanten Zahlen (*Magic Number*), fallen nicht unter die Mängelzählung. Da wir annahmen, daß jeder Mangel im Rahmen einer Inspektion gefunden werden kann, hatten wir nur–testbare Mängel nicht separat erfaßt (vgl. Abschnitt 2.3).

Da uns im Rahmen des Praktikums die Hardwareseite der betrachteten Systeme nicht zur Verfügung stand, mußten wir diese durch Software emulieren. In Abbildung 2.4 ist der prinzipielle Aufbau der betrachteten Systeme dargestellt. Die Hardwaresimulation bildet dabei das Verhalten der Sensoren und Aktuatoren nach. Außerdem bietet die Hardwaresimulation die Möglichkeit, Skripte abzuarbeiten, um so zu reproduzierbaren Testfällen zu gelangen. Diese Skripte sind in der Regel Zeit–Ereignis Protokolle, d. h. eine Folge von Zeitpunkten mit den Ereignissen (z. B. Tastendruck, Pfeifton einer bestimmten Frequenz), die zu diesem Zeitpunkt aktiv sind. Die Ausgabe eines Programmlaufs besteht sowohl aus graphischen Dokumenten (Zeit–Signal Diagramme) wie auch textuellen Protokollen.

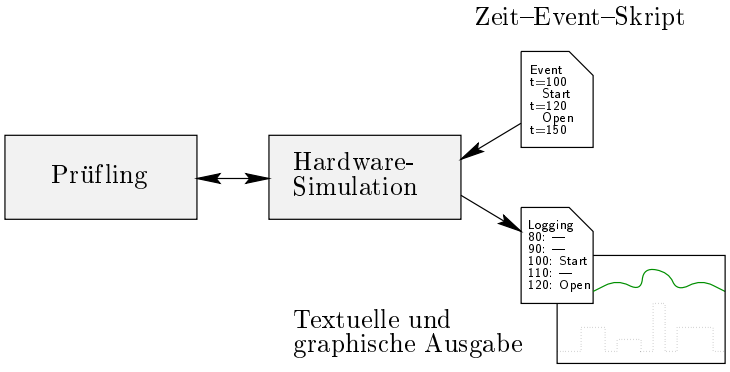



**Abb. 2.4:** Aufbau der eingesetzten Systeme.

| |
|---|
| **Mikrowelle**$^{+}$<br>*Größe:* 308 Zeilen<br>*Enthaltene Mängel:* 10+3<br>*Charakteristika:* Ein Prozeß, Zustandsgesteuert, harte Zeitbedingungen<br>*Kurzbeschreibung:* Steuerung einer Mikrowelle. Die Bedienung hat in fest vorgegebenen Schritten (Leistungsstufe, Laufzeit) zu erfolgen. Während der Heizphase muß überwacht werden, ob die Tür geöffnet wird, um daraufhin sofort die Mikrowellenspule abzuschalten. |
| **Roboter**$^{+}$<br>*Größe:* 448 Zeilen<br>*Enthaltene Mängel:* 11+6<br>*Charakteristika:* Sequentielles Mehrprozeßsystem<br>*Kurzbeschreibung:* Steuerung eines mobilen, autonomen Roboters. Der Roboter fährt normalerweise geradeaus. Bei einer Kollision versucht er, dem Hindernis auszuweichen. Erkennt der Roboter Pfeiftöne einer bestimmten Frequenz, so fährt er eine Kurve. |
| **Motorsteuerung**$^{+}$<br>*Größe:* 485 Zeilen<br>*Enthaltene Mängel:* 14+5<br>*Charakteristika:* Zwei Prozesse, einfacher Regler<br>*Kurzbeschreibung:* Motorsteuerung für eine schwere landwirtschaftliche Maschine. Die verteilte Steuerung regelt die Start- und Betriebsphase eines Verbrennungsmotors und zeigt die aktuellen Betriebsdaten dem Fahrer an. |
| **Kopierer**$^{*}$<br>*Größe:* 539 Zeilen<br>*Enthaltene Mängel:* 13+0<br>*Charakteristika:* Fünf Prozesse<br>*Kurzbeschreibung:* Steuerung eines einfachen Kopierers. Durch den Benutzer können die Anzahl der Kopien und die Vergrößerung eingestellt werden. Die Steuerung überwacht den Durchlauf eines Blattes durch den Kopierer und aktiviert jeweils die entsprechenden Hardwarekomponenten. Außerdem wird ein automatischer Vorlageneinzug unterstützt. |

**Tab. 2.2:** Eingesetzte C–Systeme. Die mit $^{+}$ markierten Systeme wurden bereits in einem früheren Praktikum (WS 1996/97, [EHSS97]) eingesetzt. Das mit $^{*}$ markierte System wurde von Studenten im Rahmen eines weiteren Praktikums [EHSa, EHSb] entwickelt. In dieses System wurden keine weiteren Mängel eingebaut; alle Mängel stammen ausschließlich aus dessen Entwicklung.

Als Ausführungsumgebung setzten wir UNIX–Workstations ein, da (1) diese für die universitäre Ausbildung zu Verfügung stehen, (2) die Beteiligten mit diesen Systemen vertraut sind, und (3) diese Systeme parallele Prozesse unterstützen. Nachteilig an dieser Wahl ist allerdings, daß das Laufzeitverhalten auf diesen Systemen nicht völlig stabil ist. So beein-

flußt die Auslastung eines Rechners auch das Verhalten der Prüflinge zur Ausführungszeit. Da die eingesetzten Rechner aber hinreichend schnell waren, traten in diesem Bereich nur selten Probleme auf[2].

Die Materialien, die an die Studenten ausgeteilt wurden, umfaßten neben einer Systemspezifikation die Listings der eigentlichen Systeme und die Schnittstellenbeschreibung zur Hardware(-simulation). Den Testern stand zudem das ausführbare und instrumentierte Programm (Systemsteuerung zusammen mit der Hardwaresimulation) zur Verfügung.

### 2.2.2 Statemate–Systeme

In Tabelle 2.3 findet sich eine Kurzbeschreibung der eingesetzten Statemate–Modelle. Die Fehlerzahlen sind ebenso wie bei den C–Code–Systemen zu verstehen (Abschnitt 2.2.1, Seite 9). Zur Charakterisierung der Modellgröße haben wir die Anzahl der Zustände (elementare Zustände wie gruppierende Zustände) verwendet.

| |
|---|
| **Anrufbeantworter**<br>*Größe:* 45 Zustände<br>*Enthaltene Mängel:* 15+1<br>*Kurzbeschreibung:* Steuerung eines Anrufbeantworters. Das System ermöglicht das Wiedergeben eines zuvor gespeicherten Textes im Falle eines Anrufs und das anschließende Aufzeichnen der Mitteilung des Gesprächspartners. Während des Abhörens der aufgezeichneten Nachrichten kann das Band beliebig vor- und zurückgespult werden. |
| **Innenlicht eines PKW**<br>*Größe:* 38 Zustände<br>*Enthaltene Mängel:* 8+0<br>*Kurzbeschreibung:* Steuerung des Innenraumlichts in einem PKW (tatsächlich wurde die Systemspezifikation der Beschreibung des Innenlichts eines BMW entnommen). Das An- und Ausschalten des Lichts hängt sowohl von der Stellung der Fahrzeugtüren wie auch der Zündung, dem Unfallsensor und der Fahrzeugbeleuchtung ab. |
| **Liftsteuerung**<br>*Größe:* 43 Zustände<br>*Enthaltene Mängel:* 6+5<br>*Kurzbeschreibung:* Steuerung eines Lifts in einem 10-stöckigen Gebäude. In jedem Stockwerk befinden sich Taster, um den Lift für eine Auf- oder Abwärtsfahrt anfordern zu können. Diese Anfragen sowie die Fahrtwünsche, die im Lift angegeben werden, sollen nach einer bestimmten Reihenfolge abgearbeitet werden. |

**Tab. 2.3:** Eingesetzte Statemate–Modelle.

Bei den Modellen selbst haben wir uns im wesentlichen auf Statecharts konzentriert. Zur Strukturierung haben wir außerdem ein Activitychart verwendet. Auf den Einsatz darüber

[2]In dem bereits angesprochenen vorangegangenen Praktikum (WS 1996/97, [EHSS97]) trat dieses Phänomen deutlich störender in Erscheinung.

hinausgehender Konstrukte, wie z. B. Entscheidungstabellen oder Einbindung externen Codes (vgl. [Sta97]), haben wir bewußt verzichtet.

Die in diesem Praktikumsteil ausgeteilten Materialien umfassten neben der Systemspezifikation das graphische Statemate–Modell sowie die Ausgaben des Data–Dictionary. Den Testern und Ad–Hoc Prüfern standen zudem ein Skript–Interface bzw. eine sogenanntes Panel zur Verfügung, mit dem interaktiv das Modell simuliert werden kann.

### 2.2.3 Matlab/Simulink–Systeme

Mit Matlab/Simulink können komplexe, nichtlineare Systeme in einer ansprechenden und übersichtlichen Art und Weise graphisch beschrieben werden [Hof98]. Durch Simulation kann die Lösung dieser Systeme numerisch bestimmt werden. Zur Illustration dieser Arbeitsweise findet sich nachfolgend ein einfaches Beispiel.

**Beispiel** (Bewegung eines elastischen Balls)

In diesem Beispiel wollen wir die Bewegung eines elastischen Balls beschreiben, der von einer Anhöhe gestoßen wird (siehe Abbildung 2.5). Das zugehörige Matlab/Simulink–

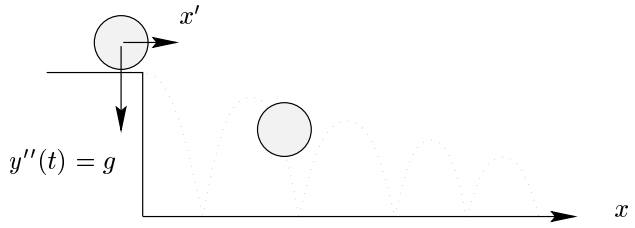
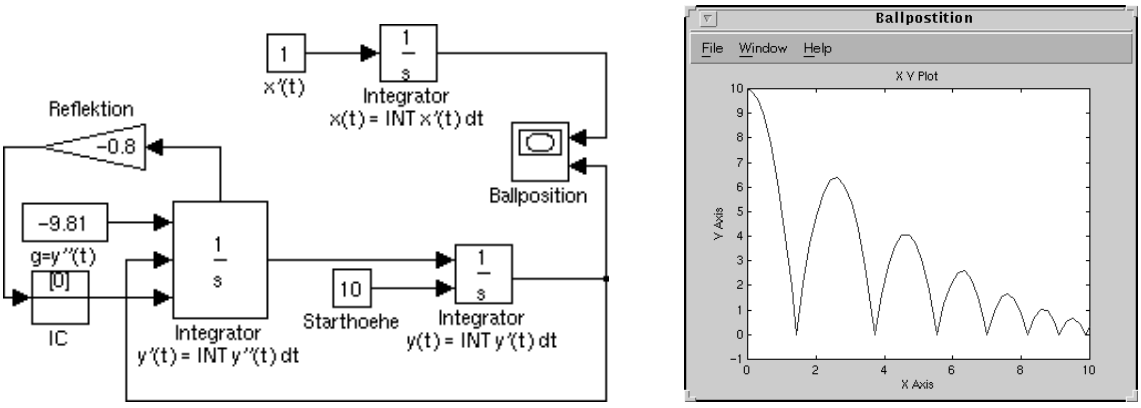



**Abb. 2.5:** Beispiel: Bewegung eines elastischen Balls.

Modell ist in Abbildung 2.6[a] wiedergegeben. Das Modell kann dabei wie folgt gelesen werden: Die Erdanziehung $g = -9,81\,\mathrm{m/s^2}$ wird über die Zeit integriert und ergibt so die




[a] [b]

**Abb. 2.6:** Beispiel: Matlab/Simulink–Modell der Ballbewegung [a] und Ausgabe [b].

Geschwindigkeit $y'(t)$ des Balls in $y$–Richtung. Diese wird erneut über die Zeit integriert und beschreibt somit die $y$–Position $y(t)$ des Balls. Bei dieser Berechnung wird mit einer initialen Höhe von 10 m gestartet. Ist $y(t) = 0$, so wird der linke Integrator mit $-0,8 \cdot y'(t)$ wieder gestartet, d. h. die Energie des Balls wird zu 80 % reflektiert. Das *Initial Condition (IC)* Element sorgt dabei dafür, daß beim ersten Starten des linken Integrators dieser mit 0 initialisiert wird, d. h. direkt nach dem Abstoßen hat der Ball keine Geschwindigkeit in $y$–Richtung. In $x$–Richtung wird nur die (konstante) Geschwindigkeit des Balls von 1 m/s über die Zeit integriert. Beide Positionen werden zusammen auf dem Ortsdiagramm *Ballposition* angezeigt (siehe Abbildung 2.6[b]). □

Die wesentlichen Charakteristiken der eingesetzten Matlab/Simulink–Modelle haben wir in Tabelle 2.4 zusammengefaßt. Da die beteiligten Studenten mit der Regelungstechnik relativ wenig vertraut waren, konnten wir bei den Modellen nur auf die einfacheren Modellierungs–Elemente, wie Integrierer, Addierer, Schalter oder Differenzierer zurückgreifen. Aus unserer Erfahrung heraus ist diese Einschränkung in bestimmten Entwicklungsfeldern (z. B. bei der Entwicklung von Steuergeräten) nicht unüblich. Der volle „Sprachumfang“ kommt eher bei umfangreichen Simulationen zum Einsatz.

| |
|---|
| **Tempomat**<br>*Größe:* 70 Elemente<br>*Enthaltene Mängel:* 7+1<br>*Kurzbeschreibung:* Automatische Regelung der Geschwindigkeit eines Fahrzeugs. Diese Regelung wird aktiv, sobald der Benutzer einen Hebel betätigt und wird wieder deaktiviert, sobald das Gaspedal oder die Bremse benutzt wird. |
| **Fenstersteuerung**<br>*Größe:* 70 Elemente<br>*Enthaltene Mängel:* 7+0<br>*Kurzbeschreibung:* Steuerung der Fenster eines Gebäudes mit Glasfront. Durch Öffnen und Schließen der Fenster soll die Temperatur im Innenraum geregelt werden. Bei Regen sind die Fenster zu schließen. |
| **Dampfkessel**<br>*Größe:* 48 Elemente<br>*Enthaltene Mängel:* 5+0<br>*Kurzbeschreibung:* Fehlertolerante Steuerung eines Heizkessels. Durch Steuerung der Wasserzufuhr eines Kessels soll in diesem die Wassermenge in einem bestimmten Intervall gehalten werden. Der Ausfall eines von mehreren Sensoren soll dabei das System nicht sofort zum Abschalten zwingen. |

**Tab. 2.4:** Eingesetzte Matlab/Simulink–Modelle.

Verglichen mit den C–Systemen oder auch den Statemate–Modellen waren die eingesetzten Matlab/Simulink–Modelle merklich einfacher. Diese Einschätzung wurde sowohl von den beteiligten Studenten bestätigt als auch durch eine Bewertung, die sich an einer Function Point Zählung [Alb86, AMDS97, EH98] orientiert. In Tabelle 2.5 finden sich sowohl die

subjektive Einschätzung der Teilnehmer als auch die modifizierte Function Point Bewertung.

| | System | Stud. Einschätzung | Bewertung eFP |
|---|---|---|---|
| C–Code | Mikrowelle | 5 | 46,6 |
| | Roboter | 8 | 55,4 |
| | Motorsteuerung | 8 | 59,0 |
| | Kopierer | 10 | 72,7 |
| Statemate | Anrufbeantworter | 6 | 51,8 |
| | Innenlicht | 5 | 27,0 |
| | Liftsteuerung | 9 | 44,3 |
| Matlab/Simulink | Tempomat | 5 | 29,6 |
| | Fenstersteuerung | 6 | 30,0 |
| | Dampfkessel | 4 | 19,9 |

**Tab. 2.5:** Komplexitätsbewertung der Systeme. Wir baten die Studenten, die betrachteten Systeme auf einer Skala von 1 bis 10 relativ zueinander einzuordnen. Die erweitere Function Point Bewertung (kurz: eFP) betrachtet Eingaben, Ausgaben, Kontrollstrukturen sowie Kritikalität der Zeitbedingungen und faßt diese zu einem einzelnen Wert zusammen. Die Absolutwerte haben in diesem Zusammenhang wenig Bedeutung, wichtig ist das Verhältnis untereinander.

## 2.3 Hypothesen der Untersuchung und Planung der Analyse

In diesem Abschnitt wollen wir kurz auf die Hypothesen[3] eingehen, die der hier vorgestellten Untersuchung zugrundeliegen, und auf die wir im Zuge der Analyse der Daten immer wieder zurückkommen werden. Die folgenden Hypothesen basieren im wesentlichen — vor allem in Bezug auf die Prüfung von C–Code — auf früheren eigenen Untersuchungen (vgl. [EHSS97]) sowie weiteren Literaturdaten (z. B. [KL95a, GG93, Mye78, BS87]). In Bezug auf die Prüfung ausführbarer Spezifikationsdokumente hatten auch eigene Beobachtungen in industriellen Entwicklungsprojekten Einfluß auf unsere Vermutungen.

- *C–Code, Aufwände:* Die Aufwände (bei einmaliger Prüfung) sind bei Inspektionen geringer als beim Testen.
- *Ausf. Spez., Aufwände:* Die Ad–Hoc Simulation ist weniger aufwendig als Inspektion und Test (da keine umfangreiche Vorbereitung stattfindet).
- *C–Code, Effektivität:* Der Anteil der Mängel, die mit Inspektionen gefunden werden, ist höher als bei Einsatz eines systematischen Testprozesses.
- *Ausf. Spez., Effektivität:* Inspektionen schneiden besser ab als Test. Ad–Hoc Simulationen haben eine deutlich geringere Effektivität.

[3] Die Hypothesen sind nicht als Hypothesen im Sinne eines statistischen Hypothesentests zu verstehen. Die hierzu nötigen Null- und Alternativhypothesen lassen sich aber teilweise leicht ableiten. Soweit sinnvoll, finden sich diese Hypothesen auch bei den jeweiligen Analysen in den Abschnitten 3 und 4.

- *C–Code, Fehlerklassen:* Theoretisch läßt sich nicht jeder Mangel mit jeder Technik gleich gut finden. Die mit Inspektionen gefundenen Mängel sind eine Obermenge der mit Test gefundenen Mängel.
- *Ausf. Spez., Fehlerklassen:* Die durch Inspektionen identifizierten Mängel sind eine Obermenge der mit den anderen Techniken gefundenen Mängel.

Neben einer Untersuchung dieser Hypothesen waren wir daran interessiert, spezielle Fragen zur statischen Softwareprüfung näher zu untersuchen. Im einzelnen waren dies:

- Was macht den Nutzen einer Inspektionssitzung aus? Dabei wollen wir insbesondere den Aspekt der Synergieeffekte (*review gains*) untersuchen, d. h. der Anteil der Meldungen, die erst im Rahmen der Sitzung entstehen und damit kein Inspektor auf seinem Vorbereitungsformular hat. Bezüglich dieser Effekte interessiert uns vor allem deren Anteil sowie die Art der Fehler, die auf diese Weise entdeckt werden.
- Wie gut ist das *Capture–Recapture Modell* [BEFL97] zur Restfehlerschätzung geeignet? Mit diesem Modell kann geschätzt werden, wie viele Fehler nach einer Inspektion noch übrig sind.
- Die Meldungen, die im Sitzungsprotokoll auftauchen, sind nur *potentielle* Fehler. Wie viele Falschmeldungen sind hier mit dabei?
- In der Inspektionssitzung werden auch Meldungen „wegdiskutiert“, teils, weil ein Inspektor etwas falsch verstanden hatte, teils zu unrecht. Wie hoch ist der Anteil der zu unrecht entfernten Meldungen (*review losses*)?

In den folgenden Abschnitten werden wir nun die beobachteten Daten vorstellen, diese analysieren und dabei auch auf die o. a. Hypothesen und Fragen eingehen. Liegen den Aussagen statistische Tests zugrunde, so wurden diese zum Signifikanzniveau $\alpha = 10\,\%$ durchgeführt.

# 3 Ergebnisse *Prüfen von C–Code*

## 3.1 Aufwand, Effektivität, Effizienz

Die Rohdaten, die den folgenden Analysen zugrunde liegen, finden sich im Anhang in Tabelle A.1. In Abbildung 3.1 haben wir den Aufwand, der für die durchgeführten Prüfungen aufgewendet wurde, aufgetragen[4]. Die Verteilung der Aufwände auf die einzelnen Teilaufgaben sind Gegenstand von Abbildung 3.2[a] und 3.2[b].

Auffällig bei den Aufwänden ist, daß in zwei Durchläufen (Roboter, Kopierer) die Testaufwände deutlich über denen einer Inspektion liegen, in den beiden anderen Fällen jedoch

[4]Die Aufwände sind dabei immer auf drei Teilnehmer je Gruppe normiert.

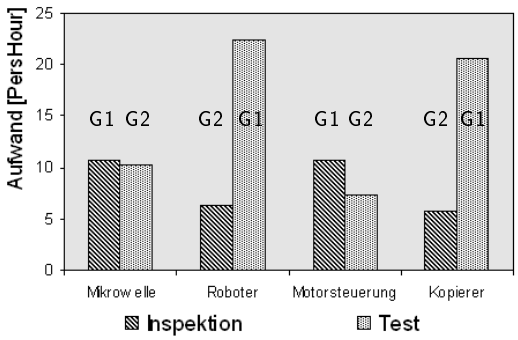
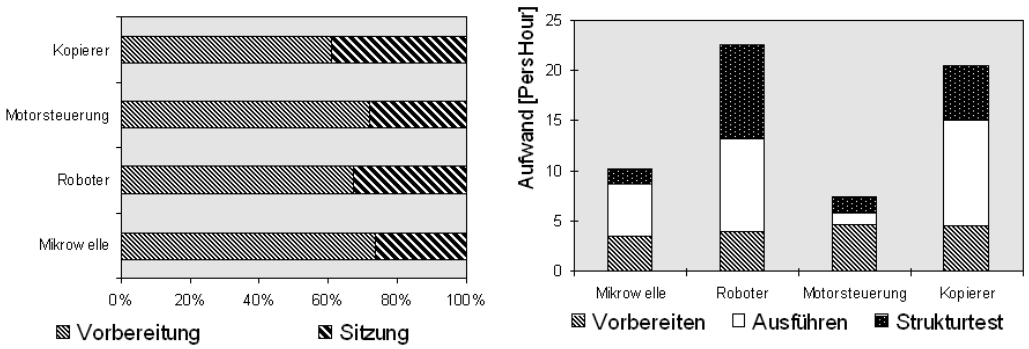



**Abb. 3.1:** Aufwand für die Softwareprüfung in Personenstunden.




[a] [b]

**Abb. 3.2:** Verteilung der Aufwände auf die einzelnen Phasen einer Inspektion [a] bzw. eines Tests [b]. Berücksichtigt wurden nur die Teile, an denen Studenten beteiligt waren.

nicht. Der wesentliche Grund hierfür ist in der entsprechenden Gruppe (G2) zu suchen. Die Studenten, die die geringen Testaufwände verursachten (insbesondere bei der Motorsteuerung), waren deutlich weniger motiviert als die Teilnehmer der andere Gruppe (G1). Dies zeigt sich auch in den geringeren Inspektionsaufwänden der Gruppe G2 (Roboter, Kopierer). Beim Vergleich mit Daten aus einer frühreren Untersuchung wird sich zeigen, daß signifikante Unterschiede eher dem beobachteten Mittel entsprechen.

Die 70 %:30 % Aufwandsverteilung für Inspektionsvorbereitung und Inspektionssitzung deckt sich mit Beobachtungen früherer Untersuchungen (vgl. [EHSS97, EHSa, EHSb]). Betrachtet man die Aufwandsverteilung beim Test (wobei wir die Motorsteuerung aus o. a. Gründen nicht beachten wollen), so fällt auf, daß die Testausführung, d. h. das Ausführen der Testfälle und das Analysieren der Ergebnisse einen erheblichen Teil der Aufwände auf sich vereint. Ein wesentlicher Grund hierfür ist sicher die nicht optimale Testumgebung und die Tatsache, daß die Auswertung der Testergebnisse manuell erfolgt ist. Da die Ausgaben aber komplex und zeitbehaftet sind, würde eine Automatisierung dieses Schrittes komplexe und proprietäre Werkzeuge erfordern.

Die Anteile der Mängel, die mit den jeweiligen Techniken identifiziert wurden, sind in

Abbildung 3.3 aufgetragen (bezogen auf testbare Mängel). Dabei haben wir zusätzlich eine Teilklasse aller Mängel betrachtet, nämlich die Klasse der kritischen und schwerwiegenden Mängel (kurz: KS–Mängel). Diese Mängel zeichnen sich dadurch aus, daß sie wesentliche Teile der Funktionalität beeinträchtigen oder sogar Leib und Leben in Gefahr bringen (z. B. Betrieb der Mikrowellenspule bei geöffneter Tür).

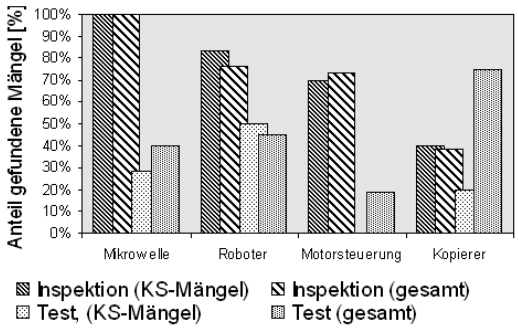



**Abb. 3.3:** Anteil der gefundenen Mängel (allgemein und bezogen auf KS–Mängel).

Interessant ist hier, daß auch die Gruppe, die mit weniger Engagement an dem Experiment beteiligt war, mittels Inspektionen dennoch einen beachtlichen Teil der enthaltenen Mängel entdecken konnte.

Die Effizienz der betrachteten Techniken, d. h. die Fehlerfindungsrate (Mängel pro Zeiteinheit), haben wir in Abbildung 3.4 graphisch dargestellt.

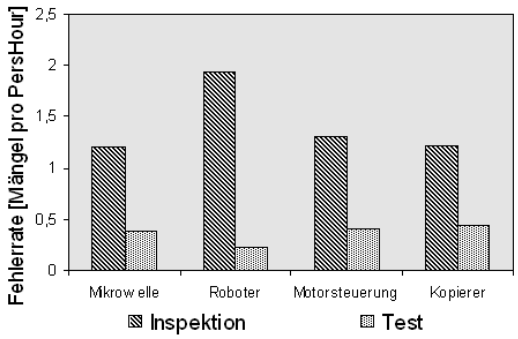


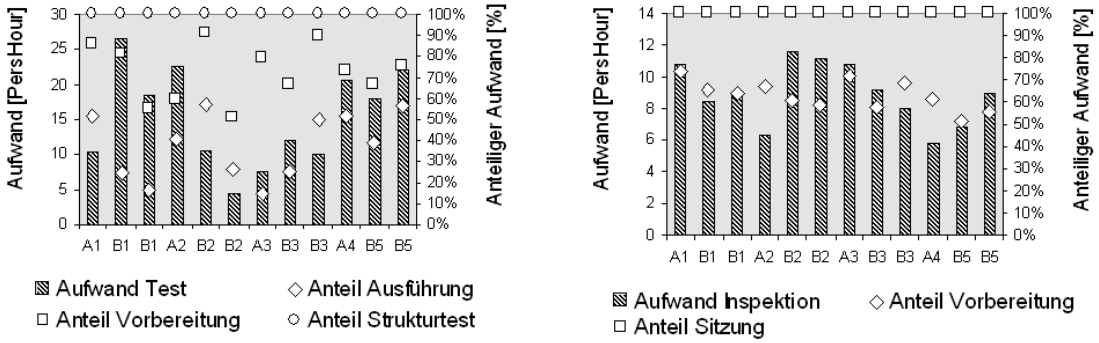

**Abb. 3.4:** Effizienz der Techniken (gemessen durch Anzahl der entdeckten Mängel pro Zeiteinheit).

### Vergleich mit der Untersuchung WS 1996/97

Wie bereits erwähnt, hatten wir den Teil unserer Untersuchung, der sich mit C–Code beschäftigt, auch zu einem früheren Zeitpunkt durchgeführt (mit einem Unterschied: anstelle des Kopierers wurde eine elektronische Mischpatrone betrachtet). In diesem Abschnitt werden wir nun unsere o. a. vorgestellten Ergebnisse den älteren Beobachtungen

[EHSS97] gegenüberstellen[5].

In Abbildung 3.5[a] haben wir die absoluten Testaufwände und deren Aufteilung auf die einzelnen Phasen dargestellt. Zwischen der aktuellen ($A$) und vorangegangenen ($B$) Untersuchung lassen sich keine systematischen Unterschiede beobachten (Mittelwert Aufwand $A$: 15,2 Ph, $B$: 15,2 Ph, $p = 0{,}987$[6]). Auch bei den Inspektionsaufwänden lassen sich keine




[a] [b]

**Abb. 3.5:** Absolute Aufwände (schraffierte Balken) und Aufwandsverteilung (offene Kreise, Rauten und Quadrate) bei Test [a] und Inspektion [b] für die aktuelle Untersuchung (Spalten A$x$) und vorangegangene Untersuchung (Spalten B$x$). Die Ziffer $x$ bezeichnet das betrachtete System (1) Mikrowelle, (2) Roboter, (3) Motorsteuerung, (4) Kopierer und (5) Mischpatrone. Die anteiligen Aufwände für die einzelnen Phasen sind kumuliert dargestellt. Da in der vorangegangenen Untersuchung mehr Studenten teilnahmen, wurde dort jedes System mit jeder Technik von je zwei Gruppen geprüft.

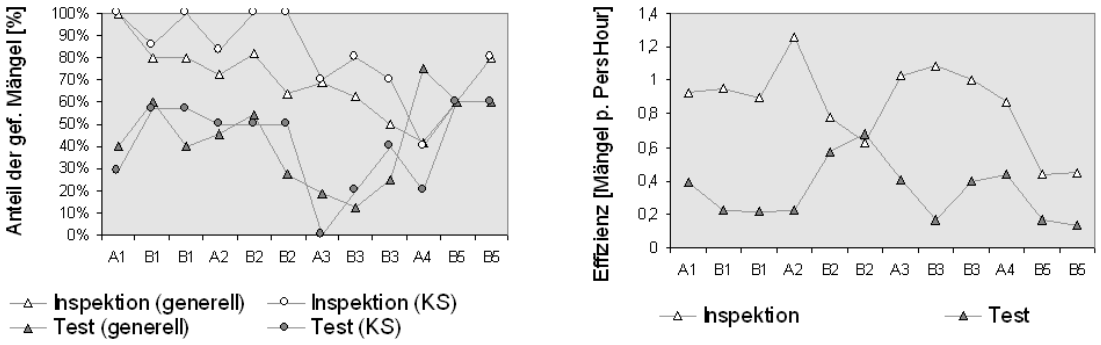
signifikanten Unterschiede zwischen den Untersuchungen $A$ und $B$ feststellen (Mittelwert Aufwand $A$: 9,1 Ph, $B$: 8,4 Ph, $p = 0{,}562$). Die absoluten und phasen–relativen Aufwände finden sich in Abbildung 3.5[b].

Effektivität und Effizienz der Untersuchungen haben wir in den Abbildungen 3.6[a] und 3.6[b] dargestellt. Signifikante Abweichungen zwischen den Untersuchungen können auch hier nicht beobachtet werden (Effektivität: $p = 0{,}860$, Effizienz: $p = 0{,}113$). Es ist also sowohl vom intuitiven wie auch statistischen Standpunkt aus zulässig, beide Untersuchungen zusammenzufassen, um so zu gesicherteren Aussagen zu gelangen.

In Tabelle 3.1 haben wir die Wertebereiche, Mittelwerte und Standardabweichungen der hier betrachteten Größen Aufwand, Effektivität und Effizienz für die Untersuchungen $A$

[5] Die Werte, die sich auf die frühere Untersuchung beziehen, weichen teilweise von den in [EHSS97] angegebenen Werten ab. Dies liegt daran, daß im Zuge der Analyse dieser Untersuchung, auch die frühere Untersuchung teilweise nach–analysiert wurde (z. B. Anwendung des erweiterten Fehlermodells). Die neuen Erkenntnisse (z. B. bzgl. Fehler eines Systems) führten zwangsläufig zu geänderten Werten. Die generellen Aussagen aus [EHSS97] haben aber nach wie vor Gültigkeit.

[6] Die $p$–Werte bezeichnen immer Überschreitungswahrscheinlichkeiten im Zuge einer einfaktoriellen Varianzanalyse (ANOVA) zum Test der Nullhypothese *Der betrachtete Faktor unterscheidet sich nicht in Hinblick auf Test und Inspektion.* Dieser Wert ist die Wahrscheinlichkeit dafür, daß die Abweichungen zwischen den beiden betrachteten Wertegruppen nur aufgrund zufälliger Ereignisse, nicht aber eines systematischen Unterschieds bestehen. Ist der Wert kleiner als 0,1, so gelten die Unterschiede als signifikant.




[a] [b]

**Abb. 3.6:** Effektivität [a] und Effizienz [b] der Untersuchungen $A$ und $B$. Die Verbindungslinien zwischen den einzelnen Punkten dienen lediglich der besseren Identifikation der einzelnen Datenpunkte. Sie stellen keine Trends dar.

und $B$ getrennt und zusammengefaßt aufgetragen. Außerdem finden sich dort die Überschreitungswahrscheinlichkeiten $p$ der Tests auf Unterschiede zwischen Test und Inspektion. Diese belegen, daß die Unterschiede zwischen Test und Inspektion in Hinblick auf alle vier Kategorien signifikant sind.

| | | Untersuchung $A$ | | | Untersuchung $B$ | | |
|---|---|---|---|---|---|---|---|
| | Technik | Bereich | Ø | $\sigma$ | Bereich | Ø | $\sigma$ |
| Aufwand | Test | 4,4 – 26,5 | 15,2 | 7,2 | 7,4 – 22,5 | 15,2 | 7,4 |
| | Inspektion | 6,8 – 11,6 | 9,1 | 1,6 | 5,8 – 10,8 | 8,4 | 2,7 |
| Effektivität (generell) | Test | 13 % – 60 % | 43 % | 19 % | 19 % – 75 % | 45 % | 23 % |
| | Inspektion | 50 % – 82 % | 70 % | 12 % | 42 % – 100 % | 71 % | 24 % |
| Effektivität (KS) | Test | 20 % – 60 % | 49 % | 14 % | 0 % – 50 % | 25 % | 21 % |
| | Inspektion | 60 % – 100 % | 84 % | 15 % | 40 % – 100 % | 73 % | 25 % |
| Effizienz | Test | 0,44 – 1,08 | 0,78 | 0,25 | 0,87 – 1,26 | 1,02 | 0,17 |
| | Inspektion | 0,14 – 0,68 | 0,32 | 0,21 | 0,22 – 0,44 | 0,36 | 0,09 |

| | | Kombinierte Untersuchung | | | |
|---|---|---|---|---|---|
| | Technik | Bereich | Ø | $\sigma$ | $p$ |
| Aufwand | Test | 4,4 – 26,5 | 15,2 | 7,0 | 0,006 |
| | Inspektion | 5,8 – 11,6 | 8,9 | 1,9 | |
| Effektivität (generell) | Test | 13 % – 75 % | 43 % | 19 % | 0,001 |
| | Inspektion | 42 % – 100 % | 70 % | 16 % | |
| Effektivität (KS) | Test | 0 % – 60 % | 41 % | 20 % | $< 0{,}001$ |
| | Inspektion | 40 % – 100 % | 81 % | 19 % | |
| Effizienz | Test | 0,44 – 1,26 | 0,86 | 0,25 | $< 0{,}001$ |
| | Inspektion | 0,14 – 0,68 | 0,33 | 0,17 | |

**Tab. 3.1:** Gegenüberstellung der signifikanten Größen der beiden Untersuchungen durch deren charakterisierenden Werte (Wertebereich, Mittelwert Ø, Standardabweichung $\sigma$).

## 3.2 Fehlerverteilung nach Prüftechnik

Eine wichtige Frage, die im Zusammenhang mit der Auswahl geeigneter Prüftechniken immer wieder artikuliert wird, bezieht sich auf die Fehlerklassen, die mit einer bestimmten Technik gefunden werden. Die Annahme, daß jeder Fehler mit jeder Technik gefunden werden kann, ist falsch. Myers [Mye78] berichtet von einer vergleichenden Untersuchung bzgl. Inspektion und Test, bei der die Fehlerklassen weitestgehend disjunkt waren.

In Abbildung 3.7 haben wir auf der $x$–Achse alle Fehler, die in den vier Systemen enthalten waren, aufgetragen. Die darunterliegenden Balken geben an, mit welcher Technik welche Fehler gefunden wurden. Die Beobachtung von Myers wird durch unsere Untersuchung also gestützt. Unsere ursprüngliche Hypothese, daß die mit Inspektionen entdeckten Mängel eine Oberklasse der Test–Mängel sind, ist somit nicht richtig. Typische Beispiele für „Ein–Technik–Fehler“ haben wir in Tabelle 3.2 zusammengetragen.

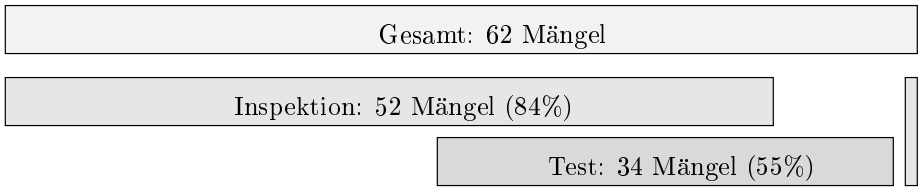



**Abb. 3.7:** Verteilung der enthaltenen Mängel nach der Technik, mit der diese gefunden wurden. Ein Mangel blieb in allen Prüfungen unentdeckt (schmaler Balken rechts). Der Datenbestand, der der Untersuchung zugrundeliegt, umfaßt die Untersuchungen $A$ und $B$.

| | |
|---|---|
| Test | • Unzulänglichkeiten in der Systemspezifikation (bei der Inspektion wird man immer sofort mit der Lösung konfrontiert, was zur Folge hat, daß diese Lösung selten hinterfragt wird)<br>• Mängel im dynamischen Regelungsverhalten (z. B. Einschwingzeiten) |
| Inspektion | • Unterlassungsfehler, die nicht oder nur schwer in den Systemausgaben erkannt werden können (z. B. fehlte bei der Robotersteuerung bei der Fahrtrichtungsumkehr ein gefordertes Stop–Kommando)<br>• Mängel in Berechnungsformeln (z. B. geringe Abweichungen in Konstanten)<br>• Mängel bezogen auf den Ressourcenverbrauch (z. B. Speicherplatz) |

**Tab. 3.2:** Typische Beispiele für „Ein–Technik–Fehler“.

Bei der Anwendung einer Technik spielt die Erfahrung und das Verständnis des Umfelds in der Regel eine wesentliche Rolle. Wir haben im Rahmen unserer Untersuchung versucht, diesen Aspekt dahingehend zu beleuchten, indem wir jeden enthaltenen Mangel mit dem zum Auffinden notwendigen Systemverständnis klassifiziert haben. Anschließend ha-

ben wir geprüft, ob es zwischen den Techniken merkliche Unterschiede gibt, d. h. ob eine Technik auch mit weniger erfahrenen Prüfern noch zufriedenstellende Resultate liefert.

Die Klassifikation bestand aus drei Klassen [SLBK99][7].

- *Hohes Systemverständnis erforderlich (Sem HIGH)*
  Das Auffinden dieser Mängel erfordert ein hinreichendes Systemverständnis. Typische Beispiele für solche Mängel sind Fehler in Regelungsalgorithmen.
- *Geringes Systemverständnis erforderlich (Sem LOW)*
  Diese Mängel können auch von weniger erfahrenen Personen gefunden werden. Die Lösung (d. h. das Programm) muß nicht bis ins letzte Detail verstanden werden. Typische Beispiele für derartige Mängel sind Inkonsistenzen zur Systemspezifikation (*„in der Spezifikation steht, daß dreimal gemessen werden soll, die Schleife umfaßt aber fünf Messungen“*).
- *Kein Systemverständnis erforderlich (Syntax)*
  Diese Mängel können auch von Fachfremden (mit hinreichender Programmiererfahrung) entdeckt werden. Mängel in dieser Klasse sind beispielsweise strukturelle Mängel (Schnittstellen) oder Inkonsistenzen im Programm (dieselbe Aufgabe wird unterschiedlich gelöst).

In Abbildung 3.8 haben wir die Klassifikation der Mängel (20 Sem HIGH, 36 Sem LOW, 6 Syntax) zusammen mit den jeweils entdeckten Anteilen aufgetragen. Während es bei Mängeln, die ein hohes Systemverständnis erfordern, so gut wie keine Unterschiede gab, sieht das Bild für „weniger anspruchsvolle“ Mängel differenzierter aus. Mit statischen Techniken wurden hier deutlich mehr Mängel entdeckt. Dieses Ergebnis deckt sich auch mit anderen Arbeiten (z. B. [DM94]).

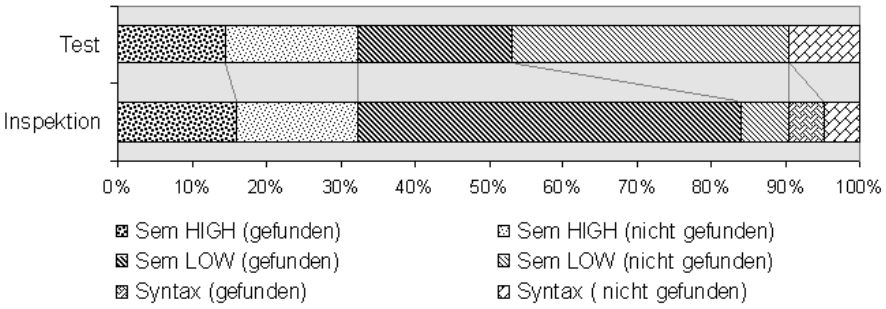



**Abb. 3.8:** Verteilung der Mängel nach dem zur Entdeckung benötigtem Systemverständnis (gesamt und anteilig für Inspektion und Test). Der Datenbestand, der der Untersuchung zugrundeliegt, umfaßt die Untersuchungen $A$ und $B$.

[7] Bei dieser Klassifikation ist es wichtig, diese nicht mit etwaigen Auswirkungen gleichzusetzten. Fehler in jeder dieser Klassen können bedeutende oder unbedeutende Auswirkungen haben.

# 4 Ergebnisse *Prüfen ausführbarer Spezifikationen*

## 4.1 Aufwand, Effektivität, Effizienz

Die Rohdaten, die den folgenden Analysen zugrunde liegen, finden sich im Anhang in Tabelle A.2. In Abbildung 4.1 sind die Mittelwerte zu Aufwands-, Effektivitäts- und Effizienzdaten graphisch aufbereitet. Auffällig ist hier der deutlich geringere Aufwand für die Ad–Hoc Prüfung (Abbildung 4.1[a]). Dieser hängt zum einen damit zusammen, daß im Gegensatz zu den Inspektionen nur eine Person beteiligt war, und zum anderen, daß der Planungs- und Dokumentationsaufwand im Vergleich zum Test geringer war.

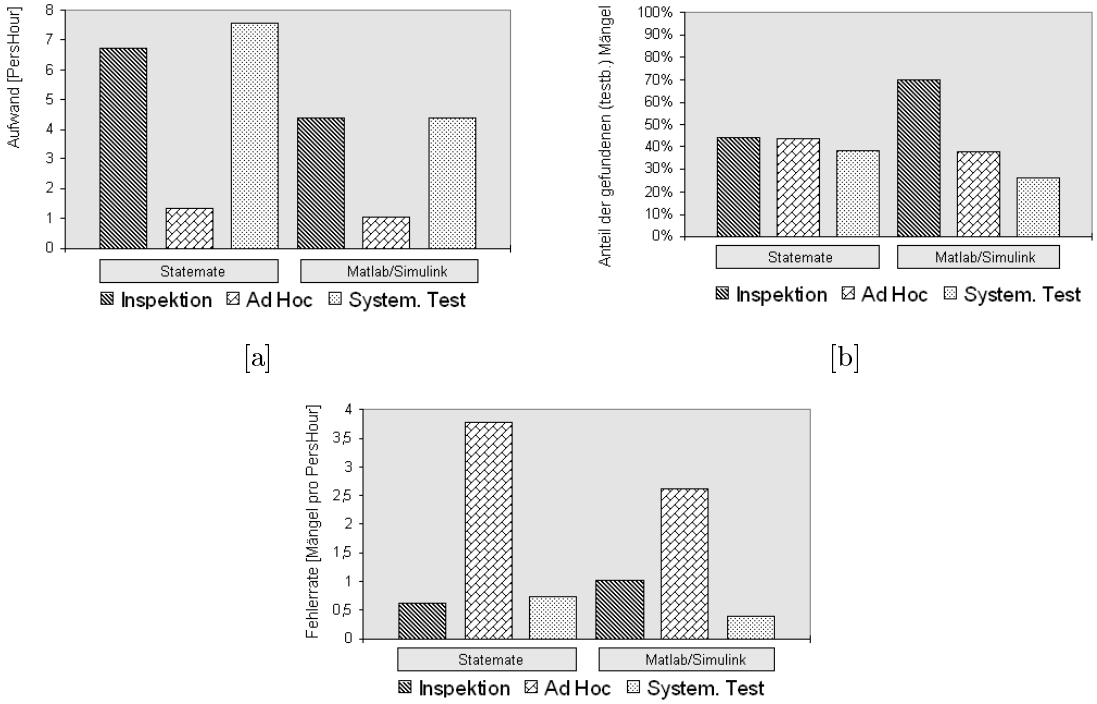



**Abb. 4.1:** Aufwand in Personenstunden [a], Anteil der gef. testbaren Mängel [b], Fehlerrate [c].

In Hinblick auf die gefundenen Mängel (Abbildung 4.1[b]) sind dagegen nur geringe Unterschiede feststellbar. Damit ist auch die Fehlerrate der Ad–Hoc Prüfung deutlich höher als bei den anderen eingesetzten Techniken (Abbildung 4.1[c]).

Wendet man eine einfaktorielle Varianzanalyse an zur Prüfung der Hypothese *„Die eingesetzten Methoden unterscheiden sich nicht in Hinblick auf Aufwand, Anteil der gefundenen Mängel und Mängelrate“*, so kann man diese Hypothese in allen drei Fällen verwerfen. Die Unterschiede sind also signifikant.

### 4.2 Fehlerverteilung nach Prüftechnik

In Abbildung 4.2 haben wir die Mängel den Techniken, mit denen sie entdeckt wurden, zugeordnet. Dabei ergeben sich vier Überlappungsbereiche, nämlich Mängel, die mit allen Techniken gefunden wurden, sowie Mängel, die mit zwei Techniken identifiziert wurden.

| | | | |
|---|---|---|---|
| Innerer Kreis | | | |
| | Nur Inspektion | 18 | (33 %) |
| | Inspektion und Ad–Hoc Sim. | 8 | (15 %) |
| | Nur Ad–Hoc Simulation | 8 | (15 %) |
| | Ad–Hoc Sim. und Test | 6 | (11 %) |
| | Nur Test | 4 | (7 %) |
| | Test und Inspektion | 3 | (5 %) |
| | Mit allen Techniken | 7 | (13 %) |
| | Mit keiner Technik | 1 | (2 %) |
| Äußere Segmente | | | |
| | Inspektion (gesamt) | 36 | (65 %) |
| | Ad–Hoc Sim. (gesamt) | 29 | (53 %) |
| | Test (gesamt) | 20 | (36 %) |

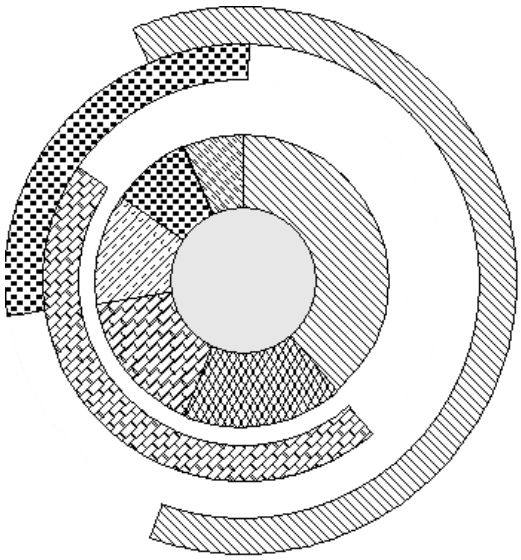


**Abb. 4.2:** Verteilung, wie viele Mängel mit welcher Prüftechnik bzw. welchen Prüftechniken gefunden wurden. Der innere Kreis steht für die Menge aller identifizierten Mängel. So wurden beispielsweise 18 Mängel nur mit Inspektionen gefunden. Die äußeren Segmente geben an, welcher Mangel überhaupt mit einer Technik gefunden wurden. *Beispiel:* Mit Inspektionen wurden insgesamt 36 = 18 (nur Inspektion) + 8 (Insp. und Ad–Hoc) + 3 (Insp. und Test) + 7 (alle Techniken) Mängel identifiziert.

Auffällig ist hier, daß keine Technik einen Großteil der Mängel abdeckt. Auch die Überlappungsbereiche (d. h. der Anteil der Mängel, der mit mehr als einer Technik gefunden wird), sind mit 44 % relativ gering. Dies ist ein Indiz dafür, daß keine der Techniken für sich betrachtet optimal ist, sondern eine Kombination aus verschiedenen Techniken sinnvoll scheint (siehe auch Abschnitte 6 und 7). Die „Obermengenhypothese“ ist damit auch hier nicht zutreffend.

## 5 Beobachtungen zu Inspektionen

### 5.1 Restfehlerschätzung

Mit einer Restfehlerschätzung soll versucht werden, aufgrund der aktuell entdeckten Mängel Rückschlüsse auf die noch verbleibenden Mängel zu ziehen. In der Literatur finden sich hierzu verschiedene Modelle.

Die eine Art der Modelle, die sogenannten Capture–Recapture Modelle (vgl. [BEFL97]), basieren auf Modellen aus der Biologie. Dort wird diese Technik zur Schätzung von Tierpopulationen eingesetzt. An einem Tag werden alle beobachteten Tiere mit einer Markierung versehen. An einem anderen Tag wird dann gezählt, wie viele markierte Tiere gesichtet werden. Aus dem Verhältnis markierte und nicht-markierte Tiere wird dann auf die Gesamtpopulation geschlossen. Die Formel hierzu lautet

$$n_{ges} = \frac{n_1 \cdot n_2}{n_{gem}},$$

wobei $n_i$ die Anzahl der am Tag $i$ gesichteten Tiere ist, $n_{gem}$ die Zahl der Tiere, die an beiden Tagen gesehen wurde, und $n_{ges}$ die Schätzung der Gesamtzahl ist.

Die zweite Art der Restfehlermodelle beruht auf (geometrischen) Verteilungsannahmen der Zahl der Mängel, die von verschiedenen Inspektoren gefunden werden [WR98]. Auf diese Art der Modelle wollen wir hier nicht näher eingehen.

Wir haben nun in unserer Untersuchung das Capture–Recapture Modell eingesetzt, um auf Basis der individuellen Vorbereitung der Inspektoren (entspricht der unabhängigen Beobachtung der Tiere) auf die Gesamtfehler hochzurechnen. In den Tabellen 5.1 und 5.2 finden sich die Schätzdaten sowie die tatsächlichen Fehlerzahlen für unsere C–Code Systeme sowie die ausführbaren Spezifikationsdokumente.

| System | $n_1$ | $n_2$ | $n_3$ | $n_{gem}$ | Schätzung | tatsächliche Mängel | protokollierte Mängel |
|---|---|---|---|---|---|---|---|
| Mikrowelle | 10 | 7 | 7 | 4 (8) | 15,3 | 13 | 13 |
| Roboter | 6 | 11 | — | 3 | 13,2 | 17 | 13 |
| Motorsteuerung | 10 | 5 | 8 | 3 (6) | 22,2 | 19 | 14 |
| Kopierer | 3 | 3 | — | 1 | 9,0 | 13 | 5 |

**Tab. 5.1:** Restfehlerschätzung mit dem Capture–Recapture Modell für C–Code. Die Gesamtfehlerzahl bezieht sich auf die Summe aus testbaren und nichttestbaren Mängel. Die Angabe $x(y)$ bei $n_{gem}$ ist relevant für drei Inspektoren und bedeutet, daß $x$ Mängel von allen Inspektoren gefunden wurden und $y$ Mängel von mindestens zwei Inspektoren. Die erweiterte Schätzformel lautet dann $n_{ges} = (n_1 \cdot n_2 \cdot n_3)/(n_{gem} \cdot n_{min2})$ mit $n_{min2}$ ist die Anzahl der Mängel, die von mindestens zwei Inspektoren gefunden wurde.

Bei der Schätzung für C–Code Dokumente fällt auf, daß die Abweichung Schätzwerte zu tatsächlicher Mängelzahl zwischen -31 % und +17 % schwankt. Bei mehr Inspektoren ist das Schätzergebnis genauer (eine Beobachtung, die sich mit Literaturergebnissen, z. B. [BEFL97] deckt).

Bei der Schätzung für Spezifikationsdokumente ist das Ergebnis weniger beeindruckend. Allerdings weisen die „sinnvollen“ Schätzungen auch nur eine Abweichung von maximal 37 % aus. Berücksichtigt man nun aber noch die Verteilung der Fehler nach Mängeln (vgl. Abschnitt 4.2), so sehen die Schätzungen deutlich besser aus.

| System | $n_1$ | $n_2$ | $n_{gem}$ | Schätzung | tatsächliche Mängel | protokollierte Mängel |
|---|---|---|---|---|---|---|
| Anrufbeantworter | 4 | 3 | 1 | 12,0 | 16 | 6 |
| Innenlicht | 5 | 4 | 3 | 6,7 | 8 | 5 |
| Liftsteuerung | 5 | 2 | 0 | $\infty$ | 11 | 8 |
| Tempomat | 5 | 3 | 3 | 5,0 | 8 | 5 |
| Fenstersteuerung | 2 | 1 | 0 | $\infty$ | 7 | 5 |
| Dampfkessel | 4 | 2 | 2 | 4,0 | 5 | 4 |

**Tab. 5.2:** Restfehlerschätzung für ausführbare Spezifikationsdokumente. Die Schätzung $\infty$ ergibt sich, wenn die Inspektoren nur disjunkte Mängel melden.

## 5.2 Synergieeffekte *(Review Gains)*

Von Synergieeffekten in einer Inspektionssitzung spricht man, wenn auf dem Inspektionsprotokoll Mängel auftauchen, die erst im Rahmen der gemeinsamen Sitzung identifiziert wurden, d. h. kein Inspektor auf seinem Vorbereitungsformular notiert hatte.

In Abbildung 5.1 haben wir den Anteil der Synergieeffekte in unserer Untersuchung (gesamt und Teilbereiche) aufgetragen. Diese Anteile sind auf den ersten Blick sehr hoch

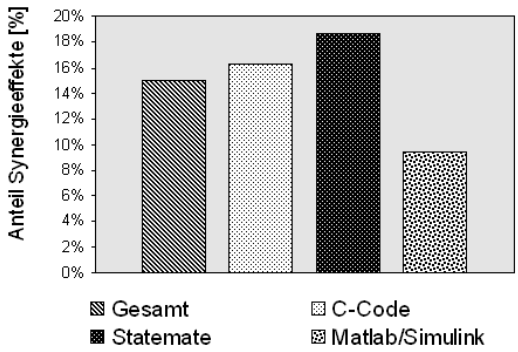



**Abb. 5.1:** Beobachtete Synergieeffekte.

und bestätigen den Eindruck, daß Inspektionssitzungen unbedingt notwendig sind. Betrachtet man aber die Arten der „Synergiemängel“, so fällt auf, daß es sich ausnahmslos um marginale Mängel handelt. Typische Beispiele sind unkommentierte Konstanten (*magic number*) und schlechte Variablennamen. Die Hypothese der aufgrund von Synergie notwendigen Inspektionssitzung wird dadurch nicht gestützt. Allerdings hat die Inspektionssitzung andere, hier nicht näher untersuchte Vorteile. So dient sie beispielsweise der Schaffung eines gemeinsamen Verständnisses zwischen den Inspektoren.

### 5.3 Nicht–Meldungen *(Review Losses)*

Generell ist das Identifizieren ungerechtfertigter Meldungen eine wichtige Aufgabe der Inspektionssitzung. Dabei besteht aber auch die Gefahr, zu viele Meldungen zu entfernen.

Unter Nicht–Meldungen verstehen wir Meldungen, die ein Inspektor auf seinem Vorbereitungsformular notiert hatte, die dann aber nicht ins Inspektionsprotokoll mit aufgenommen wurden, obwohl es sich um echte Mängel handelte.

In unserer Untersuchung sind ca. 10 % aller in der Vorbereitung identifizierten Mängel nicht in die Inspektionsprotokolle übernommen worden. Von diesen waren wiederum ca. 10 % echte Mängel, die zu unrecht verworfen wurden.

### 5.4 Falschmeldungen *(Faults positive)*

Die Identifikation von Mängeln basiert auf dem Verständnis der einzelnen Inspektoren. Dabei besteht natürlich die Gefahr, daß ein falsches Verständnis ungerechtfertigte Meldungen hervorbringt.

Unter Falschmeldungen verstehen wir diejenigen Meldungen, die ins Inspektionsprotokoll aufgenommen worden sind, obwohl es sich um keinen echten Mangel handelt. In unserer Untersuchung waren 3 von 370 Meldungen auf den Inspektionsprotokollen solche Falschmeldungen ($< 1$ % aller Meldungen).

## 6 Kritische Betrachtung des Experiments

Bevor wir uns konkret mit den Experimentergebnissen auseinandersetzen, wollen wir zuerst die *interne Gültigkeit*[8] unserer Untersuchung betrachten.

Durch den Aufbau des Experiments haben wir versucht, störende Einflußfaktoren (wie z. B. Unterschiede zwischen den Teams) durch einen wechselseitigen Einsatz der verschiedenen Prüftechniken zu minimieren. Die zufällige Zuordnung der Personen zu den einzelnen Gruppen unterstützte ebenfalls dieses Ziel.

Ein weiterer möglicher Störfaktor für die interne Gültigkeit ist der Informationsaustausch zwischen den Beteiligten (z. B. Austausch entdeckter Fehler). Diesen Faktor konnten wir nicht vollständig kontrollieren. Wir glauben aber, daß die Studenten nicht versucht haben, die Untersuchung zu untergraben (z. B. um uns eine bessere eigene Leistung vorzuspielen). Diskussionen mit den Studenten bestärken uns in dieser Annahme.

Ein letzter Störfaktor liegt in der Ungenauigkeit der Datenerhebung. Hier sind sicherlich einige Schwächen zu beobachten. So wurden die Aufwände immer erst am Ende eines einwöchigen Prüfabschnittes ermittelt. Größere Abweichungen von berichtetem und tatsächlichem Wert sind also wahrscheinlich. Da die Techniken aber sehr unterschiedliches

[8] Grad zu dem sinnvollerweise kausallogische Schlußfolgerungen aus dem Datenbestand gezogen werden dürfen, d. h. gibt es experimentinterne Faktoren, die die gemachten Schlüsse in Frage stellen (z. B. Meßfehler) [JSK91].

Verhalten zeigten, sind diese Meßfehler wenig kritisch, zumal wir an Trends und nicht an exakten Zahlen interessiert sind.

In den folgenden Abschnitten wollen wir nun — getrennt nach den großen Praktikumsabschnitten (Prüfen von C–Code und Prüfen ausführbarer Spezifikationen) — die jeweiligen Ergebnisse kritisch beleuchten und uns auch dem Aspekt der *externen Gültigkeit*[9] zuwenden.

## 6.1 Prüfen von C–Code

Auf den ersten Blick scheint das deutlich schlechtere Abschneiden der Testaktivitäten in unserer Untersuchung in deutlichem Widerspruch zur industriellen Praxis zu stehen. So sind nach einer an der Universität Köln durchgeführten Untersuchung [Mül98] Testaktivitäten Bestandteil der Entwicklungsprozesse in mehr als 82 % der betrachteten Softwareunternehmen, während statische Prüftechniken nur in 49 % Unternehmen eingesetzt werden (bezogen auf Dokumenten–Reviews; bei Codeinspektionen oder Walkthroughs beträgt der Anteil weniger als ein Viertel).

Für das schlechte Abschneiden der Testaktivitäten in unserer Untersuchung gibt es eine Reihe von Erklärungsmöglichkeiten:

- In unserem Experiment konnten die dynamischen Prüftechniken nicht ihre volle Stärke ausspielen. Die Testumgebungen waren sicherlich nicht ideal, so daß hier mehr mißlungene Testläufe (z. B. Syntaxfehler im Eingabeskript) und ein merklich höherer Aufwand nötig waren.

- Die Interpretation der Ausgaben eines Testlaufs ist aufwendig und fehlerträchtig (und damit wenig attraktiv).

- Wir haben jeden Test nur einmal durchgeführt. In der praktischen Anwendung sollten Testfälle (idealerweise) nach jeder Überarbeitung wieder neu „gefahren“ werden. Dabei ist der Aufwand für einen erneuten Test deutlich geringer als beim Ersttest.

- Die beteiligten Personen mußten die Ergebnisse der Prüfaktivitäten nicht dazu verwenden, das Produkt marktfähig zu machen. Ohne diesen Druck ließ aber auch das Interesse an optimalen Prüfergebnisse nach. Da der (Erst-) Test aber mit höheren Aufwänden bezogen auf Inspektion und Ad–Hoc Simulation verbunden war, blieb dieser „auf der Strecke“.

Doch selbst wenn man diese Faktoren berücksichtigt und die Ergebnisse entsprechend modifiziert, bleibt das gute bis sehr gute absolute Abschneiden der statischen Prüfaktivitäten erhalten; auch ein relativer Vorteil der statischen gegenüber den dynamische Prüfaktivitäten bleibt bestehen.

[9] Grad, zu dem die Ergebnisse eines Experiments auch außerhalb des Laborumfelds Gültigkeit haben.

Nun spiegelt die Wahl der Prüftechniken in der industriellen Praxis aber nicht zwingend die *best–practice* wider. Ein wesentlicher Grund für die Wahl der dynamischen Prüfung liegt im Bereich der menschlichen Psyche. Nur durch eine dynamische Prüfung kann man sich davon überzeugen, daß der Prüfling das macht, was er machen soll. Die statische Betrachtung läßt dies immer nur indirekt zu; ein schaler Nachgeschmack bleibt. Außerdem will auch ein Auftraggeber „sehen“, daß das Produkt seinen Anforderungen gerecht wird. Testen ist also unausweichlich.

Das gute Abschneiden der statischen Prüftechniken deckt sich auch mit den Beobachtungen in anderen Kontexten (vgl. [WBM96]). Auch wenn die konkreten Zahlen die Situation sicher überzeichnen, so kann doch festgehalten werden, daß statische Prüfverfahren auch im Bereich eingebetteter Systeme gewinnbringend eingesetzt werden können.

## 6.2 Prüfen ausführbarer Spezifikationen

Unsere Untersuchung hat neben den Zahlen zu den verschiedenen Prüftechniken auch zu einer Reihe interessanter Erkenntnisse in Hinblick auf die Umsetzbarkeit der Techniken bei den verschiedenen Dokumentarten geführt.

### 6.2.1 Statemate

- Inspektionen auf Statemate–Dokumenten sind prinzipiell durchführbar, allerdings erschweren die verteilten Informationsquellen (Data Dictionary, Statechart) das Lesen erheblich. Insbesondere folgt die Informationsverteilung keinem festen Schema, sondern hängt von der Modellierungsweise des Erstellers ab. Mit besseren Ausgaben könnte man hier sicher noch einiges verbessern.
- Ein funktionaler Test ist gut machbar. Die dazu notwendige Beschreibung der Testfälle ist mittels Skriptsprache möglich, wenn auch aufwendig. Der strukturelle Test, der eigentlich durch systemseitige Protokollierung der benutzten Zustände und Übergänge unterstützt wird, ist praktisch nicht durchführbar. Hier ist aber eine automatische Aufbereitung der erzeugten Ausgaben hin zu einer vernünftigen Überdeckungsaussage denkbar.
- Die Ad–Hoc Prüfung wird durch die im Werkzeug eingebauten Simulationsmöglichkeiten adäquat unterstützt. Insbesondere die Zeitliniendiagramme sind nützlich zur Analyse des Systemverhaltens.

### 6.2.2 Matlab/Simulink

- Die Ausgaben, die das Werkzeug produziert, sind für eine sinnvolle Inspektion wenig geeignet. So finden sich in der graphischen Ausgabe nicht alle benötigten Informationen (z. B. Grenzwerte eines Integrierers). Ein Nachprüfen dieser Werte in der (sehr schlecht lesbaren) textuellen Modellbeschreibung ist unumgänglich. Auch hier wäre eine wesentlich bessere Ausgabe wünschenswert.

- Ein funktionaler Test kann mit Hilfe der Skriptsprache von Matlab durchgeführt werden. Die Beschreibung als Zeit–Wert Matrix ist aber aufwendig und fehleranfällig, da nicht nur modifizierte Werte, sondern auch alle übrigen Werte angegeben werden müssen. Die Eigenschaft der Signalinterpolation[10] führt bei diskreten Signalen zu erhöhtem Schreibaufwand und potentiellen Falscheingaben. Für den strukturellen Test findet sich keinerlei Unterstützung. Die notwendigen Ausgaben (z. B. Schalterstellungen) müssen manuell erzeugt werden (z. B. durch Hinzufügen eines Ausgabeelementes).
- Die Ad–Hoc Simulation wird ebenfalls wenig unterstützt. Zwar gibt es eine Reihe interaktiver Schalt- und Ausgabeelemente, doch sind diese nicht immer angemessen. Insbesondere Taster und mehrstufige Schalter hatten wir schmerzlich vermißt.

Bei den eigentlichen empirischen Beobachtung fällt auf, daß die Fehlerklassen, die mit den verschiedenen Techniken gefunden werden, relativ wenig Überdeckung (vgl. Abschnitt 4.2) aufweisen. Ein Grund hierfür liegt sicher in der geringen Grundgesamtheit der Untersuchung. Bei mehr Daten werden sich die Bereiche sicher stärker überlappen.

Interessant ist die Beobachtung, daß der systematische Test wenig erfolgreich abgeschnitten hat. Ein wesentlicher Grund hierfür ist in jedem Fall die unzureichende Unterstützung durch die eingesetzten Werkzeuge (s. o.). Doch auch bei einer Verbesserung der Werkzeuge scheint ein systematischer Test für diese Dokumente nur sinnvoll zu sein, wenn die Testfälle in späteren Entwicklungsphasen wiederholt eingesetzt werden können. Dann allerdings könnte der Entwicklungs- und insbesondere der Prüfprozeß deutlich verbessert werden, da sich die Testfälle wirklich inkrementell mit der Systementwicklung weiterentwickeln.

Zusammenfassend kann für die ausführbaren Spezifikationsdokumente festgehalten werden, daß keine Technik allein hinreichend erfolgreich ist. In unserer Untersuchung scheint eine Kombination aus Inspektion und Ad–Hoc Simulation (vgl. Abschnitt 7) besonders erfolgversprechend zu sein. Die geschätzte Effektivität läge dann bei 93 %.

# 7 Zusammenfassung und Ausblick

In der hier beschriebenen Untersuchung haben wir verschiedene Aspekte der Softwareprüfung empirisch betrachtet, um so einen Beitrag zur geeigneten Auswahl und Kombination von Prüftechniken zu liefern.

Unsere Untersuchungen fand im Rahmen eines Hauptstudium–Praktikums an der Universität Ulm statt, da wir dort gezielt einzelne Faktoren, wie Prüftechnik oder Prüfling, beeinflussen konnten. In dem Experiment verglichen wir Inspektion und systematischen Test in Hinblick auf Aufwände, Effektivität (d. h. Anzahl der gefundenen Mängel), Effizienz (Mängel pro Zeit) und Fehlerklassen.

[10] Sind für zwei Zeitpunkte $t_1$ und $t_2$ die Werte einer Eingangsgröße $i$ mit $i_1$ und $i_2$ spezifiziert, so interpoliert Matlab für alle Zwischenzeitpunkte $t \in ]t_1, t_2[$ den Wert $i(t)$ linear.

Als Prüflinge setzten wir C–Code und ausführbare Spezifikationsdokumente, namentlich Statemate- und Matlab/Simulink-Modelle, aus dem Umfeld reaktiver Systeme ein. Bei den Spezifikationsdokumenten betrachteten wir als Technik zudem die Ad–Hoc Simulation.

Unsere Beobachtungen bei C–Code decken sich im wesentlichen mit einer früheren Untersuchung [EHSS97]. Mit Inspektionen werden bei geringerem Zeitaufwand deutlich mehr Mängel identifiziert.

Bei ausführbaren Spezifikationsdokumenten ist das Bild weniger klar. So sind zwar die Aufwände für eine Ad–Hoc Simulation signifikant niedriger, was aber aufgrund des einfacheren Ablaufs (nur eine Person beteiligt, wenig Planung) nicht überrascht. In Hinblick auf die gefundenen Mängel kann keine Technik alleine überzeugen.

Im Zuge der Nachbetrachtung des Experiments wurde daher von den Beteiligten der folgende Vorschlag für eine verbesserte Prüfstrategie gemacht.

- Kurze Simulation des Systems, um einen Eindruck von dessen Funktionsweise zu bekommen. Dies kann auch ein Element im Rahmen einer Einführungssitzung bei Inspektionen sein.
- Formale Inspektion, d. h. individuelle Vorbereitung und moderierte Inspektionssitzung. Unklarheiten und Fragen, insbesondere zum Laufzeitverhalten, werden dabei separat protokolliert.
- Diese Fragen werden in einer nachfolgenden Simulationssitzung gezielt untersucht.

Basierend auf den erhobenen Daten dürfte diese Strategie nur zu unwesentlich höheren Aufwänden führen. Die Effektivität sollte aber deutlich höher sein. Der Anteil an Mängeln, der in unserer Untersuchung damit nicht abgedeckt wäre, ist geringer als 7 %. In künftigen Untersuchungen werden wir diese Strategie näher betrachten.

Das schlechte Abschneiden des systematischen Tests für beide Dokumentarten ist verwunderlich. Neben methodenbedingter Schwächen (z. B. hoher Initialaufwand für Planung der Testfälle) spielt in unserer Untersuchung sicher die mangelnde werkzeugseitige Unterstützung hier eine wesentliche Rolle. Für das Werkzeug Statemate planen wir daher eine Ergänzung, die dessen Ausgaben zur Modellabdeckung aufbereitet und somit auch einen strukturellen Test adäquat unterstützt.

Positiv sind unsere Beobachtungen zur Capture–Recapture Methode als Mittel zur Restfehlerschätzung bei Inspektionen ausgefallen. Das Modell lieferte in unsere Untersuchung hinreichend gute Schätzwerte. Die Ausreißer (die sich durch unrealistische Schätzungen auszeichnen) können somit auch als Mittel zur Erkennung mangelhaft durchgeführter Inspektionen (z. B. unzureichende Vorbereitung, fachliche Unkenntnis) verwendet werden.

Basierend auf unseren Beobachtungen und Ergebnissen haben wir folgende Punkte für weitere Arbeiten identifiziert.

- Die Laborergebnisse müssen durch reale Ergebnisse bestätigt werden. Auf Basis unserer Arbeit in industriellen Projekten konnten wir zwar bereits eine Reihe von

Indizien beobachten, daß unsere Ergebnisse die „Wirklichkeit“ widerspiegeln, ein hinreichender Nachweis steht aber noch aus.

- Aufgrund der Heterogenität der Software–Branche sind keine allumfassende Aussagen zu erwarten. Vielmehr haben die individuellen Charakteristiken der Umgebungen einen wesentlichen Einfluß. Ziel weiterer Untersuchungen — im Labor wie im industriellen Umfeld — ist es daher, diese Faktoren weiter einzugrenzen. Dazu gehört die Einbeziehung neuer Techniken ebenso wie die Berücksichtigung anderer Dokumentarten.

- Das manuelle Anwenden der Capture–Recapture Methode ist sehr aufwendig. Insbesondere müssen verschiedenen Meldungen als übereinstimmende Mängel identifiziert werden. Eine werkzeugseitige Unterstützung wäre an dieser Stelle wünschenswert. Derartige Tools könnten dann auch andere Bereiche des Inspektionsprozesses sinnvoll unterstützten.

### Danksagung

Wir möchten uns an dieser Stelle ganz herzlich bei Herrn Prof. Partsch bedanken, der mit seinem Vertrauen in unsere Arbeit uns sehr große Freiheiten bei der Gestaltung und Durchführung unserer Untersuchung ermöglicht hat. Seinem steten Drängen ist es auch zu verdanken, daß die Ergebnisse nun endlich dokumentiert sind.

Unser Dank gilt auch „unseren“ Studenten, ohne die wir unsere Untersuchungen niemals hätten durchführen können.

## Literatur

[Alb86] A.J. Albrecht. *Measuring application development productivity.* In: C. Jones (Herausgeber): *Programming Productivity: Issues for the Eighties.* IEEE Computer Society Press, Los Alamitos, CA, 1986.

[AMDS97] A. Abran, M. Maya, J.-M. Desharnais und D. St-Pierre. *Adapting function points to real–time software.* American Programmer, 10(11):32–43, November 1997.

[BEFL97] L.C. Briand, K. El Emam, B. Freimut und O. Laitenberger. *Quantitative evaluation of capture–recapture models to control software inspections.* Technischer Bericht 053.97/E, IESE Fraunhofer Institute, Kaiserslautern, Germany, 1997.

[Boe81] B.W. Boehm. *Software engineering economics.* Englewood Cliffs, 1981.

[BS87] V.R. Basili und R.W. Selby. *Comparing the effectiveness of software testing strategies.* IEEE Transactions on Software Engineering, 13(12):1278–1296, 1987.

[DM94] H.E. Dow und J.S. Murphy. *Detailed product knowledge is not a prerequisite for an effective formal software inspection.* available at http://www.ics.hawaii.edu/johnson/FTR/Bib/Dow94.html, 1994.

[EH98] D. Ernst und F. Houdek. *Applying metrics to cross–technical evaluations.* In: *Proceedings of the European Software Measurement Conference (FESMA 98)*, Seiten 257–264, Antwerpen, 1998. Technologisch Instituut vzw.

[EHSa] D. Ernst, F. Houdek und T. Schwinn. *Experimenteller Vergleich strukturierter und objektorientierter Entwicklungsmethoden für eingebettete Systeme: Analyse.* Universität Ulm, (in Vorbereitung).

[EHSb] D. Ernst, F. Houdek und T. Schwinn. *Experimenteller Vergleich strukturierter und objektorientierter Entwicklungsmethoden für eingebettete Systeme: Entwurf und Implementierung.* Universität Ulm, (in Vorbereitung).

[EHSS97] D. Ernst, F. Houdek, T. Schwinn und W. Schulte. *Experimenteller Vergleich statischer und dynamischer Softwareprüfung.* Technischer Bericht 97–13, Universität Ulm, 1997.

[FP96] N. Fenton und S.L. Pfleeger. *Software metrics — A rigorous and practical approach.* International Thomson Computer Press, London, 2. Auflage, 1996.

[GG93] T. Gilb und D. Graham. *Software Inspections.* Addison–Wesley, Reading, Masachusetts, 1993.

[GW95] M. Grochtmann und J. Wegener. *Test Case Design Using Classification Trees and the Classification–Tree Editor CTE.* In: *Proceedings of the Eighth International Software Quality Week, San Franscisco*, 1995.

[HLN+90] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring und M. Trakhtenbrot. *STATEMATE: A Working Environment for the Development of Complex Reactive Systems.* IEEE Transactions on Software Engineering, 16(4):403–414, April 1990.

[Hof98] J. Hoffmann. *MATLAB und SIMULINK.* Addison–Wesley, 1998.

[HP96] D. Harel und M. Politi. *Modeling Reactive Systems with Statecharts: The Statemate approach.* i–Logix Inc., 1996.

[JSK91] C. Judd, E.R. Smith und L. Kidder. *Research methods in social relations.* Holt, Rinehart and Winston, 6. Auflage, 1991.

[KL95a] E. Kamsties und C. Lott. *An empirical evaluation of three defect–detection techniques.* Technischer Bericht ISERN-95-02, University of Kaiserslautern, 1995.

[KL95b] E. Kamsties und C. Lott. *An expirical evaluation of three defect–detection techniques.* In: W. Schäfer und P. Botella (Herausgeber): *Proceedings of the 5th European Software Engineering Conference*, Nummer 989 in *Lecture Notes in Computer Science*, Seiten 362–383. Springer–Verlag, September 1995.

[LRRA98] P. Liggesmeyer, M. Rothfelder, M. Rettelbach und T. Ackermann. *Qualitätssicherung Software–basierter technischer Systeme — Problembereich und Lösungsansätze.* Informatik Spektrum, 21(5):249–258, 1998.

[Mar91] B. Marick. *Experience with the Cost of different Coverage Goals for Testing.* Pacific Northwest Software Quality Conference, Oktober 1991. Available from ftp://cs.uiuc.edu/pub/testing/experience.ps.Z.

[Mar92] B. Marick. *Generic Coverage Tool (GCT) User's Guide: Documentation for version 1.4 of GCT*, 1992. Available from ftp://cs.uiuc.edu/pub/testing/gct.files/doc.ps.tar.Z.

[Mül98] U. Müller. *Prüfen und Testen von Software in Deutschland. Stand der Praxis — Ergebnisse einer Umfrage.* Softwaretechnik–Trends, 18(2):12–15, 1998.

[Mye78] G.J. Myers. *A Controlled Experiment in Program Testing and Code Walkthroughs/Inspections.* Communications of the ACM, 21(9):760–768, September 1978.

[Mye91] G.J. Myers. *Methodisches Testen von Programmen.* Oldenbourg Verlag, 4. Auflage, 1991.

[Pfl95] S.L. Pfleeger. *Experimental design and analysis in software engineering.* Annals of Software Engineering, 1:219–253, 1995.

[PVB95] A.A. Porter, L.G. Votta und V.R. Basili. *Comparing detection methods for software requirements inspections: A replicated experiment.* IEEE Transactions on Software Engineering, 21(6):563–575, 1995.

[SLBK99] T. Schwinn, D. Landes, T. Beil und H. Kempter. *Making know–how transfer work in a purchaser–supplier setting by deploying software inspection.* In: *Proceedings of the $6^{th}$ European Software Quality Conference*, Vienna, Austria, April 1999. Arbeitsgemeinschaft für Datenverarbeitung (ADV). (to appear).

[Sta97] *STATEMATE User Manuals, Version 1.3.* Three Riverside Drive, Andover, Massachusetts 01810, 1997.

[WBM96] D.A. Wheeler, B. Brykczynski und R.N. Meeson Jr. (Herausgeber). *Software Inspection: An Industry Best Practice.* IEEE Press, 1996.

[WR98] C. Wohlin und P. Runeson. *Defect estimation from review data.* In: *Proceedings of the $20^{th}$ International Conference on Software Engineering (ICSE)*, Band 1, Seiten 400–409, Kyoto, Japan, 1998.

# A Rohdaten

| | C–Code | | | |
|---|---|---|---|---|
| | Mikrowelle | Roboter | Motorsteuerung | Kopierer |
| Generell | | | | |
| Enthaltene Mängel (gesamt) | 13 | 17 | 17 | 13 |
| Enthaltene Mängel (testbar) | 10 | 11 | 16 | 12 |
| Enthaltene KS–Mängel | 7 | 6 | 10 | 10 |
| Inspektion | | | | |
| Dauer Vorbereitung [h] | 5,0 / 1,7 / 1,3 | 1,5 / 1,5 /1,5 | 2,5 / 1,5 / 3,8 | 1,5 / 1,0 |
| Dauer Inspektionssitzung [h] | 0,6 | 0,4 | 0,6 | 0,4 |
| Pers. in Inspektionssitzung | 5 | 6 | 5 | 4 |
| Gefundene Mängel (gesamt) | 13 | 13 | 14 | 5 |
| Gefundene Mängel (testbar) | 10 | 8 | 11 | 5 |
| Gefundene Mängel (KS) | 7 | 5 | 7 | 4 |
| Test | | | | |
| Dauer Vorbereitung [h] | 3,5 | 4,0 | 4,8 | 4,5 |
| Dauer Ausführung [h] | 5,3 | 9,3 | 1,1 | 10,5 |
| Dauer Strukturtest [h] | 1,5 | 9,3 | 1,6 | 5,5 |
| Ident. Fehlverhalten (fkt.) | 4 | 6 | 2 | 12 |
| Ident. Fehlverhalten (str.) | 0 | 0 | 1 | 0 |
| Isolierte Mängel (testbar) | 4 | 5 | 3 | 9 |
| Isolierte Mängel (KS) | 2 | 3 | 0 | 2 |

**Tab. A.1:** Rohdaten aus der Prüfung von C–Programmen. Gibt es zum einem Attribut mehrere Werte (z. B. mehrere Inspektoren haben sich auf eine Inspektion vorbereitet), so sind die Werte durch / getrennt.

| | Statemate | | | Matlab/Simulink | | |
|---|---|---|---|---|---|---|
| | Anrufbeantw. | Innenlicht | Lift* | Tempomat* | Fensterst. | Dampfkessel* |
| Generell | | | | | | |
| Enthaltene Mängel (gesamt) | 16 | 8 | 11 | 8 | 7 | 5 |
| Enthaltene Mängel (testbar) | 15 | 8 | 6 | 7 | 7 | 5 |
| Inspektion | | | | | | |
| Dauer Vorbereitung [h] | 2,5 / 1,5 | 1,2 / 1,5 | 4,8 / 1,5 | 1,0 / 1,5 | 1,5 / 1,2 | 3,0 / 1,0 |
| Dauer Inspektionssitzung [h] | 0,5 | 0,5 | 0,75 | 0,25 | 0,4 | 0,4 |
| Pers. in Inspektionssitzung | 4 | 4 | 4 | 4 | 4 | 4 |
| Gefundene Mängel (gesamt) | 6 | 5 | 8 | 5 | 5 | 4 |
| Gefundene Mängel (testbar) | 3 | 5 | 3 | 4 | 5 | 4 |
| Test | | | | | | |
| Dauer Vorbereitung [h] | 2,8 / 2,3 | 1,5 / 3,0 | 1,3 | 1,3 | 1,8 / 1,8 | 1,8 |
| Dauer Ausführung [h] | 11,3 / 2,7 | 1,9 / 3,5 | 2,3 | 1,3 | 2,0 / 2,3 | 2,3 |
| Dauer Strukturtest [h] | 2,0 / 1,0 | 1,0 / 0,0 | 0,0 | 0,0 | 1,8 / 1,5 | 0,0 |
| Ident. Fehlverhalten (fkt.) | 10 / 16 | 6 / 9 | 1 | 3 | 3 / 4 | 1 |
| Ident. Fehlverhalten (str.) | 0 / 0 | 2 / 0 | 0 | 0 | 0 / 0 | 0 |
| Isolierte Fehler (testbar) | 2 / 11 | 5 / 2 | 1 | 1 | 2 / 3 | 1 |
| Ad–Hoc Simulation | | | | | | |
| Aufwand für Simulation [h] | 1,5 / 1,3 | 0,8 / 0,7 | 2,5 / 1,5 | 1,0 / 1,2 | 0,8 / 0,8 | 1,0 / 1,5 |
| Gefundene Mängel (testbar) | 4 / 4 | 6 / 4 | 2 / 3 | 3 / 4 | 3 / 3 | 1 / 1 |

**Tab. A.2:** Rohdaten aus der Prüfung ausführbarer Spezifikationen. Bei den mit * markierten Systemen fehlen krankheitsbedingt die Daten für den zweiten Test.