Einreichung zu ''Informatik -- Forschung und Entwicklung''
(Beitrag von ''Softwaretechnik '98'' in Paderborn)

Methodik und Ergebnisse einer Experimentreihe über Entwurfsmuster

Lutz Prechelt, Barbara Unger (prechelt,unger@ira.uka.de)
Fakultät für Informatik, Universität Karlsruhe
D-76128 Karlsruhe
+49/721/608-4068, Fax: +49/721/608-7343

11. März 1999

Zusammenfassung: Software-Entwurfsmuster sind eine intuitiv einleuchtende Idee, die viele Befürworter hat. Als Forscher dürfen wir die behaupteten Wirkungen von Entwurfsmustern jedoch nicht einfach glauben, sondern müssen sie gründlich prüfen. Dieser Artikel beschreibt die wichtigsten Behauptungen und entwickelt daraus zugehörige Forschungsfragen. Wir diskutieren die Methodik für ein Forschungsprogramm, das die Fragen beantworten soll, und skizzieren praktische Beschränkungen, unter denen diese Forschung ablaufen muß.

Es folgt eine kurze Beschreibung von drei kontrollierten Experimenten und den Hauptfolgerungen aus ihren Ergebnissen. So sollte man zum Beispiel die Benutzung von Entwurfsmustern in einem Entwurf genau dokumentieren und Entwurfsmuster nicht einsetzen, ohne alternative Entwürfe zu prüfen. Abschließend diskutieren wir ein im Rahmen des Forschungsprogramms geplantes viertes Experiment.

Der Beitrag dieses Artikels liegt vorrangig in einer Beschreibung und Diskussion wichtiger methodischer Aspekte kontrollierter Experimente in der Softwaretechnik.

Schlüsselwörter: empirische Methodik, kontrollierte Experimente, Entwurfsmuster

Abstract: Software design patterns are an idea that is intuitively appealing and has found many advocates. However, as scientists we must be concerned about gathering hard evidence for the claims of beneficial consequences of design patterns. This article describes the major claims and derives the corresponding research questions. It discusses the methodology of a research programme for investigating these questions and sketches the practical constraints that make this research difficult.

It then shortly summarizes three controlled experiments that were successfully carried out within these constraints and lists the main results and their consequences, such as: One should document design patterns when they are used and one must not apply design patterns without judgement of alternatives. Finally, design considerations of a fourth experiment are discussed.

The contribution of this paper is a description of important methodological aspects of practical experimental work and how these relate to the results obtained. Understanding these relations will be important in future empirical software engineering research.

Key words: empirical methodology, controlled experiments, design patterns

CR Subject Classification: Experimentation, D2.11, D3.3(Patterns)

1 Das Phänomen „Entwurfsmuster“

Software-Entwurfsmuster sind im wesentlichen eine ingenieurmäßige *Methodik*: Beschreibe eine bewährte Lösung für ein wiederkehrendes Entwurfsproblem und benutze dabei eine standardisierte Form, die den Kontext und die Eigenschaften der Lösung klarstellt und die Wiederbenutzung der Lösung erleichtert.

Diese Idee ist intuitiv einleuchtend und hat sehr schnell die Aufmerksamkeit vieler Praktiker und Forscher auf sich gezogen. Die einschlägige Literatur wächst rapide. Die erste systematische Sammlung von Entwurfsmustern [4] stammt von Gamma, Helm, Johnson und Vlissides (genannt „the Gang of Four“) und ist seit ihrem Erscheinen 1995 ein Riesenerfolg. Andere Bücher folgten, z. B. [2, 11], und eine jährliche Konferenz [10] beschäftigt sich mit der Suche nach neuen Mustern und der Pflege einer einheitlichen Beschreibungsform. Praktiker haben anekdotische Berichte über gute Erfahrungen mit der Anwendung von Entwurfsmustern publiziert und behaupten diverse konkrete Vorteile verglichen mit dem bisherigen Vorgehen [1].

1.1 Nochmal der „Objektorientierungs-Effekt“?

Der heutige Enthusiasmus für Entwurfsmuster erinnert an die Zeit, als die Objektorientierung (OO) populär wurde: Gewaltige Projekterfolge wurden berichtet und viele dachten, OO sei *the silver bullet*, das Allheilmittel für die Probleme der Softwaretechnik. Erst allmählich wurde klar, daß es durchaus auch Schwierigkeiten dabei

gab, die neuen Möglichkeiten richtig anzuwenden. Es gab über lange Zeit mindestens ebenso viele Projekte, die sich mit OO in noch größere Schwierigkeiten brachten als zuvor, wie andere Projekte, denen OO half. Nur ganz allmählich wurden die Regeln und Tricks für den richtigen Gebrauch von OO allgemein verstanden — und dieser Prozeß ist heute noch im Gange.

In der Informatikforschung wurde OO schnell aufgegriffen und als Basis von Sprachen, Methoden und Werkzeugen eifrig benutzt, aber kaum ein Forscher kümmerte sich um Fragen wie *„Ist OO wirklich besser als herkömmlicher strukturierter Entwurf und strukturierte Programmierung? Unter welchen Umständen?“*. Vermutlich hätte eine zügige empirische Erforschung viel schneller den Weg zur breiten richtigen Anwendung von OO in der Praxis geebnet — aber solche Forschung wurde kaum durchgeführt. Diesen Fehler sollten wir bei Entwurfsmustern nicht noch einmal begehen.

1.2 Folgerung

Wie wir sehen, sollte die Softwaretechnik-Forschung die Idee der Entwurfsmuster gründlich empirisch untersuchen. Zum einen können wir dadurch ihre Vorteile quantifizieren und die Anwendungsbedingungen besser verstehen, so daß der praktische Einsatz von Entwurfsmustern verbessert wird. Zum anderen lassen sich durch gezielte Erforschung auch die Mechanismen besser verstehen, durch die Entwurfsmuster ihre positiven Wirkungen erzielen, was zur Verbesserung oder Verallgemeinerung der Entwurfsmusteridee führen dürfte.

Im folgenden Abschnitt diskutieren wir ein Forschungsprogramm für die Validierung von Entwurfsmustern. Wir beschreiben zuerst die Behauptungen und die zugehörigen Forschungsfragen und analysieren dann die dafür geeignete Forschungsmethodik.

Abschnitt 3 beschreibt dann eine noch laufende Reihe kontrollierter Experimente, die wir in diesem Bereich durchführen, und nennt *kurz* die bisher erzielten Resultate.

2 Herleitung eines Forschungsprogramms

Wir skizzieren nun ein Forschungsprogramm, mit dem die Eigenschaften der Entwurfsmusteridee untersucht werden können. Wir reihen zunächst die Behauptungen auf, die über mögliche Vorzüge von Entwurfsmustern gemacht werden. Jede dieser Behauptungen B führt direkt zu zwei Forschungsfragen: Nummer Eins lautet *„Ist B zutreffend?“*. Die Untersuchung dieser Frage führt unweigerlich zu neuen Beobachtungen und Hypothesen. Diese können dann geprüft werden, um Frage Zwei zu beantworten: *„Unter welchen Bedingungen gilt B?“*.

Anschließend diskutieren wir die Methoden, die man zur Untersuchung dieser Fragen anwenden könnte, und beschreiben praktische Beschränkungen, die diese Forschung erschweren. Diese Analyse gilt in ähnlicher Form auch für viele andere Themen der Softwaretechnik.

2.1 Behauptungen über Entwurfsmuster

Die Hauptvorzüge von Entwurfsmustern sind laut der einschlägigen Literatur die folgenden:

Behauptung LE: Entwerfer lernen schnell besser zu entwerfen, wenn sie Entwurfsmuster studieren.

Behauptung PE: Die Benutzung von Entwurfsmustern erhöht die Produktivität von Entwerfern (und evtl. auch von Implementierern).

Behauptung QA: Anfänger können die Qualität ihrer Entwürfe erheblich verbessern, indem sie Entwurfsmuster einsetzen.

Behauptung QE: Auch bei erfahrenen Entwerfern unterstützen Entwurfsmuster die Verwendung bewährter Lösungen und stellen hohe Entwurfsqualität sicher.

Behauptung KE: Entwurfsmuster verbessern die Kommunikation zwischen Entwicklern (sowohl Entwerfern als auch Implementierern).

Behauptung KW: Entwurfsmuster verbessern die Kommunikation von Entwicklern und Wartungspersonal.

Die genannten Formulierungen sind sicher anfechtbar, denn die Behauptungen werden selten so ausdrücklich aufgestellt, aber ihrem Wesen nach entspricht die Liste den Behauptungen, die sich aus der Entwurfsmusterliteratur entnehmen lassen, zum Beispiel aus [4, Kapitel 1, Seite 2] oder [1].

2.2 Methodische Erwägungen und praktische Beschränkungen

Wir analysieren nun wissenschaftliche und praktische Aspekte empirischer Erforschung von Entwurfsmustern: den grundlegenden Forschungsansatz (kontrolliertes Experiment oder Feldstudie), die Typen von Versuchspersonen (Studenten oder Profis[1]), die Arten von Arbeiten, die untersucht werden (Entwicklung oder Wartung, Einzel- oder Teamarbeit, etc.), und die technische Durchführung der Studie.

Die Analyse führt zu Entwurfsentscheidungen bezüglich der konkreten Durchführung der Forschung.

2.2.1 Labor- oder Feldforschung?

Eine fundamentale Unterscheidung empirischer Forschungsarbeiten in der Softwaretechnik ist die folgende: einerseits Studien in einer „sauberen“, kontrollierten Laborumgebung (kontrollierte Experimente, [3]) und andererseits Studien in realen software-

[1] Unter *Softwareprofis* oder *Profis* verstehen wir berufsmäßige Softwareingenieure mit akademischer Informatikausbildung.

technischen Arbeitsumgebungen (Feldstudien). Letztere können direkte Beobachtungen laufender Arbeiten sein oder Postmortem-Analysen („Software-Archäologie“). Die Unterscheidung zwischen Labor und Feld ist nicht scharf, denn auch im Feld kann manchmal eine überwiegend kontrollierte Arbeitsumgebung hergestellt werden, z. B. bei Fallstudien.

Der Hauptvorteil von Feldforschung ist ihre Realitätsnähe: Die Resultate sind in jedem Fall für mindestens eine reale Situation gültig. Andererseits kann dieser Vorteil irreführend sein, denn die hohe Komplexität, der Mangel an Kontrolle über die Bedingungen und die Unmöglichkeit, eine Feldumgebung genau zu charakterisieren, machen es sehr schwierig, die Ergebnisse einer Feldstudie zu verallgemeinern.

Der Hauptvorteil von Laborforschung ist ihre Reproduzierbarkeit: Die Umgebungsbedingungen können genau kontrolliert und dokumentiert werden. Insbesondere kann die extrem wichtige Variable *Leistung des einzelnen Programmierers* durch interne Replikation kontrolliert werden, d. h. durch statistischen Vergleich einer Zufallsauswahl mehrerer Programmierer anstatt nur einzelner. Da nur reproduzierbare Studien wissenschaftlich „harte“ Ergebnisse darstellen, sind kontrollierte Experimente im Prinzip Feldstudien vorzuziehen.

Das Problem bei Laborexperimenten besteht darin, die Nützlichkeit der Ergebnisse sicherzustellen. Bislang fehlen Modelle dafür, wie sich Laborergebnisse auf professionelle Arbeitsumfelder übertragen lassen. Das gleiche Problem besteht jedoch auch für Feldstudien. Deren Resultate sind zwar in einem realen Arbeitsumfeld garantiert gültig, nämlich dort, wo sie ermittelt wurden, aber auch hier fehlen Modelle, die die Übertragung der Ergebnisse auf andere Arbeitsumfelder beschreiben. Wenn es solche Modelle gäbe, wäre die Übertragung von Feldresultaten sogar schwieriger als die von Laborresultaten, weil die Arbeitsumgebung einer Feldstudie wegen ihrer höheren Komplexität schwieriger zu charakterisieren ist.

Der sinnvollste Gesamtansatz für die Forschung ist zumeist, beides einzusetzen: Die Ergebnisse von Laborexperimenten führen zu neuen Ansätzen, die im Feld ausprobiert und untersucht werden können. Umgekehrt führen Beobachtungen im Feld zu neuen Hypothesen, die im Labor geprüft werden sollten — und genau das geschieht gerade mit Entwurfsmustern: Sie wurden von Praktikern im Feld entwickelt und implizieren eine Reihe von Hypothesen, die mit kontrollierten Experimenten überprüft werden können. Um quantitative Modelle über die Wirkung von Entwurfsmustern aufzustellen und zu überprüfen, werden Resultate sowohl aus dem Labor als auch aus dem Feld nötig sein.

Eine letzte Bemerkung: Selten, aber manchmal, ist es möglich, in einer realen Arbeitsumgebung im Feld ein kontrolliertes Experiment auszuführen und damit das beste beider Welten zu erhalten. Ein Beispiel ist die Arbeit von Porter et al. über Teamgrößen bei Inspektionen [5].

2.2.2 Versuchspersonen

Im Mittelpunkt jedes Laborexperiments stehen die Versuchspersonen; hier: Entwerfer oder Programmierer. Diese Versuchspersonen können erfahrene oder wenig erfahrene Softwareprofis sein oder Studenten. Größtmögliche Verallgemeinerbarkeit der Ergebnisse läßt sich nur mit einer breiten Mischung von Profis erreichen, die alle möglichen Softwareprozesse, Anwendungsfelder, Erfahrungsniveaus etc. repräsentieren. Dies ist aber praktisch unmöglich zu erreichen.

Alle anderen Möglichkeiten haben Nachteile: Studenten sind oft vergleichsweise unerfahren, was die Verallgemeinerbarkeit einschränkt. Im Gegenzug sind Profis mit langjähriger Erfahrung sehr auf ihr angestammtes Arbeitsumfeld, ihr Anwendungsfach und ihren gewohnten Softwareprozeß angepaßt und spezialisiert — was ebenfalls die Verallgemeinerbarkeit einschränkt, denn es ist wie erwähnt vollkommen unbekannt, wie man Ergebnisse für eine homogene Gruppe von Profis auf irgendeine andere übertragen kann. Berufsanfänger mit geringer Erfahrung liegen dazwischen; sie bilden einen Kompromiß der Vor- und Nachteile der beiden Gruppen und liegen im Verhalten nah an Studenten kurz vor dem Diplom.

In der Praxis sind Profis nur selten überhaupt für kontrollierte Experimente verfügbar, denn die dabei nötige Replikation führt zu großen Mengen scheinbar unproduktiver Arbeit, von deren Nutzen sich Firmen nur selten überzeugen lassen. Im Endeffekt werden deshalb die allermeisten Experimente mit studentischen Versuchspersonen durchgeführt. Wenn diese hinreichend viel Erfahrung haben, ist das kein Nachteil: Je nach Aufgabe kann es einfacher sein, die Ergebnisse für Studenten auf höhere Erfahrungsniveaus zu übertragen, als die Ergebnisse von einem ganz speziellen professionellen Arbeitsumfeld auf ein anderes umzudeuten.

Unabhängig von der Art der Versuchspersonen spielt ihre Anzahl eine wichtige Rolle bei Experimenten: Je mehr Teilnehmer ein Experiment hat, desto einfacher wird es, die individuellen Verhaltens- und Leistungsschwankungen von denjenigen zu unterscheiden, die durch die untersuchte Variable erzeugt werden; sind umgekehrt zuwenige Teilnehmer vorhanden, sind die Ergebnisse oft statistisch nicht signifikant (d. h. sie könnten auf Zufall beruhen). Dieses Problem wird geringer, wenn die individuelle Leistung der Versuchspersonen sehr ähnlich ist oder wenn man die Leistung jeder einzelnen zumindest schon vorher gut abschätzen kann.

Ein letztes Problem: Die meisten Experimente in der Softwaretechnik verlangen von den Versuchspersonen spezifische Kenntnisse; beispielsweise sind für die meisten Experimente über die Wirksamkeit und Wirkung von Entwurfsmustern Kenntnisse über Entwurfsmuster vonnöten. Wenn also das Experiment eine solche *state-of-the-art*-Technik betrifft, müssen die Versuchspersonen oftmals vor dem Experiment zunächst passend aus-

gebildet werden, was den Aufwand nochmals beträchtlich erhöht.

2.2.3 Aufgabentyp

Die Aufgabe, die den Versuchspersonen gestellt wird, kann sich in verschiedener Hinsicht unterscheiden: Es kann eine Entwurfs- oder eine Implementationsaufgabe sein, beginnend von Null oder mit einem existierenden Programm (Wartung), sie kann alleine oder in Teamarbeit erledigt werden, die Programme können unterschiedliche Größe haben und aus verschiedenen Anwendungsfeldern kommen, und schließlich können verschiedene Sorten von Entwurfsmustern darin vorkommen.

Im Prinzip wären große Teamaufgaben am realistischsten. Es ist jedoch praktisch unmöglich, solche Aufgaben genügend oft zu wiederholen, deshalb sollte man im allgemeinen eher kleine Aufgaben benutzen und die Versuchspersonen einzeln daran arbeiten lassen. Allerdings besteht bei kleinen Programmen die Gefahr, daß deren Gehalt an Entwurfsmustern zu gering wird. Der Ausweg liegt im Bevorzugen von Wartungsaufgaben (z. B. Programmerweiterung) gegenüber Neuentwicklung, denn für Wartungsaufgaben lassen sich größere Programme benutzen, ohne daß die Arbeitsmenge zu sehr ansteigt.

2.2.4 Arbeitsumgebung

Die realistischste Arbeitsumgebung bestünde darin, daß die Versuchspersonen ihren normalen Arbeitsplatz benutzten, also ihren eigenen Schreibtisch im gewohnten Büro, ihre normale Hardware- und Softwareumgebung, ihre Bücher und Unterlagen, die Kollegen zum Befragen etc.

Aber je nach der Frage, die ein Experiment stellt, kann dies unkontrollierbar sein oder sogar widersinnig: Vielleicht dürfen eben gerade *keine* Kollegen befragt oder bestimmte Bücher oder Computerunterstützung nicht benutzt werden; externe Unterbrechungen durch das Telefon oder Kollegen stören den Ablauf und verändern das Ergebnis in unkontrollierter und nicht reproduzierbarer Art und Weise und so fort.

Deshalb würden wir eine „saubere“, wohlausgestattete, relativ ungestörte Laborumgebung bevorzugen, die die wesentlichen Eigenschaften der normalen Arbeitsumgebung nachbildet, aber die Verfälschungen vermeidet. Auch die ist jedoch schwierig bereitzustellen, denn es ist eine kaum lösbare technische Herausforderung, Dutzenden verschiedener Versuchspersonen beispielsweise ihre gewohnte Softwareumgebung zu bieten — von der Hardware gar nicht zu reden.

Praktische Beschränkungen zwingen deshalb in der Regel dazu, entweder eine standardisierte Hardware- und Softwareumgebung zu benutzen oder das Experiment ganz ohne den Computer abzuwickeln, beispielsweise auf Papier mittels Handschrift und Zeichnen. Wie schädlich für das Ergebnis solche Einschränkungen sind, hängt stark von der jeweiligen Arbeitsaufgabe ab. Beispielsweise können Entwurfsaufgaben meist recht gut auch auf Papier gelöst werden, während Implementierungsaufgaben nach einer gewohnten Computerumgebung verlangen.

2.2.5 Fazit

Wie wir aus dieser Analyse sehen können, sind kontrollierte Experimente die richtige Methode für die anfängliche wissenschaftliche Untersuchung der im Feld aufgestellten Hypothesen über Entwurfsmuster. Solche kontrollierten Experimente werden neue Fragen liefern, die zum Teil dann wiederum in Feldstudien untersucht werden müssen. Für diese ersten Experimente sind Studenten kurz vor ihrem Diplom und Profis ähnlich gut als Versuchspersonen geeignet. Praktische Überlegungen lassen Wartungsaufgaben als besonders geeignet erscheinen, da dies die mögliche Programmgröße am wenigsten einschränkt. Die konkreten Aufgaben sollten so ausgewählt werden, daß die Einflüsse anderer technischer Beschränkungen (insbesondere die Computerinfrastruktur) möglichst klein ausfallen.

3 Die Experimentreihe

Wir skizzieren nun konkret Methodik und Ergebnisse für drei bisher durchgeführte Experimente (mit den Nummern 1a, 1b und 2) und ein geplantes viertes Experiment (mit der Nummer 3).

3.1 Experimente 1a und 1b: Musterdokumentation [Wartung]

Experiment 1a und seine abgewandelte Wiederholung 1b vergleichen die Geschwindigkeit und Korrektheit von Wartung für je zwei Paare von Programmen: Ein Programm jedes Programmpaars verwendet Entwurfsmuster und dokumentiert deren Verwendung ausdrücklich. Das andere Programm jedes Paars enthält denselben Code, jedoch ohne die ausdrückliche Beschreibung der Verwendung von Entwurfsmustern. Beide Programme sind auch ansonsten so gründlich kommentiert, daß der Verzicht auf die Entwurfsmusterdokumentation keinen Unterschied machen dürfte, falls ihr nicht eine besondere Qualität zukommt — und genau das will das Experiment prüfen. Rein sachlich gesehen war die Information in der Entwurfsmusterdokumentation weitgehend zu den übrigen Kommentaren redundant. Eine ausführliche Beschreibung des Experiments und dessen Ergebnisse findet sich in [6, 8, 7].

Fragestellung: Ist die Behauptung **KW** (bessere Kommunikation für Wartung) richtig? Unser Experiment vermeidet ein schwieriges Problem beim Prüfen dieser Behauptung: Wenn man Programme, die Muster enthalten, vergliche mit anderen Programmen ohne Muster, wäre kaum zu unterscheiden, welcher Teil der Ergebnisse von Kommunikationsverbesserungen und welcher einfach von sonstigen Programmunterschieden

Variable	Mittel mit PD PD^+	Mittel ohne PD PD^-	Mitteldifferenz (90% Konfidenz) I	Signifikanz p
Gesamtpunkte	20.8	21.1	$-6.0\% \ldots +3.3\%$	0.35
relevante Pkte.	16.1	16.3	$-8.0\% \ldots +4.0\%$	0.35
Zahl korrekter Lösungen	17 von 36	15 von 38		
Zeit (Minuten)	51.5	57.9	$-22\% \ldots +0.3\%$	**0.055**

Tabelle 1: Ergebnisse für die *Beobachter*-Aufgabe von Experiment 1a. Die Spalten bedeuten (von links nach rechts): Name der Variablen, arithmetischer Mittelwert PD^+ der Teilnehmergruppe mit Entwurfsmusterdokumentation (pattern documentation, PD), dito ohne Entwurfsmusterdokumentation, 90%–Konfidenzinterval I für die Differenz $PD^+ - PD^-$ der Variablen, Signifikanz p dieser Differenz (einseitig). p ist die Wahrscheinlichkeit dafür, daß die beobachtete Differenz auf reinem Zufall beruht. Die „Punkte“ bewerten die Korrektheit einer Lösung; „relevante Punkte“ sind die Punkte nur für die musterrelevanten Teilaufgaben. Diese Tabelle soll keine komplette Beschreibung der Ergebnisse darstellen, sondern dient zur Illustration unserer Datenmessung und -analyse.

herrührt. Stattdessen benutzt unser Programm exakt dasselbe Programm in beiden Experimentgruppen, nur die Dokumentation ist verschieden und auch dies nur im Hinblick auf die Entwurfsmuster.

Versuchspersonen: Für Experiment 1a gaben wir an der Universität Karlsruhe einen sechswöchigen Intensivkurs mit praktischen Übungen (Vorlesung plus Praktikum) über Java und Entwurfsmuster für Informatikstudenten nach dem Vordiplom. Die besten 58 von ca. 100 Kursteilnehmern nahmen am Experiment teil. Im Kurs und in den Übungen lehrten wir eine kleine Anzahl von Entwurfsmustern als Beispiele für guten Entwurfsstil in Java. Experiment 1b ist gleich aufgebaut und benutzte ähnliche Programme, jedoch in C++. Es wurde an der Washington-Universität in St. Louis mit 22 Studenten eines *undergraduate*-Kurses über Entwurfsmuster durchgeführt.

Vorgehen: Experiment 1a wurde auf Papier durchgeführt, die Teilnehmer von 1b arbeiteten am Rechner. Jeder Teilnehmer arbeitete an zwei verschiedenen Programmen (siehe unten) und lieferte Lösungen für Wartungsaufgaben. Die Teilnehmer wurden in einen gegenbalancierten Experimententwurf mit vier Gruppen arrangiert.

Programme und Aufgaben: Das Programm *Telefonbuch* realisiert ein einfaches Adreßbuch. Es benutzt das *Beobachter*-Muster; zwei Beobachterklassen mit unterschiedlicher Funktionalität waren bereits implementiert. Die Aufgabe für die Teilnehmer bestand aus mehreren Teilen:
(1) Zufügen von Beobachtern, d. h. Verstehen der spezifischen Beobachterkonstruktion im Programm und Definieren neuer Beobachterklassen, wobei die Unterschiede in der Funktionalität zwischen den Beobachtern erheblich waren. Dieser Teil der Aufgabe betrifft das Beobachtermuster direkt und ein schnelleres Erkennen bzw. Verstehen des Musters wird einen Unterschied in den Daten hervorrufen.
(2) Ändern der Darstellung der Einträge, d. h. Verstehen der Zeichenkettenverarbeitung. Dieser Aufgabenteil war unabhängig vom Verständnis des verwendeten Entwurfsmusters und müßte also so in allen Gruppen gleich gut gelöst werden.

Programm *Und/Oder-Baum* realisiert Bäume, deren Knoten entweder Strings sind (*Blatt*) oder aber Stringverkettung (*Und*) oder Stringalternation (*Oder*) ihrer Unterbäume realisieren. Ein *Kompositum*- und ein *Besucher*-Muster im Programm ermöglichen Berechnungen auf einem solchen Baum; eine Besucherklasse war vorgegeben, die die Tiefe des Baums berechnet. Die Aufgaben bestanden auch hier wieder aus mehreren Teilen, die teilweise musterrelevant und teilweise nicht musterrelevant waren, d. h. unabhängig vom Verstehen des Musters gelöst werden konnten. Die Aufgaben bestanden im Ändern der Darstellungsform für Stringverkettungen, im Berechnen der Gesamtanzahl von Stringverkettungen des Baums mit einer schon gegebenen, aber ineffizienten Prozedur und im Hinzufügen eines neuen Besuchers, der die Gesamtanzahl von Stringverkettungen des Baums (Kreuzprodukt der *Oder*-Knoten) effizient ausrechnen sollte. Unter Verletzung der Entwurfsidee konnte man als Lösung statt eines Besuchers auch je eine Methode zu jeder Baumknotenklasse zufügen.

Ergebnisse: Für das Programm mit *Beobachter*-Muster war in Experiment 1a die Gruppe mit Musterdokumentation erheblich schneller (11%) als die Kontrollgruppe. Signifikante Qualitätsunterschiede wurden nicht gefunden, weder bei der Gesamtqualität (Zeile „Gesamtpunkte“ in Tabelle 1) noch bei den musterrelevanten Teilaufgaben (Zeile „relevante Punkte“ in Tabelle 1). Wir ignorieren hier Experiment 1b, weil sich herausstellte, daß die Aufgabe dort Kenntnisse voraussetzte, die die Teilnehmer nicht hatten.

Für das andere Programm mit dem *Kompositum*- und dem *Besucher*-Muster hatte die Gruppe mit Musterdokumentation entweder weitaus weniger Fehler (Experiment 1a) oder sie benötigte wiederum weniger Zeit (Experiment 1b) als die Vergleichsgruppe. Die Unterschiede in der Wirkweise der Muster, bessere Zeit contra weniger Fehler, kommen durch den unterschiedlichen Experimentaufbau der beiden Experimente zustande: In Experiment 1a wurde nur auf Papier gearbeitet und so wurden Fehler wegen fehlender Implementierung nicht als Fehler erkannt und verbessert im Gegensatz zu den Experimentteilnehmern aus 1b, die

die Möglichkeit des Testens hatten.

Alle hier genannten Unterschiede sind statistisch signifikant.

Diese Ergebnisse stützen die Behauptung **KW**. Die verbesserte Kommunikation durch Entwurfsmusterdokumentation kann sich je nach Situation als Produktivitäts- und/oder als Qualitätszuwachs zeigen.

Zu dieser Fragestellung bieten sich weitere Experimente an, so zum Beispiel wie sich unterschiedliche Arten der Dokumentation auf die Produktivität auswirken. Gibt es Unterschiede, ob die Muster in dem Quelltext dokumentiert sind, in der Systemdokumentation oder in Objektdiagrammen? Reichen kurze Annotationen oder sind ausführlichere Erklärungen sinnvoller?

3.2 Experiment 2: Muster versus alternative Entwürfe [Wartung]

Experiment 2 untersucht, ob die Verwendung von Entwurfsmustern *überhaupt* nutzbringend ist und ob der Nutzen von den Entwurfsmusterkenntnissen abhängt. Die genaue Beschreibung des Experiments steht in [9].

Fragestellung: Ist die Benutzung von Entwurfsmustern für die Programmwartung vorteilhaft, verglichen mit einfacheren Alternativentwürfen für das gleiche Programm? Diese Frage prüft Anteile der Behauptungen **QE** (bessere Qualität der Entwürfe erfahrener Entwerfer) und **KW** (bessere Kommunikation zwischen Entwerfern und Wartern). Wir untersuchen die Fragestellung mit den gleichen Versuchspersonen vor und nach einem Kursus über Entwurfsmuster, so daß auch einige Aspekte von Behauptung **LE** (schnelleres Lernen von Entwerfen) berührt werden.

Versuchspersonen: Wir hielten einen Entwurfsmusterkurs (zwei halbe Tage) für 29 Softwareingenieure der Firma sd&m in München. Das Experiment wurde in zwei Teilen vor und nach dem Kurs abgehalten. Die Versuchspersonen waren C++-Programmierer mit durchschnittlich 4,1 Jahren Berufserfahrung. Die meisten hatten wenig vorherige Kenntnisse über Entwurfsmuster, die Hälfte der Versuchspersonen gar keine.

Vorgehen: Es gab insgesamt vier Programmpaare und zu jedem davon zwei oder drei Wartungsaufgaben. Jede Versuchspersonen arbeitete vor dem Kurs an zwei Programmen (Vortest) und nach dem Kurs an den restlichen beiden (Nachtest). Jedes Programmpaar bestand aus zwei Fassungen eines funktionsgleichen Programms: Eine Fassung benutzte Entwurfsmuster, die andere eine alternative funktionsgleiche Lösung. Dies führt zu dem methodischen Problem eines fairen Vergleichs: Was ist der „richtige“ alternative Entwurf, mit dem man vergleichen sollte? Unsere Antwort: Ein Teil der Flexibilität, die das Entwurfsmuster liefert, wurde im jeweiligen Programm gar nicht benötigt, und ebendiesen sparte der alternative Entwurf ein; er war insofern also einfacher als die Lösung mit dem Entwurfsmuster, aber ebenfalls ein sinnvoller Entwurf für das gegebene Problem. Die Antworten wurden auf Papier angegeben.

Programme und Aufgaben: Programm *Börsenticker* enthielt ein *Beobachter*-Muster für mehrfache, unterschiedliche, dynamische Sichten auf einen Strom von Börsenhandelsereignissen. Die alternative Lösung war ähnlich, hatte aber keinen Registrierungsmechanismus; die Benachrichtigung der Sichten war im Programm hartkodiert. Die Aufgaben waren: einen weiteren, schon implementierten Beobachter anzuzeigen und das dynamische Hinzufügen neuer Anzeiger zu ermöglichen. Die zweite Teilaufgabe durchbrach also die sonstige Regel, daß der vereinfachte Entwurf nur *unnötige* Flexibilität des musterbasierten Entwurfs weglies, denn die Aufgabe ist bei einem komplett umgestetzten Beobachtermuster bereits geloest. Sie diente dadurch als Plausibilitätskontrolle und brachte das erwartete Ergebnis, daß die Alternativgruppe stark schlechter abschnitt.

Programm *Kommunikationskanäle* enthielt ein *Dekorierer*-Muster zum transparenten Zufügen von Verschlüsselung, Kompression etc. zu einem Kommunikationsverbindungsobjekt. In der alternativen Lösung war alle Funktionalität in einer einzigen Klasse realisiert und konnte durch Parameter angeschaltet werden. Als Wartungsaufgaben erhielten die Teilnehmer: Hinzufügen neuer Übertragungsfunktionalität (der Bitsicherungschicht), Herausfinden, wann ein gewisser Programmzustand auftreten kann, und die Formulierung eines Konstruktors zum Erzeugen eines Kommunikationskanals.

Programm *Graphikbibliothek* enthielt ein *Kompositum*-Muster zur transparenten Gruppierung graphischer Objekte und ein *Abstrakte-Fabrik*-Muster zur transparenten Handhabung unterschiedlicher Typen von Ausgabegeräten. Die alternative Version war ähnlich bis auf moderate strukturelle Vereinfachungen. Hier mußten die Versuchspersonen als Aufgabe die Unterstützung eines neuen Ausgabegerätes skizzieren und eine Verständnisfrage beantworten.

Programm *Boolesche Formeln* enthielt ein *Kompositum*- und ein *Besucher*-Muster zur Darstellung und formatierten Ausgabe von booleschen Formeln bestehend aus And-, Or-, Xor- und Not-Termen und Variablen. In der alternativen Version wurde die Besucherklasse ersetzt durch lauter einzelne Methoden in den Termklassen; das Besuchermuster verschwand somit ganz. Als Aufgabe mußten die Versuchspersonen das Programm um das Auswerten von Formeln erweitern. Die zweite Arbeitsaufgabe, das Ändern eines Ausdrucks, wurde von vielen Versuchspersonen falsch interpretiert und daher nicht von uns ausgewertet.

Ergebnisse: Im Prinzip kann die Benutzung eines Entwurfsmusters anstelle einer einfacheren Alternativlösung nützlich (mehr Flexibilität bei gleichen oder sogar geringeren Kosten) oder schädlich sein. In diesem Experiment traten beide Fälle auf:

Beim Programm *Börsenticker* (*Beobachter*) benötig-

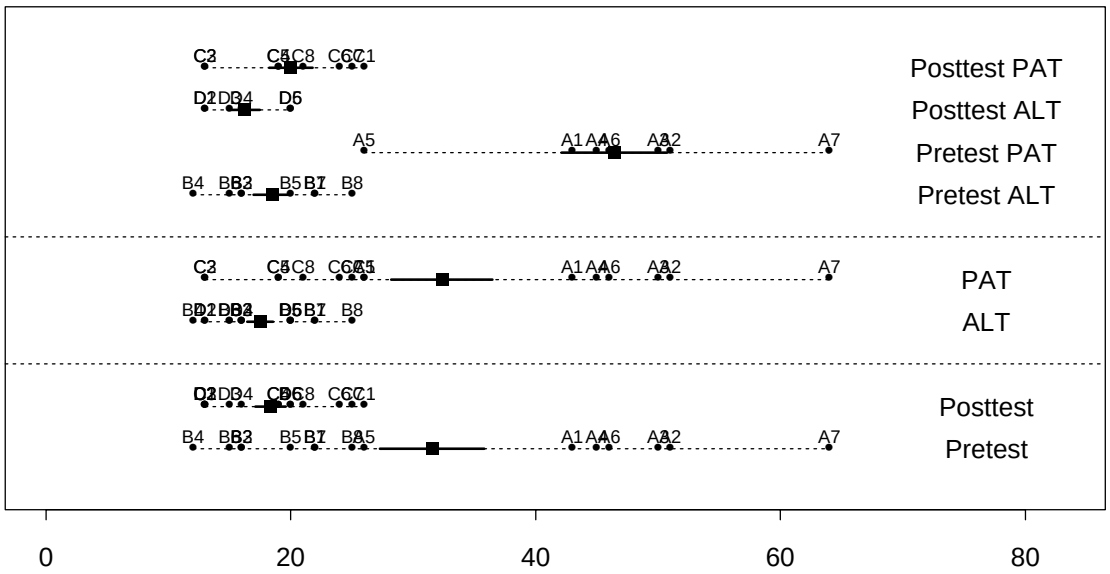

Abbildung 1: Benötigte Zeit in Minuten für Teilaufgabe 1 von Programm *Börsenticker* in Experiment 2, separat für die Fassung mit Entwurfsmustern („PAT“) und die Alternativfassung ohne („ALT“). Jeder Punkt steht für eine Versuchsperson, das Quadrat markiert den Mittelwert, die durchgezogene Linie rechts und links davon gibt plus/minus einen Standardfehler des Mittelwerts an. Wenn zwei solche Linien sich überlappen ist der Unterschied der Mittelwerte statistisch nicht signifikant. Der obere Bereich (vier Linien) zeigt die vier einzelnen Gruppen. Der mittlere Bereich (zwei Linien) zeigt die gleichen Daten, wenn man die Vortest- und Nachtest-Gruppen von PAT vereint bzw. die Vortest- und Nachtest-Gruppen von ALT vereint. Entsprechend zeigt der untere Bereich (zwei Linien) die Ergebnisse des Vortests bzw. des Nachtests, wenn man die PAT- und ALT-Gruppen jeweils vereint.

te die Wartung der Programmfassung mit Muster mehr Zeit, insbesondere im Vortest, als die Musterkenntnisse noch gering waren. Siehe auch Abbildung 1 für die 1. Teilaufgabe, in der die einzelnen Wartungszeiten der Experimentteilnehmer nach Gruppen sortiert aufgetragen wurden. Hier sieht man deutlich, daß die Mustergruppe im Vortest Schwierigkeiten mit dem Entwurfsmuster hatte (Durchschnitt, markiert durch schwarzes Quadrat: 46 Minuten) und im Nachtest gerade mal annähernd so gut war wie die Gruppe mit dem alternativen Entwurf (Durchschnitt Gruppe mit Mustern 20 Minuten vs. 16 Minuten Gruppe ohne Muster).

Bei den Programmen *Graphikbibliothek* (*Kompositum* und *Abstrakte Fabrik*) sowie *Boolesche Formeln* (*Kompositum* und *Besucher*) gab es praktisch keine Unterschiede im Wartungsaufwand zwischen den jeweiligen beiden Fassungen.

Beim Programm *Kommunikationskanäle* (*Dekorierer*) war die Fassung mit Entwurfsmuster klar überlegen, sogar im Vortest bei geringen Entwurfsmusterkenntnissen. Siehe auch Abbildung 2: Für Teilaufgabe 1 wurden in der Mustervariante durchschnittlich 29 Minuten zur Erledigung der Wartungsaufgabe benötigt, während in der alternativen Variante 46 Minuten benötigt wurden. Dieser Trend ist in Vor- und Nachtest gleich.

Bei Betrachtung der Ergebnisse ist interessant, daß es nur wenige Fälle gab, in denen die Mustervariante der alternativen Variante unterlegen war. Weiterhin interessant ist, daß die meisten der Resultate sich korrekt vorhersagen ließen, wenn man die Lokalität der Wechselwirkungen im jeweiligen Programm betrachtete oder auch die Lokalität der erwarteten Änderungen. Wir konnten aus dem Experiment folgende Schlüsse ziehen:

1. Im allgemeinen ist softwaretechnischer gesunder Menschenverstand, wenn er sorgfältig eingesetzt wird, ein gutes Orakel für die Nützlichkeit von Entwurfsmustern im Vergleich zu alternativen Lösungen.

2. Entwurfsmuster sind oft nützlich, indem sie mehr Flexibilität liefern, ohne dabei einfache Wartungsfälle zu verteuern.

3. Es gibt aber auch Fälle, in denen Entwurfsmuster die Wartung erschweren.

Punkt 1 legt zusammen mit der Flexibilität der *Gang-of-four*-Entwurfsmuster nahe, Entwurfsmuster stets zu benutzen, wenn nicht sich konkrete Gründe angeben lassen, warum ein Alternativentwurf besser wäre. Als Fernziel benötigen wir ein statistisches Modell, das den Beitrag diverser Variablen (wie z. B. Art der Berufserfahrung) zu diesen Ergebnissen beschreibt. Damit ließen sich in der softwaretechnischen Praxis solche Entwurfsentscheidungen auf objektiver Basis treffen, anstatt auf Intuition angewiesen zu sein.

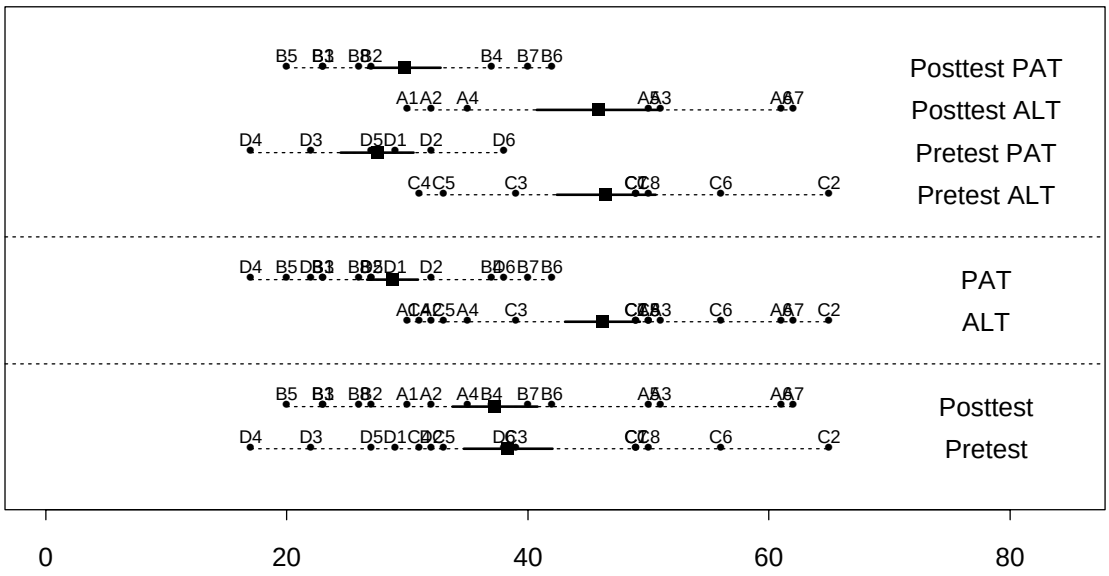

Abbildung 2: Benötigte Zeit für Teilaufgabe 1 bei Programm *Kommunikationskanäle* von Experiment 2.

3.3 Experiment 3: Kommunikation zwischen Entwerfern [Entwurfsphase]

Im Rahmen des hier vorgestellten Forschungsprogramms planen wir gegenwärtig ein viertes Experiment. In der Mustergemeinde wird die bessere Kommunikation in der Entwurfsphase als eines der stärksten Argumente für den Einsatz von Entwurfsmustern angegeben. Die anekdotischen Berichte von Praktikern schreiben Mustern gerade bei dieser wichtigen Tätigkeit besonders positive Wirkung zu. Darum wollen wir in dem geplanten Experiment direkt die Nützlichkeit von Entwurfsmustern für die Verbesserung der Kommunikation während der Entwurfsphase untersuchen (Behauptung **KE**).

Ein Experiment, das die verbesserte Kommunikation testet, kann man wie folgt aufbauen:

Um explizite Kommunikation zu erreichen, kann man Aufgaben an ein Team anstatt an Einzelpersonen stellen. Um den Grad an Kommunikation zu erhöhen, kennt nur eine Person aus dem Team einen gegebenen Entwurf E einschließlich der dahinterstehenden Entwurfsüberlegungen und muß ihn zuerst den anderen Teammitgliedern erklären (Behauptung **KW**), bevor von dem Team eine gemeinsame Entwurfsaufgabe A an diesem gegebenen Entwurf gelöst werden muß.

Die unabhängigen Variablen in diesem Experiment sind: (1) Musterkenntnisse: die Teammitglieder können Kenntnisse über Entwurfsmuster besitzen ($K = wahr$) oder nicht ($K = falsch$). Alternativ könnten auch innerhalb des Teams unterschiedliche Kenntnisstände herrschen. (2) Muster im Programm: Der ursprüngliche Entwurf E kann Entwurfsmuster benutzen oder auch nicht. (3) Muster in der Aufgabe: Wenn E Muster enthält, kann die Erweiterungsaufgabe A diese betreffen oder auch nicht und die Entwurfs- und Aufgabenbeschreibungen können dies verschleiern oder auch nicht.

Die Kommunikation wird auf Video aufgezeichnet, transkribiert und analysiert, um typische Kommunikationsabfolgen zu entdecken, Mißverständnisse und ihre Ursachen zu finden etc. Wir erwarten, daß man die Anteile, die die Behauptungen **KE** und **KW** auf unsere Beobachtungen haben, auseinanderhalten kann und die Experimentergebnisse damit noch nützlicher werden. Im Gegensatz zu den anderen hier beschriebenen Experimenten verfolgen wir in diesem Experiment einen Entwicklungsprozeß und nicht nur das Endresultat. Die Experimente 1 und 2 schließen aus Resultat und Zeitverbrauch auf die Wirkung der zu untersuchenden Variablen zurück. Aus den Beobachtungen von Experiment 3 dürften sich hingegen direkt konkrete Ratschläge für das Vorgehen bei Entwurfssitzungen ergeben.

Zu beobachtende Variablen sind zum Beispiel (1) die Häufigkeit und der Grund von Mißverständnissen oder Kommunikationszusammenbrüchen, (2) das Vorgehen zu deren Auflösung, (3) die Häufigkeit und der Kontext der Benutzung verschiedener Kommunikationsstile, Medien, technischer Ausdrücke, Konzepte aus dem fraglichen Programm etc. und (4) der jeweilige Zweck dieser Benutzungen: entweder das Starten einer Kommunikationsepisode oder die Fortsetzung oder Wiederaufnahme einer früheren. Im Gegensatz zu den Experimenten 1a, 1b und 2 sind hier viele abhängige Variable von qualitativer Art oder stehen in Zusammenhang mit einem Kontext qualitativer Art.

So zeigt Experiment 3 eine deutlich andere Facette der in Kapitel 2 beschriebenen Methodik.

4 Fazit

Um nicht noch einmal den gleichen Fehler wie bei der Objektorientierung zu machen, sollte die Forschungs-

gemeinde der Softwaretechnik die Behauptungen über die Wirkung von Entwurfsmustern gründlich prüfen. Leider ist dies in der Praxis ziemlich schwierig.

Die Kunst des empirischen Arbeitens in der Softwaretechnik besteht darin, nützliche wissenschaftliche Einsichten *trotz* der engen praktischen Randbedingungen zu produzieren, in denen diese Arbeit stattfindet. Bei kontrollierten Experimenten bedeutet dies, die Wesensmerkmale (Charakteristik der Aufgabe) einer softwaretechnischen Situation zu erhalten, obwohl alle quantitativen Merkmale (Größe der Aufgabe) schwindelerregend weit herunterskaliert werden.

Erfolgreiches Herunterskalieren lieferte die ersten reproduzierbaren wissenschaftlichen Resultate über die Wirkungen von Entwurfsmustern. Solche Forschung hat ein günstiges Preis-Leistungs-Verhältnis, denn aus den Ergebnissen können direkte praktische Ratschläge abgeleitet werden: Die Experimente 1a und 1b zeigen, daß es sich empfiehlt, die Benutzung von Entwurfsmustern in Software sorgfältig zu dokumentieren, weil sich dieser geringe Zusatzaufwand bei der Wartung deutlich positiv bemerkbar macht. Experiment 2 zeigt erstens, daß Entwurfsmuster selbst dann nützlich sein können, wenn eine einfachere alternative Lösung möglich wäre, daß aber zugleich, zweitens, eine ungeeignete Anwendung von Mustern in anderen Situationen schädlich sein kann. Der praktische Ratschlag lautet, den gesunden Menschenverstand immer mitzubenutzen, anstatt Entwurfsmuster nach Kochbuchmanier einzusetzen — in Zweifelsfällen sollte man sie allerdings aufgrund ihrer Flexibilität eher bevorzugen.

Viele weitere Experimente und andere Studien werden nötig sein, bis wir ein gutes Verständnis der Wirkung von Entwurfsmustern erreicht haben.

Weitergehende Informationen

Detaillierte Information über unsere Experimente kann unter http://wwwipd.ira.uka.de/EIR/ abgerufen werden. Dort befinden sich insbesondere alle Experimentmaterialien und die rohen Resultatdaten.

Danksagungen

An der Planung und Durchführung haben Michael Philippsen (Experiment 1a), Douglas Schmidt (Experiment 1b) und Walter Tichy (Experiment 2) mitgewirkt. Ernst Denert initiierte und Peter Brössler organisierte Experiment 2 bei der Firma sd&m. Vor allem jedoch danken wir unseren Versuchspersonen.

Literatur

1. Beck, K., Coplien, J.O., Crocker, R., Dominick, L., Meszaros, G., Paulisch, F., Vlissides, J.: Industrial experience with design patterns. In: *18th Intl. Conf. on Software Engineering*, pages 103–114, Berlin, March 1996. IEEE CS press

2. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: *Pattern-Oriented Software Architecture — A System of Patterns.* Chichester, UK: John Wiley and Sons 1996

3. Christensen, L.B.: *Experimental Methodology.* Needham Heights, MA: Allyn and Bacon 1994

4. Gamma, E., Helm, R., Johnson, R., Vlissides, R.: *Design Patterns: Elements of Reusable Object-Oriented Software.* Reading, MA: Addison-Wesley 1995

5. Porter, A.A., Siy, H., Toman, C.A., Votta, L.G.: An experiment to assess the cost-benefits of code inspections in large scale software development. In: *Proc. Third ACM Sigsoft Symposium on the Foundations of Software Engineering* 1995

6. Prechelt, L.: An experiment on the usefulness of design patterns: Detailed description and evaluation. Universität Karlsruhe, Germany: Technical Report 9/1997, Fakultät für Informatik June 1997 ftp.ira.uka.de

7. Prechelt, L., Unger, B., Philippsen, M., Tichy, W.F.: Two controlled experiments assessing the usefulness of design pattern information during program maintenance. *Empirical Software Engineering* 1999. Submitted. http://wwwipd.ira.uka.de/~prechelt/Biblio/

8. Prechelt, L., Unger, B., Schmidt, D.: Replication of the first controlled experiment on the usefulness of design patterns: Detailed description and evaluation. Washington University, St. Louis: Technical Report wucs-97-34, Dept. of CS December 1997

9. Prechelt, L., Unger, B., Tichy, W.F., Br össler, P., Votta, L.G.: A controlled experiment in maintenance comparing design patterns to simpler solutions. *IEEE Trans. on Software Engineering*, 1999. Submitted. http://wwwipd.ira.uka.de/~prechelt/Biblio/

10. Schmidt, D.: Collected papers from the PLoP '96 and EuroPLoP '96 conferences. Washington University, St. Louis: Technical Report wucs-97-07, Dept. of CS February 1997 (Conference "Pattern languages of programs").

11. Shaw, M., Garlan, D.: *Software Architecture — Perspectives on an Emerging Discipline.* Prentice Hall 1996