Zwei kontrollierte Experimente über die Nützlichkeit der Dokumentation von Entwurfsmustern bei der Programmwartung

Henning Staib

30. März 2004

Inhaltsverzeichnis

1 Dokumentation von Entwurfsmustern

1.1 Entwurfsmuster

Entwurfsmuster stellen eine Menge bewährter Konzepte und Ideen dar, die es ermöglichen, gut strukturierte und wiederverwendbare Software zu entwickeln. Durch die Kenntnis und den Einsatz von Entwurfsmustern lässt sich die Programmqualität sowie Programmierproduktivität des Entwicklers verbessern. Außerdem ermöglichen Entwurfsmuster eine bessere Kommunikation zwischen den Entwicklern einer Software untereinander und den Programmierern die später Wartungsarbeiten an dieser Software durchführen, da sie mithilfe der Begriffswelt der Entwurfsmuster effizienter miteindander kommunizieren können und das Programm schneller verstehen können. Zu diesem Aspekt erschien im Juni 2002 ein Aufsatz über die erste formale, empirische Untersuchung zweier kontrollierter Experimente zur Nützlichkeit der Dokumentation von Entwurfsmustern bei der Programmwartung.

1.2 Dokumentation

Dokumentation von Entwurfsmustern sieht man heute schon häufig bei Namenskonventionen von Klassen und Methoden, z.B. Listener, Event. In den Experimenten wurde die Dokumentation der Entwurfsmuster durch Hinzufügen zusätzlicher „Entwurfsmuster-Kommentar-Zeilen“ (Pattern Comment Lines), im folgenden kurz PCL genannt, realisiert, d.h. neben der normalen Kommentierung des Programms wurden Kommentare eingefügt, die die jeweils verwendeten Entwurfsmuster angeben sowie u.U. mit welchen weiteren Komponenten zusammengearbeitet wird.

2 Experiment

2.1 Motivation

Die folgende Untersuchung wurde angestellt um einen Aspekt der Nützlichkeit von Entwurfsmustern zu beweisen, die zwar immer unterstellt, aber nicht wissenschaftlich bewiesen ist. Es gab zwar bereits Untersuchungen zum Verstehen von Programmen, allerdings nicht speziell auf Entwurfsmuster bezogen.

Ein Problem bei der Wartung von Software ist gerade das Verstehen des Programms. Dies geschieht in der Regel so, dass sich die einzelnen Teile des Programms angeschaut werden und dann, nachdem ein Verständnis der Funktionalität dieser Einzelteile erlangt wurden ist, aus den Teilen auf das Ganze geschlossen wird, d.h. wie die Einzeitele im Programm miteinander arbeiten. Dieser Prozess des „bottom-up“-Verstehens ist sehr schwierig und zeitaufwendig. Besser wäre es zunächst einen groben Überblick auf die Gesamtheit des Programms

zu haben und so zielstrebiger und effizienter die Stellen im Programm zu finden, die für die Wartung interessant sind. Hier setzt nun die Dokumentation von Entwurfsmustern an, die dem Programmierer eine „top-down“-Ansicht des Programms vermittelt. Diese „top-down“-Ansicht ist insbesondere bei objektorientierten Programmen wichtig, da gerade dort Komponenten, die miteinander zusammenarbeiten, sehr verteilt sind (verschiedene Klassen an verschiedenen Orten). Untersucht wurden die beiden Fragen:

- Macht PCL die Wartungsarbeiten schneller?
- Werden mit PCL weniger Fehler gemacht?

2.2 Beschreibung

Für diese, von Lutz Prechelt, Barbera Unger-Lamprecht, Michael Philippsen und Walter F. Tichy angestellte Untersuchung wurden zwei ähnliche Experimente durchgeführt. Das erste fand im Januar 1997 an der Universität Karlsruhe (UKA) statt. Es nahmen 74 Versuchspersonen, von denen 64 Akademiker und 10 Studenten der Informatik waren, teil. Alle Versuchspersonen hatten zuvor einen 6-wöchigen Intensivkurs über Java und Entwurfsmuster, der direkt auf das Experiment ausgerichtet war, absolviert.

Das zweite Experiment wurde im Mai 1997 an der Washington University of St. Louis (WUSTL) in St. Louis, Missouri, USA statt. Hier standen nur 22 Studenten der Informatik, die zuvor einen 12-wöchigen Kurs über C++ und Entwurfsmuster belegt hatten, als Versuchspersonen zur Verfügung. Im Gegensatz zu den Teilnehmern der UKA hatten die WUSTL Studenten keine praktische Erfahrung mit Entwurfsmustern.

Tabelle mit Teilnehmnerinfos:

	UKA	WUSTL
Programmiererfahrung	7,5 Jahre	5 Jahre
Programmiersprachen	4,6	4,0
größtes Programm	3.510 LOC	2.557 LOC
Erfahrung mit OO-Programmierung	69%	76%
Erfahrung mit GUI-Programmierung	58%	50%

2.2.1 Experimententwurf

Die Experimente sind als Intra-Subjekt Entwurf mit Gegenbalancierung konzipiert, d.h. jede Versuchsperson hat mehrere Aufgaben zu lösen, so dass mehr Messungen möglich sind als wenn jeder nur eine Aufgabe bearbeitet. Gegenbalancierung bedeuted, dass zu jeder Aufgabe jeweils zwei Gruppen existieren, eine Versuchsgruppe bei der der Quellcode zusätzlich mit PCL versehen ist und eine Kontrollgruppe, bei der dies nicht der Fall ist. Die unabhängige Variable in diesen Experimenten ist also das Vorhandensein bzw. das Nichtvorhandensein von PCL. Die beobachteten abhängigen Variablen sind die Zeit, die für das Lösen der aufgaben benötigt wird sowie die Korrektheit der Lösungen.

2.2.2 Aufgabe

Im Rahmen des Experimentes musste jeder Teilnehmer zwei, bereits gründlich kommentierte, Programme (And/Or-Tree, Phonebook) bearbeiteten, wobei jeweils nur eins von beiden PCL enthielt. Damit die Reihenfolge, in der die Programme bearbeitet wurden (zuerst das mit PCL oder das ohne), keinen Einfluss auf die Messergebnisse hat, wurden alle Versuchspersonen zufällig in 4 Gruppen eingeteilt:

Grösse der Gruppen (UKA / WUSTL):

	erst mit PCL, dann ohne PCL	erst ohne PCL, dann mit PCL
erst And/Or-Tree, dann Phonebook	19 / 6	18 / 5
erst Phonebook, dann And/Or-Tree	18 / 6	19 / 5

Allerdings gab es bei den Randbedingungen zwischen den Experimenten in UKA und in WUSTL einige Unterschiede. So mussten die Versuchspersonen bei UKA ihre Lösungen auf Papier ausarbeiten, während WUSTL UNIX-Workstations zur verfügung stellte auf denen die Lösungen implementiert und getestet werden konnten.

And/Or-Tree And/Or-Tree ist eine Anwendung zur Handhabung von Und/Oder Bäumen von Strings (Beispiel ... TODO!!!). In diesem Programm kommen die Entwurfmuster Kompositum und Besucher vor.

Statistik:

	UKA	WUSTL
LOC	362	498
Klassen	7	6
Kommentar	133	178
PCL	18	22

Es sollten nun vier Teilaufgaben zu diesem Programm gelöst werden:

1. Finden der richtigen Stelle um Veränderungen der Ausgabe zu bewerkstelligen
2. Entwickeln einer Formel um die Anzahl der Sätze aus einem gegebenen Syntaxbaum zu berechnen
3. Schreiben einer Besucherklasse, die die Anzahl aus 2. schneller berechen kann
4. Einbauen der Klasse aus 3. und Ergebnis ausfgeben

Im gegeben Quelltext des Programmes befand sich bereits eine Besucherklasse, die die Tiefe des Syntaxbaumes berechnen könnte, so dass sich Aufgabe 3 durch modufizieren dieser Klasse sehr einfach lösen ließ.

Nur die Aufgaben 3 und 4 waren entwurfsmusterrelevant und nur die dort erzielten Punkte fließen am Ende in die Auswertung des Experiments ein.

Phonebook Dieses Programm stellt Eingaben des Benutzers (Name, Vorname, Telefonnummer) in verschiedenen Ansichten dar. Während bei UKA diese Ausgabe mittels einer GUI erfolgt, wird bei der Version von WUSTL nur der I/O Stream zur Ausgabe aller Ansichten genutzt. Die hier verwendeten Entwurfsmuster sind Beobachter und Schablonenmethode.

Statistik:

	UKA	WUSTL
LOC	565	448
Klassen	11	6
Kommentar	197	145
PCL	14	10

Fünf Teilaufgaben waren hierbei zu lösen:

1. Finden der richtigen Stelle für kleine Programmveränderungen (Änderung der Formatierung der Ausgabe)
2. Finden der richtigen Stelle für kleine Programmveränderungen (Änderung der Fenstergröße)
3. Schreiben einer zusätzlichen Ansicht (weiterer Beobachter), die die Schablonenmethode benutzt
4. Einbauen des in 3. geschriebenen Beobachters
5. Schreiben einer weiteren Ansicht, ohne Schablonenmethode

Wie beim And/Or-Tree Programm konnten die Teilnehmer bei UKA hier die Aufgaben 3 und 5 durch Verändern einer bereits existierenden Beobachter-Klasse lösenl. Allerdings reichte es aus wenn sie nur das Interface zu den geforderten Klassen angeben. WUSTL hingegen hatte keine Implementierung eines Beobachters gegeben und sollte die geforderten Klassen auch implementieren. Die Aufgaben 3 bis 5 waren entwurfsmusterrelevant.

2.2.3 Messungen

Für jede Versuchsperson separat wurde die Zeit von der Ausgabe der Aufgaben bis zu deren einsammeln gemessen und festgehalten. Eine Messung der benötigten Zeit zum Bearbeiten der einzelnen Teilaufgaben war nicht möglich, da sich nicht trennen lässt, wieviel Zeit zum Verstehen des Programms und wieviel Zeit zum eigentlichen Lösen der Aufgabe verwendet worden ist. Die Zeit für das Verstehen des Programms müsste also willkürlich auf die Teilaufgaben verteilt werden.

2.2.4 Hypothesen und Gültigkeit

Zu beweisen waren nun folgende Hypothesen:

- Durch Hinzufügen von PCL werden Wartungsarbeiten schneller beendet.
- Durch Hinzufügen von PCL werden bei Wartungsarbeiten weniger Fehler gemacht.

Innere Gültigkeit Das Ergebnis beeinflussende Faktoren wie die Programmiererfahrung, die Fähigkeiten, die Motivation der Teilnehmer usw. werden durch die zufällige Einteilung der Gruppen beglichen. Der gegenbalancierte Experimententwurf kompensiert eine mögliche „unglückliche“ Verteilung der Teilnehmer.

Die dominierende Störvariable war die Sterblichkeit bei WUSTL, da hier das Experiment am letzten Tag des Semesters stattgefunden hat, so dass die Studenten einfach aufgegeben haben, falls ihnen die Aufgabe zu schwer oder zu lang erschien, damit sie noch rechtzeitig ihren Bus oder ihr Flugzeug erwischen konnten. Allerdings haben in beiden Gruppen etwa gleich viele Versuchspersonen das Experiment abgebrochen, so dass die Ergebnisse von WUSTL immernoch aussagekräftig bleiben.

Obwohl die Teilnehmerzahl recht hoch war, lassen sich dennoch keine unstrittigen Ergebnisse erhalten.

Äußere Gültigkeit Im Vergleich zu realen Situationen war die Erfahrung der Teinehmer zu gering um die Ergebnisse verallgemeinern zu können. Es ist unklar wie erfahrene Entwickler PCL nutzen, denn möglicherweise brauchen sie PCL gar nicht um das Programm schneller verstehen zu können oder umgekehrt, sie können die Informationen vielleicht effizeinter nutzen.

Ein weiterer Punkt ist, dass im Experiment keine Teamarbeit zugelassen wurde, dabei könnte gerade bei der Teamarbeit PCL von Nutzen sein, da es einer bessere Kommunikation ermöglicht.

Die im Experiment zu bearbeitenden Programme haben außerdem unrealistische Große und Komplexität. Bei größeren Programmen existieren mehr Stellen die für die Wartung nicht relevant sind, die sich aber durch PCL schneller finden und somit für die Wartung ausschließen lassen.

Obwohl die Größe der Programme nicht denen in der Praxis gleicht, so ist doch der Aufbau der Programme relativ repräsentativ. Das Verhältnis der Anzahl der Klassen insgesamt zur Anzahl der zu Entwurfsmustern gehörenden Klassen entspricht in etwa dem großer Softwareprojekte (z.B. And/Or-Tree: 3,0; Phonebook: 5,5; Java AWT: 3,8; NextStep: 3,1).

Die Programme waren außerdem auch sehr gründlich kommentiert, was in der Praxis meist nicht der Fall ist. PCL könnte hier noch wertvoller sein um das Programm überhaupt zu verstehen.

2.2.5 Ergebnisse

And/Or-Tree

	mit PCL	ohne PCL	Signifikanz
UKA			
relevante Punkte (Durchschnitt)	8,5	7,8	0,20
korrekte Lösungen	15 von 38	7 von 36	0,077
Zeit (Minuten)	58,0	52,2	0,094
Zeit - nur korr. Lösungen	52,3	45,4	0,17
WUSTL			
relevante Punkte (Durchschnitt)	6,7	6,5	0,28
korrekte Lösungen	4 von 8	3 von 8	1
Zeit (Minuten)	52,1	67,5	0,046

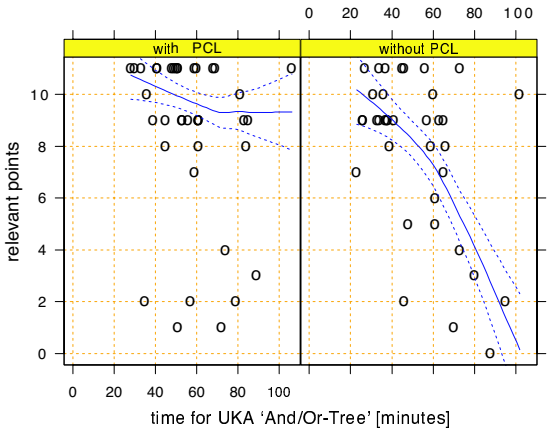
Beim Zusammenfassen der, von den Versuchspersonen, erreichten Punkten bei UKA zeigt sich, dass im Durchschnitt mehr „relevante“ Punkte, also die Punkte aus den entwurfsmusterrelevanten Aufgaben, mit PCL erreicht wurden. Allerdings hat dieses Ergebnis einen α-Fehler von 20%, d.h. es besteht die recht hohe Wahrscheinlichkeit von 20%, dass das Ergebnis nur zufällig die Hypothese unterstützt.

Wie in der Tabelle ersichtlich ist der Zeitbedarf mit PCL zwar größer, aber dafür gibt es auch wesentlich mehr korrekte Lösungen als ohne PCL. Falsche Lösungen würden in einem realistischen Szenario zu längerer Bearbeitungszeit führen, die im Experiment nicht berücksichtigt werden konnte, da die Fehler korrigiert werden müssten.

Da die Lösungen auf Papier ausgearbeitet werden mussten, war es sehr schwer für die Teilnehmer die Korrektheit ihrer Lösung zu überprüfen. Deshalb lassen

sich die Zeiten, die für korrekte und nicht korrekte Lösungen benötigt wurden, nicht vernünftig miteinander vergleichen. Daher werden nur die korrekten Lösungen miteinander verglichen. Hierbei zeigt sich, dass die Zeiten mit und ohne PCL in etwa gleich sind. Dennoch wurde für die korrekten Lösungen mit PCL geringfügig mehr Zeit benötigt, da es diese Gruppe den viel größeren Teil der gesamten korrekten Lösungen bildet und weil möglicherweise die Versuchspersonen weniger „fähig“ waren.

Aus dem Ergebnis, dass mit PCL viel mehr Lösungen korrekt waren, ließe sich schließen, dass durch PCL die weniger „fähigen“ Versuchspersonen dennoch in der Lage waren, korrekte Lösungen zu erstellen.



Wie aus der Grafik ersichtlich wird die Qualität der Lösungen je mehr Zeit dafür benötigt wird bei der Gruppe ohne PCL immer schlechter, während sie bei der Gruppe mit PCL die Qualität relativ unabhängig von der benötigten Zeit ist. Eine Erklärung dafür wäre, dass die Versuchspersonen durch PCL erkennen, dass sie eine Aufgabe nicht lösen können.

Bei der Analyse der Ergebnisse bei WUSTL zeigt sich, dass die Gruppe mit PCL schneller war als die ohne, d.h. mit einer Sicherheit von 90% wird mit PCL 0% bis 43% der Zeit für Wartungsarbeiten gespart.

Die Anzahl der korrekten Lösungen in beiden Gruppen ist allerdings gleich. Dies ist darauf zurückzuführen, dass die Teilnehmer ihre Lösungen implementieren und testen könnten.

Phonebook

	mit PCL	ohne PCL	Signifikanz
UKA			
relevante Punkte (Durchschnitt)	16,1	16,3	0,35
korrekte Lösungen	17 von 36	15 von 38	0,64
Zeit (Minuten)	51,5	57,9	0,055

Hier gab es bei UKA keinen signifikanten Unterschied bei den relevanten Punkten und der Anzahl der korrekten Lösungen. Allerdings fällt auf, dass die Ergebnisse bei den relevanten Punkten sehr hoch sind (mehr als 16 von möglichen 18), was darauf schließen lässt, dass die Aufgabe für die Versuchspersonen zu einfach war. Die Gruppe mit PSP war mit einer Sicherheit von 90% um 0% bis 22% schneller als die ohne PSP.

Die Ergebisse für WUSTL sind leider für eine sinnvolle Analyse unbrauchbar, da in beiden Gruppen nur jeweils eine korrekte Lösung vorkommt. In einer Analyse nach dem Experiment stellte sich heraus, dass die Aufgabe für die Teilnehmer zu schwer war. Dies lag wohl vor allem daran, dass die Teilnehmer einerseits keine praktischen Erfahrungen mit dem Beobachtermuster hatten und ihnen auch keine Beispielklasse zur Verfügung stand, andereseits durch die Ausgabe in den I/O Stream und durch eine GUI das Beobachtermuster in einem sehr unintuitiven Zusammenhang verwendet wurde. Es lässt sich also schlussfolgern, dass unter solchen Umständen das Vorhandensein von PCL wertlos ist.

Aus der Analyse der And/Or-Tree Aufgabe lässt sich die Schlussfolgerung ziehen, dass das Vorhandensein von PCL Zeit bei Wartungsarbeiten sparen könnte (WUSTL) und es helfen kann Fehler zu vermeiden (UKA).Die Analyse der Phonebook Aufgabe zeigt ebenfalls, dass PCL Zeit sparen kann. Die Ergebnisse aus der Analyse beider Aufgaben haben jeweils nur eine der beiden Hypothesen bestätgen können, jedoch nie beide gleichzeitig. Allerdings gibt es auch keinen Hinweis auf das Gegenteil der Hypothesen.

3 Bewertung

3.1 Experiment

Der generelle Experimententwurf ist grundsätzlich gut geeigent um den Vorteil der Verbesserten Kommunikation zwischen Entwicklung und Warung durch Entwurfsmuster zu zeigen. Die Durchführung selbst wies aber einige Mängel auf. So scheint es überhaupt nicht sinnvoll, dass die Lösungen der Aufgaben bei UKA auf Papier zu lösen waren. Dies entfernt das Experiment unnötig von realistischen Szenarien und erschwert das Auswerten der Ergebnisse, da die Zeiten korrekter und nicht korrekter Lösungen nicht mehr miteinander vergleichbar waren.

Aufgrund dieser Einschränkung musste auch die Aufgabenstellung angepasst werden, so dass nur noch Interfaces zu den geforderten Klassen geschrieben werden mussten. Insbesondere beim Phonebook Programm ist diese Aufgabenstellung nicht vernünftig, da die Interfaces der zwei geforderten Beobachter genau gleich sind, da dies im Entwurfsmuster Beobachter so vorgesehen ist.

Fragwürdig ist auch, welchen Zweck die nicht entwurfsmusterrelevanten Teilaufgaben hatten, da sie in die Auswerung des Ergebnisses nicht einbezogen werden konnten. Stattdessen haben sie möglicherweise sogar zu einem Ermüdungseffekt bei den Teilnehmern beigetragen, unter dem dann die Konzentration für die Lösungen der relevanten Teilaufgaben litt. Da allerdings zu beiden Programmen jeweils zwei nicht relevante Teilaufgaben zu beginn zu lösen waren, hat dies vermutlich kaum Einfluss auf die Ergebnisse.

Bei WUSTL verlief der Experiment noch etwas unvorteilhafter. Neben der Tatsache, dass zu wenige Teilnehmer mitgemacht haben und das der Termin recht ungünstig gewählt wurde, so dass der Sterblichkeitsfaktor sehr hoch war, wurden die Aufgabenstellungen auch so modifiziert, dass die Ergebnisse weder mit denen von UKA vergleichbar waren, noch, dass sie zumindest beim Phonebook Programm relevant im Sinne des Experimentes waren. D.h. günstiger wäre es hier gewesen, die Aufgabenstellung von UKA direkt zu übernehmen oder neue Aufgaben, die das Ziel des Experimentes unterstützen, zu entwickeln, aber nicht mehr oder weniger krampfhaft die Aufgabenstellung für UKA für die unterschiedliche Arbeitsumgebung bei WUSTL anzupassen.

3.2 Aufsatz

Der Aufsatz zum Experiment ist gut strukturiert aufgebaut und Analysen der Ergebnisse der Experimente sinnvoll und gut begründet. Allerdings wird nicht kritisch auf die Durchführung des Experiments selbst eingegangen. Nur die Begründung, warum beim And/Or-Tree Programm die Gruppe mit PCL bei UKA etwas langsamer ist als die ohne erscheint nicht vernünftig. Die Autoren meinen, dieses Ergebnis sei damit zu erklären, dass sich in der Gruppe mit PCL weniger „fähige“ Teilnehmer befanden. Daraus ziehen sie anschließend die Schlussfolgerungen, weniger „fähige“ Teilnehmer können durch PCL dennoch richtige Lösungen erstellen oder aber besser erkennen, dass sie die Aufgabe nicht lösen können und früher aufgeben. Der hierbei angenommene Faktor der Fähigkeit der Teilnehmer sollte allerdings, wie zuvor von den Autoren erwähnt, durch das gegenbalancierte Experimentdesign mit zufälliger Einteilung in die Gruppen kompensiert sein. Beide Schlussfolgerungen sind somit nur unbegründete Hypothesen und sollten nicht in der Auswertung der Ergebnisse auftauchen.

3.3 Zusammenfassung

Um zu einem eindeutigerem und besser verallgemeinbarerem Ergebnis zu kommen hätten die zuvor genannten Fehler bei der Durchführung des Experiments vermieden werden müssen. Für eine wissenschaftlich formale und eindeutigere Antwort auf die im Aufsatz gestellten Fragen, müsste eine erneute Untersuchung zu diesem Thema angestellt werden. Allerdings halte ich die Fragestellung an sich für trivial, da es intuitiv klar erscheint, dass es einfacher ist, wenn man zum Verstehen eines Programms bereits erworbenes abstrakteres Wissen anwenden kann als wenn man sich erst in die Begriffswelt der Anwendungsdomäne des Programms einarbeiten muss. Eine Empfehlung PCL zum Programmcode hinzuzufügen sollte also ausreichen, zumal der Mehraufwand für den Entwickler minimal ist, der Nutzen aber offensichtlich scheint.

4 Literaturverzeichnis

1. Karlsruhe Empirical Informatics Research Group (EIR): *PatDoc: on design pattern documentation during maintenance.* `http://www.ipd.uka.de/EIR/patdoc/index.html`

2. Lutz Prechelt, Barbara Unger, Michael Philippsen, Walter F. Tichy. *Two Controlled Experiments Assessing the Usefulness of Design Pattern Documentation in Program Maintenance.* Accepted by IEEE Trans. on Software Engineering, July 2001, to appear.