Empirische Methodik in der Softwaretechnik im Allgemeinen und bei der Software-Visualisierung im Besonderen

Walter Tichy, Frank Padberg

Informatik
Universität Karlsruhe
76128 Karlsruhe
tichy@ipd.uka.de, padberg@ipd.uka.de

Abstract: Empirische Untersuchungen sind ein fundamentaler Bestandteil der Forschung in der Softwaretechnik geworden. Software-Entwicklungsmethoden und Werkzeuge werden empirisch untersucht, um sie zu verstehen, zu evaluieren, einzusetzen und zu verbessern. Dieser Artikel erläutert einige der wichtigsten empirischen Methoden, die in der Softwaretechnik und insbesondere bei der Visualisierung von Software einsetzbar sind (Korrelationsstudien, Fallstudien, Umfragen, Ethnografie, Metastudien, kontrollierte Experimente) und geht auf einige wichtige Aspekte wie Studenten versus professionelle Softwareentwickler sowie die Technik des Akzeptanztests ein.

1 Einführung

Empirische Untersuchungen haben sich einen festen Platz in der Softwareforschung erobert [HT07]. Zu groß und variantenreich ist das Repertoire an Methoden und Werkzeugen zur Softwarebearbeitung geworden, als dass jeder Softwaretechniker oder auch nur jedes Softwarehaus die beste Methoden- und Werkzeugkombination aus eigener Anschauung bestimmen könnte. Auch sind die Einsatzbedingungen wichtig für die Effektivität von Methoden und Werkzeugen, so dass nur sorgfältig durchdachte Studien glaubwürdige Ergebnisse liefern können. Entsprechend wird in führenden wissenschaftlichen Tagungen und Zeitschriften erwartet, dass Vorschläge über neue Werkzeuge oder Methoden von empirischen Studien über ihre Brauchbarkeit begleitet sind.

Generell untersucht die empirische Softwaretechnik das Verhalten von Menschen, wenn sie mit Software umgehen und versucht dabei, verschiedene Methoden zu evaluieren, zu vergleichen, zu verstehen, die Einsatzbedingungen zu beschreiben, sowie Ansätze für Verbesserungsmöglichkeiten zu entdecken. In letzter Zeit haben die Softwarearchive von (oftmals quelloffenen) Projekten und die begleitenden Fehlermeldungsdatenbanken realistische Untersuchungen mit großen Datenmengen ermöglicht. Allerdings können nicht alle Fragen mit den Projektdaten aus diesen Archiven beantwortet werden; auch andere Untersuchungsmethoden, die Nutzer von Softwaretechniken direkt beobachten, sind von großer Wichtigkeit.

Das Instrumentarium der empirischen Methodik ist allerdings vielen Forschern in der Softwaretechnik noch wenig vertraut. Dieser Artikel erklärt die wichtigsten, quantitativen und qualitativen empirischen Methoden und ihre Anwendung, auch im Hinblick auf den speziellen Bereich der Softwarevisualisierung. Softwarearchive sind eine mögliche Quelle für Daten, deren Visualisierung von Interesse ist, jedoch erschließt sich der Nutzen von Visualisierungen erst durch Beobachtung der Nutzer.

Zunächst bedeutet Empirie nichts weiter als auf Erfahrung oder Beobachtung fußende Erkenntnis. Die vielfältigen empirischen Methoden unterteilt man zunächst in experimentelle und deskriptive Forschung. Experimentelle Forschung versucht mittels kontrollierter Experimente Kausalitätsbeziehungen, also Wirkungszusammenhänge zu identifizieren. Deskriptive Forschung beschreibt Phänomene, Ereignisse oder Situationen. Sie prüft in der Regel keine Kausalitätsbeziehungen, ist deswegen aber nicht weniger wichtig. Ferner gibt es die Unterscheidung in qualitative und quantitative Studien. Eine quantitative Studie sammelt numerische Daten, um eine gegebene Forschungsfrage zu beantworten, zum Beispiel die Fehlerdichte oder die Entwicklungszeit für Software. Qualitative Forschung sammelt nicht-numerische Daten, z.B. beobachtetes Verhalten bei der Verwendung einer bestimmten Softwarewerkzeugs, Antworten aus Interviews, oder schriftliche Dokumente. Deskriptive Forschung kann sowohl quantitativ als auch qualitativ sein, während experimentelle Forschung i.d.R. quantitativ ist. Abbildung 1 gibt eine Übersicht über diese Klassifizierung, die Christensen [Chr07] folgt. Aus Platzmangel können wir nur die wichtigsten Studientypen erklären.

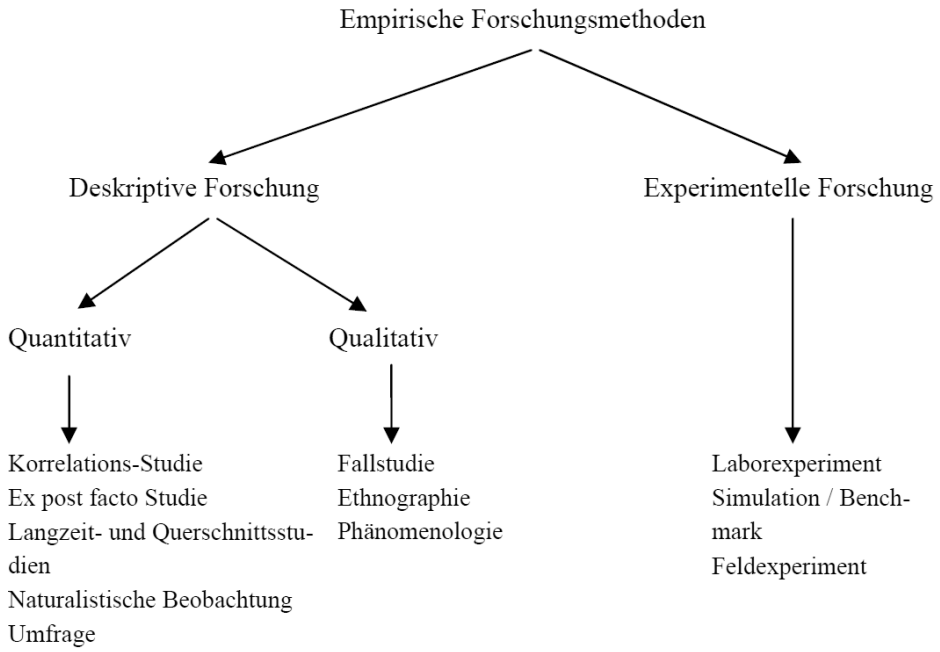

Abbildung 1: Klassifikation empirischer Forschungsmethoden

2 Quantitative, deskriptive Forschungsmethoden

2.1 Korrelationsstudie

Eine Korrelationsstudie ist ein quantitatives Verfahren, das zwei oder mehr Variablen misst und ihre Wechselbeziehung bestimmt. Statistische Tests liefern Korrelationskoeffizienten, die die Stärke des Zusammenhangs angeben. Eine Regression quantifiziert dann den funktionalen Zusammenhang zwischen den Variablen.

Korrelationsstudien gibt es in der Softwaretechnik schon lange in Form von Untersuchungen, die Beziehungen zwischen Produkt- oder Prozessmetriken einerseits und Entwicklungsdauer, Teamgröße oder Fehlerdichte andererseits bestimmen. Mit einer verlässlichen Korrelation kann man Vorhersagen leisten: aus der Kenntnis einer Variable lässt sich die andere bestimmen. Allerdings lässt eine Korrelation keine Annahme über Kausalität zu, denn die beiden Variablen könnten von einer dritten, unbekannten Variablen beeinflusst sein. Ein bekanntes Beispiel hierzu ist die Korrelation zwischen Schuhgröße und Einkommen: Je größer die Schuhe, desto mehr verdient ihr Träger! Dieser verblüffende Zusammenhang sollte aber nicht dazu verleiten, sich schlecht sitzende Schuhe zu kaufen, denn eine dritte Variable, das Geschlecht, ist hier ausschlaggebend: Männer verdienen im Durchschnitt mehr und haben auch größere Füße. Wenn man die Untersuchung getrennt nach Geschlecht vornimmt, verschwindet die erwähnte Korrelation. Das Problem verdeckter Variablen ist natürlich auch in der Softwaretechnik bekannt, da es eine Vielzahl von Faktoren gibt, die sich auf ein Softwareprojekt auswirken. Padberg et al. [PRS04] illustrierten eine Methodik, wie man, ausgehend von vorliegenden Messdaten, systematisch sowohl lineare als auch nicht lineare Korrelation und Regression ermittelt.

2.2 Ex Post Facto Studie

In einer ex post facto Studie können die Variablen von Interesse nicht direkt manipuliert werden, sondern müssen im Nachhinein so akzeptiert werden, wie sie vorgefunden werden. Nehmen wir den Fall, dass Teilnehmer einer Studie wählen dürfen, ob sie bei einer Software-Aufgabe ein Visualisierungswerkzeug benutzen oder nicht. Der Experimentator ist daran interessiert, ob sich die beiden Gruppen in Lösungsqualität oder Durchführungstempo unterscheiden. Diese Studie scheint ein kontrolliertes Experiment zu sein (siehe Abschnitt 4), hat aber eine grundsätzliche Schwäche: der Experimentator hat keinen Einfluss darauf, welche Teilnehmer das Visualisierungswerkzeug wählen und welche nicht. Diejenigen, die das Werkzeug nehmen, sind unter Umständen besser motiviert, wollen dem Experimentator gefallen, sind im Umgang mit dem Werkzeug außergewöhnlich gut vorbereitet, sind intelligenter oder einfach nur neugierig. Diejenigen, die das Werkzeug nicht wählen, wollen eventuell zeigen, dass sie auch ohne modernen „Schnickschnack“ zurechtkommen und strengen sich deswegen besser an, was wiederum eine ungewollte Verhaltensänderung bedeutet. Das Kernproblem hierbei ist, dass zwei nicht-äquivalente Gruppen verglichen werden, die durch Selbst-Zuordnung (engl. self as-

signment) entstanden sind. Der Experimentator kann nur beobachten, wie sich bereits vorhandene, individuelle Unterschiede auswirken. Eine Aussage über den Unterschied bei Benutzung und Nicht-Benutzung des Werkzeugs wäre nicht zulässig. Studien von Softwareentwicklung im Nachhinein, insb. von Softwarearchiven, weisen potenziell diese Schwäche auf, wenn sie z.B. unterschiedliche Methoden vergleichen. Ein ausführliche Kritik einer ex post facto Studie findet sich in Referenz [BT03].

2.3 Langzeit- und Querschnittsstudie

Eine Langzeitstudie verfolgt eine Gruppe von Teilnehmern über einen längeren Zeitraum, um zeitliche Änderungen der Individuen zu erfassen. Eine interessante Frage wäre zum Beispiel, ob Software-Entwickler gewisse Diagrammtypen, z.B. UML-Diagramme, über einen längeren Zeitraum gleichmäßig, abnehmend, oder zunehmend benutzen, oder ob sie mit der Zeit bestimmte Visualisierungen bevorzugen. Beispielsweise stellten Hulkko et al. [HA05] in einer achtwöchigen Studie fest, dass der Anteil an Paarprogrammierung in einem Team über die Zeit stark abfallen kann. Damit wäre die Nutzung dieser Methode auf einen Neuigkeitseffekt zurückzuführen, oder aber es gab Zwänge, die den Paarprogrammierungsanteil mit der Zeit senkten.

Im Gegensatz zu einer Langzeitstudie untersucht eine Querschnittsstudie eine Stichprobe einer Population zu einem festen Zeitpunkt, betrachtet dabei aber Unterschiede wie Alter, Geschlecht, und Ausbildung. Im Softwarekontext wäre die Nutzung von bestimmten Werkzeugen über verschiedene Alters- oder Ausbildungsgruppen hinweg interessant. Dabei ist die Kohortenbildung über das Alter zu beachten. Fünfzigjährige von heute haben z.B. nicht bereits im Schulalter Erfahrung mit Internet, Mobiltelefonie und Computerspielen gemacht. Weil bei Querschnittsstudien die Teilnehmer nicht ähnliche Erfahrungen gemacht haben, ist die Langzeitstudie zuverlässiger, aber auch wesentlich aufwändiger. Beide Forschungstypen sind in der Informatik noch extrem selten.

2.4 Umfrage

Eine Umfrage erhebt Information durch Befragung einer repräsentativen Stichprobe aus einer bestimmten Population. Jeder kennt Ergebnisse von Umfragen, z.B. Wahlprognosen, Lehrevaluationen oder Statistiken zur Kundenzufriedenheit. Umfragen können sich sowohl auf objektive als auch subjektive Sachverhalte beziehen. Subjektive Faktoren wie Stress, Zufriedenheit, Bekanntheit oder Angemessenheit eines Verfahrens sind meist nur durch Umfragen ermittelbar. Bei Visualisierungswerkzeugen könnte die Frage lauten, ob die Nutzer die Visualisierung als nützlich oder angemessen empfinden, ob sie mit dem Werkzeug Schwierigkeiten haben und wie häufig sie es einsetzen. Die Datenerhebung bei Umfragen erfolgt im direkten Interview, durch das Versenden von Fragebögen, über das Telefon, oder über das Web. Manche Umfragen werden beim Besuch bestimmter Web-Seiten automatisch angeboten (engl. pop-up surveys), aber ihre Repräsentativität ist frag-

lich. Wichtig ist immer, eine repräsentative Stichprobe zu erhalten; telefonische Umfragen gehen dabei mit Zufallsgeneratoren vor. Wenn Fragebögen nicht zurückkommen, muss telefonisch nachgefragt werden. Generell ist die Repräsentativität des Rücklaufs wichtig und sollte in einem wissenschaftlichen Artikel thematisiert werden. Der Vorteil von Umfragen ist, dass sie einfach durchzuführen sind und einen großen Stichprobenumfang erreichen können. Natürlich gilt, dass was nicht gefragt wurde, auch nicht beantwortet werden kann. Daher sollten neben Ankreuzfragen auch Freitextantworten zugelassen werden. Fragen sollten möglichst neutral formuliert werden und jeder Punkt auf einer vorgegebenen Skala sollte „ruhigen Gewissens“ angekreuzt werden können.

Umfragen werden auch gerne Experimenten vor- und nachgeschaltet. In einer vorgeschalteten Umfrage wird z.B. Information über Alter, Ausbildungsstand und Erfahrung der Experimentteilnehmer erhoben. Der Nachtest-Fragebogen prüft, ob die Teilnehmer die Aufgaben verstanden hatten, ob es Probleme gab, oder ob die Teilnehmer ahnten, auf was das Experiment abzielte. Der letzte Punkt ist besonders wichtig, da Wissen über den Zweck des Experiments das Verhalten der Teilnehmer beeinflussen und daher die Daten verfälschen kann. Daly et al. [DMB+95] veröffentlichten eine Umfrage, die eine Vorstudie zu einem Experiment über Vererbungstiefe war.

2.5 Meta-Analyse

Meta-Analysen (auch Metastudien) vereinigen die Daten aus verwandten Experimenten, um zu genaueren Aussagen zu kommen. Bei kleinen Stichprobengrößen eines Einzelexperiments kann oftmals die gewünschte statistische Signifikanz nicht erreicht werden, wohl aber durch das Zusammenfassen mehrere Studien. Prechelt et al. [PUPT03] fasste die Ergebnisse von drei Experimenten über Vererbungstiefe zusammen und kam dabei zu neuen Schlussfolgerungen. Noch sind in der Softwaretechnik Metastudien selten, jedoch sollten Veröffentlichungen so aufbereitet werden, dass sie in zukünftigen Metastudie verwendet werden können. Das bedeutet insbesondere, dass Forschergruppen, wie z.B. die Visualisierungsgemeinde, einen Konsens über die betrachteten Variablen und Messmethoden entwickeln, die Experimente genau beschreiben und die Rohdaten zugänglich machen sollten (z.B. im Anhang oder im Web). Ferner sollten Veröffentlichungen nicht allein deswegen abgelehnt werden, weil sie keine hohe statistische Signifikanz vorzuweisen haben; spätere Studien könnten die fehlenden Datenpunkte für eine Metastudie durchaus liefern.

3 Qualitative, deskriptive Forschungsmethoden

3.1 Fallstudie

Eine Fallstudie liefert eine eingehende Analyse und Beschreibung eines einzelnen oder einiger weniger Individuen, Organisationen oder Ereignisse. Die Untersuchung der Absturzursache der Ariane-5 ist ein Beispiel [JM97]. Als Quellen einer Fallstudie werden

Dokumente, Testergebnisse, Prüfberichte, Zeitungsberichte, Interviews u.ä. ausgewertet. Eine Fallstudie ist in der Regel qualitativ, obwohl sie auch quantitative Anteile haben kann. In der Softwareforschung werden Fallstudien of dazu benutzt, die prinzipielle Funktionsfähigkeit eines Werkzeuges oder einer Methodik zu zeigen. Fallstudien sind auch nützlich, um seltene Vorkommnisse zu erfassen oder ein bestimmtes Phänomen zu verstehen. Jedoch ist ihr Nutzen beschränkt, da die Ursache eines speziellen Ereignisses nicht mit Sicherheit festgestellt werden kann und eine Generalisierung der Beobachtungen nicht möglich ist. Man weiß manchmal nicht einmal, ob ein Phänomen tatsächlich mehrmals auftritt. Unter der Rubrik „Risks to the Public“ publiziert ACM SIGSOFT Software Engineering Notes seit Jahren zahlreiche Fallstudien über Aus- und Unfälle, die mit Software in Beziehung stehen. Eine ausführliche Fallstudie über Extreme Programming stammt von Müller [MT01].

3.2 Ethnografie

Ethnografie (auch deskriptive Feldforschung) ist die Beschreibung und Interpretation der Verhaltensweisen einer Gruppe von Menschen. Ethnografie will die Verhaltensmuster, Gepflogenheiten, Lebensart, kurzum die „Kultur“ dokumentieren, die sich zwangsläufig herausbildet, wenn eine Gruppe von Menschen länger zusammenlebt oder -arbeitet. Softwaretechniker haben ihre eigene Kultur, mit Subkulturen definiert z.B. durch Programmiersprachen, Softwareplattformen, Anwendungsbereichen oder Firmen. Um die Eigenarten einer solchen Gruppe zu bestimmen, würde ein Ethnograf zunächst Mitglied der Gruppe werden, um dann die natürlichen Verhaltensweisen der Gruppenmitglieder zu dokumentieren. Eine spannende ethnografische Studie stammt von Chong et al. [CH07]; sie beschreibt, wie Softwaretechniker, die Paarprogrammierung betreiben, zusammenarbeiten. Die Studie fand, dass die in der Literatur angenommenen Rollen von „Driver“ und „Navigator“ überhaupt nicht vorkommen. Vielmehr ergeben sich andere Verhaltensweisen, je nach Erfahrung, Wissensstand und Status der Gruppenmitglieder. Insbesondere arbeiten eingespielte Paare so intensiv zusammen, dass sie nicht einmal mehr Sätze vollständig aussprechen, sondern die Intention des Partners aus wenigen Gesten oder Äußerungen sofort erfassen. Ohne Ethnografie wären diese Verhaltensweisen nicht zu Tage gekommen. Studien über die Informationsbedürfnisse von Programmierern oder das Zusammenspiel von geografisch getrennten Teams wurden ebenfalls auf diese Weise untersucht. Es ist vorstellbar, dass auch weitere Software-Methoden durch intensive Beobachtung der Nutzer besser verstanden würden, z.B. die tatsächliche Nutzung von Werkzeugen wie etwa Programmierumgebungen, Testwerkzeugen oder Visualisierungshilfsmitteln.

4 Kontrollierte Experimente

Im Gegensatz zu den bis jetzt besprochenen, deskriptiven Methoden untersuchen kontrollierte Experimente Wirkungszusammenhänge. Im Experiment wird „durch planmäßiges Beobachten eines Sachverhaltes [...] und dessen Veränderung unter kontrollierten, [...]

wiederholbaren Bedingungen eine Hypothese bestätigt bzw. widerlegt“[1]. Man unterscheidet beim Experiment unabhängige und abhängige Variablen, sowie Störvariablen. Der Experimentator manipuliert die unabhängigen Variablen systematisch und beobachtet die Auswirkung auf die abhängigen Variablen. Die unabhängigen Variablen in der Softwareforschung sind typischerweise Methoden und Werkzeuge: eine Gruppe bekommt Werkzeug A, die andere Werkzeug B, oder eine Gruppe arbeitet in unter „normalen“ Bedingungen (sie heißt dann die Kontrollgruppe), die andere unter veränderten Bedingungen[2]. Eine Übersicht über die inzw. zahlreichen kontrollierten Experimente in der Softwaretechnik findet sich bei Sjberg et al. [SHH+05]. In der Softwarevisualisierung ist die zentrale Frage, ob sich Visualisierungen und Animationen bei der Ersterstellung, Analyse, oder Modifikation von Software auswirken.

Wichtig im Experiment ist, dass der Einfluss der Störvariablen kontrolliert wird, was heißt, dass ihre Werte konstant gehalten werden oder aber ihr Einfluss anderweitig neutralisiert wird. Wenn Störvariablen nicht kontrolliert würden, dann könnten Änderungen in den abhängigen Variablen auch auf die Störvariablen zurückgeführt werden. Wenn man aber ihren Einfluss ausschalten kann und Änderungen der unabhängigen Variablen sich auf die abhängigen auswirken, dann besteht ein kausaler Zusammenhang zwischen diesen letzteren beiden Variablen. Typische Störvariablen in Experimenten in der Softwaretechnik sind Fähigkeiten und Vorwissen der Experimentteilnehmer. Fähigkeits-Unterschiede werden durch Randomisierung kontrolliert, d.h. durch zufällige Zuweisung der Teilnehmer zu den Gruppen, die unter unterschiedlichen Bedingungen der unabhängigen Variablen arbeiten. Unterschiede im Vorwissen kann man durch Wahl der Teilnehmer (z.B. gesteuert durch Umfragen) oder gezielte Schulung mildern.

Objektive Beobachtung und Wiederholbarkeit sind ebenfalls wichtige Merkmale des Experiments. Objektiv ist eine Beobachtung, wenn sie unabhängig von der Person des Beobachters ist, wenn also jeder Experimentator die gleichen Messungen oder Veränderungen konstatieren würde. Das ist leichter gesagt als getan! In der Softwareforschung ist man oft damit konfrontiert, die Qualität von Lösungen zu beurteilen, was kaum objektiv möglich ist. Man behilft sich damit, dass man mindestens zwei Beobachter einsetzt und ihre Bewertungen vergleicht. Bei Abweichungen muss man den Grund für die unterschiedliche Einschätzung bestimmen und die Beurteilung soweit korrigieren oder verfeinern, dass in etwa gleiche Bewertungen herauskommen (engl. inter-rater agreement).

Die Wiederholbarkeit ist notwendig, um Ergebnisse überprüfbar zu machen, denn kein Experiment ist perfekt und jedem Experimentator können Fehler unterlaufen. (Fälle von bewusstem Betrug sind in der empirischen Softwareforschung noch nicht bekannt geworden.) Zur Wiederholbarkeit ist es notwendig, eine genaue Beschreibung des Experimentaufbaus und -verlaufs zu liefern sowie mögliche Fehlerquellen ehrlich zu diskutieren. In den meisten Veröffentlichungen erscheinen die potentielle Fehlerquellen unter den Überschriften „Konstrukt-Gültigkeit“ (wird das richtige Konstrukt richtig gemessen?), „interne Gültigkeit“ (liegt eine kausaler Zusammenhang vor?) und „externe Gültigkeit“ (auf welche Situationen lässt sich das Ergebnis generalisieren?). Selbst wenn ein Experiment

[1] Aus Microsoft Encarata, Deutsche Ausgabe, 2005.

[2] Die Bezeichnung „kontrolliertes Experiment“ kommt von der Kontrolle der Störvariablen, nicht von der Anwesenheit einer Kontrollgruppe.

nie wiederholt werden sollte, führt die Eigenschaft, dass es für Wiederholung ausgelegt ist, zu besseren Experimenten und glaubwürdigeren Ergebnissen.

Die wichtigsten abhängigen Variablen in der Softwareforschung sind die Zeit und die Qualität von Arbeitsprodukten, die unter Experimentbedingungen erstellt werden. Die Zeit lässt sich einfach und objektiv messen. Die Qualität der Arbeit kann über die Anzahl vorhandener Fehler bestimmt werden. Bei ausführbaren Programmen empfiehlt es sich, umfangreiche Testsuiten ablaufen zu lassen. Allerdings sind diese beiden Variablen nicht voneinander unabhängig: niedrige Qualität lässt sich sehr rasch erzeugen, während hohe Qualität Zeit braucht. Manche Teilnehmer werden also rasch fertig, liefern aber viele Defekte ab, während andere lange brauchen aber alle Defekte eliminieren. Und dann gibt es diejenigen, die schnell und gut arbeiten, und solche, die auch bei Zeitverlängerung nicht zu Rande kommen. Wie sollen solche Datenpunkte verglichen werden? Hierbei hat sich der Akzeptanztest bewährt: Teilnehmer führen zu einem geeigneten Zeitpunkt einen automatisch ablaufenden Akzeptanztest durch und müssen dann die auftretenden Fehler komplett entfernen. Damit wird eine vergleichbare Qualität erreicht, und man misst die Zeit, die bis zum Bestehen des Akzeptanztests benötigt wird. Außerdem misst man in einem großen Zufallstest die verbleibende Fehleranzahl. Mit dem Akzeptanztest ist eine Normierung geschaffen, mit der die Arbeitsgeschwindigkeit der Teilnehmer erst verglichen werden kann. Ohne den Akzeptanztest müsste man ein faktorielles Experiment durchführen, in dem sowohl Zeit als auch eingesetzte Methode die abhängige Variable Qualität beeinflussen. Das ist zwar möglich, erfordert jedoch wesentlich mehr Datenpunkte. Die Alternative wäre, statt einer vergleichbaren Qualität eine fest Zeitschranke vorzugeben. Hier tritt das Problem auf, wie diese Zeitschranke zu wählen ist. Schnelle Teilnehmer sind trotzdem früher fertig, und für andere ist die Zeitschranke zu kurz, um zu einer Lösung zu kommen. Die Methode des Akzeptanztests wurde erstmalig von Müller und Hagner [MH02] im Experiment eingesetzt.

Bei kontrollierten Experimenten treten zahlreiche Gefährdungen der inneren Gültigkeit auf. Die wichtigsten sind: Subjekteffekte (Motivation, positive Selbstdarstellung, Reifung, Historie, Hawthorne-Effekt,), Instrumentierungseffekte, Selektionseffekte, Experimentatoreffekte (Erwartungshaltung, tendenzielle Interpretierung), und Folgeeffekte. Die Diskussion dieser Effekte und der Experimentaufbauten, die sie neutralisieren können, nimmt in den entsprechenden Lehrbüchern beachtlichen Raum ein. Der Leser sei auf [Chr07], Kap. 8 bis 11 oder [SCC02], Kap. 4 bis 8 verwiesen.

Ein häufiger Diskussionspunkt betrifft studentische Experimentteilnehmer. Offensichtlich sind Ergebnisse nur dann von Studenten auf professionelle Softwareentwickler übertragbar, falls Studenten über vergleichbare Ausbildung und Übung verfügen, was die Lösung der ihnen gestellten Aufgaben betrifft. Mit Sicherheit kann man die Vergleichbarkeit nur feststellen, in dem man Professionelle und Studenten in einem Experiment vergleicht oder ethnografische Studien durchführt. Hierbei können durchaus unerwartete und subtile Unterschiede zutage treten. Allerdings können Studenten in Vorstudien benutzt werden, um einen Trend festzustellen oder alternative Hypothesen abzulehnen. Studenten sind auch in ihrem Vorwissen deutlich homogener als Professionelle, wodurch kleinere Effektgrößen sichtbar werden. Ferner sind Probeläufe von Experimente mit studentischen Teilnehmern eine Vorbedingung zur Durchführung mit professionellen. Eine ausführlichere Diskussion

dieses Themas findet sich in [Tic00]. Die Artikel [PV98] und [AS04] beschreiben Experimente, bei denen sich Studenten und Professionelle einmal gleich und einmal unterschiedlich verhalten.

5 Theoriebildung

In der Softwaretechnik liegt mittlerweile eine große Zahl empirischer Einzelergebnisse zu verschiedenen Werkzeugen und Methoden vor. Um weiteren Fortschritt in der Forschung zu erzielen, ist es notwendig, diese Ergebnisse zu strukturieren und zu bündeln. Das Ziel empirischer Arbeiten ist ja nicht nur, die Phänomene in der Softwareentwicklung aufzudecken und zu beschreiben, sondern die zugrundeliegenden Mechanismen auch erklären zu können – kurz: Theorien zu bilden (engl. explanatory theories). Eine gute Theorie ist in der Lage, die empirischen Beobachtungen zu erklären, Widersprüche in empirischen Ergebnissen aufzulösen und die weitere Forschung systematisch auf lohnende Fragestellungen auszurichten. Im Bereich der Werkzeuge zur Softwarevisualisierung etwa sollten Theorien erklären, warum ein Werkzeug in einem bestimmten Kontext und für einen bestimmten Zweck „besser“ ist als andere; sie sollten auch fundierte Hinweise geben, welche neuen Visualisierungsansätze sich lohnen würden.

Hat man eine Theorie aufgestellt, um damit bestimmte Beobachtungen in der Softwaretechnik zu erklären, besteht der nächste Schritt der Forschung darin, die Theorie auf ihre Stichhaltigkeit zu prüfen, indem man Vorhersagen aus der Theorie ableitet und gezielt experimentell überprüft. Falls nötig, wird die Theorie danach angepasst oder durch eine neue ersetzt. Dieser übliche wissenschaftliche Zyklus aus Empirie, Theoriebildung und Validierung muss sich in der Softwaretechnik noch weiter durchsetzen. Schon bei einer einzelnen empirischen Studie (vor allem bei kontrollierten Experimenten) sollte eine Theorie vorhanden sein, auch wenn sie noch unvollständig, vorläufig oder „klein“ ist; das gibt der empirischen Studie ein klares Ziel.

Es gibt bisher erst wenige gute Theorien in der Softwaretechnik. Dabei muss eine Theorie nicht immer quantitativ sein; qualitative Theorien sind oft genauso wertvoll. Ein Beispiel ist die qualitative Theorie von [SJLY00] zur Effektivität von Software-Inspektionen. Durch Anlehnung an die Verhaltensforschung für Gruppen können die Autoren erklären, warum die Effektivität von Inspektionen am besten dadurch zu steigern ist, dass die individuellen Fähigkeiten der Inspektoren bei der Fehlersuche verbessert werden, und weniger durch Verbesserungen am Inspektionsprozess. Die Theorie erklärt auch einige widersprüchliche Ergebnisse von Einzelexperimenten zu Inspektionen und richtet die weitere Forschung auf die Verbesserung der Lesetechniken aus. Eine stochastische Theorie von Padberg [Pad06] erklärt, wie sich Änderungen in Entwürfen ausbreiten und die Kosten hochtreiben.

6 Fazit

In der Softwareforschung wird inzwischen ein ganzer Strauß von empirischen Methoden erfolgreich angewandt. Empirische Studien bilden selbstverständlich nur einen Teil der Forschung; die beiden anderen Säulen sind die Technologieentwicklung (Entwicklung neuer Methoden und Werkzeuge zur Softwarebearbeitung) sowie die Theoriebildung. Letztendlich ist das Ziel der Softwareforschung ein Erkenntnisgewinn, der zum Verständnis dessen beiträgt, was bei der Softwareentwicklung und -wartung eigentlich vor sich geht, um aus dieser Kenntnis die Softwarebearbeitung zu verbessern.

Literatur

[AS04] E. Arisholm und D. Sjøberg. A Controlled Experiment with Professionals to Evaluate the Effect of a Delegate versus Centralized Control Style on the Maintainability of Object-Oriented Software. *IEEE Transactions on Software Engineering*, 30(8):521–534, 2004.

[BT03] D. Berry und W. Tichy. Comments on: Formal Methods Application: An Empirical Tale of Software Development. *IEEE Transaction on Software Engineering*, 29(6):567–571, 2003.

[CH07] J. Chong und T. Hurlbutt. The Social Dynamics of Pair Programming. In *29th International Conference on Software Engineering*, 2007.

[Chr07] L. B. Christensen. *Experimental Methodology*. Pearson, 2007.

[DMB+95] J. Daly, J. Miller, A. Brooks, M. Roper und M. Wood. A Survey of Experiences amongst Object-Oriented Practitioners. In *APSEC 2*, Seiten 137–146, 1995.

[HA05] H. Hulkko und P. Abrahamsson. A Multiple Case Study on the Impact of Pair Programming on Product Quality. In *27th International Conference on Software Engineering*, Seiten 495–504, 2005.

[HT07] A. Höfer und W. Tichy. Status of Empirical Research in Software Engineering. *Empirical Software Engineering Issues, LNCS*, 2007. Im Druck.

[JM97] J-M. Jezequel und B. Meyer. Design by Contract : The Lessons of Ariane. *IEEE Computer*, 30(1):129–130, 1997.

[MH02] M. Müller und O. Hagner. Experiment about test-first programming. *IEE Proceedings Software*, 149(5):131– 136, 2002.

[MT01] M. Müller und W. Tichy. Case Study: Extreme Programming in a University Environment. In *23rd International Conference on Software Engineering*, Seiten 537–544, 2001.

[Pad06] F. Padberg. A Study on Optimal Scheduling for Software Projects. *International Journal on Software Process Improvement and Practice*, 11:7791, 2006.

[PRS04] F. Padberg, T. Ragg und R. Schoknecht. Using Machine Learning for Estimating the Defect Content After an Inspection. *IEEE Transactions on Software Engineering*, 30(1):17–28, 2004.

[PUPT03] L. Prechelt, B. Unger, M. Philippsen und W. Tichy. A Controlled Experiment on Inheritance Depth as a Cost Factor for Code Maintenance. *Journal of Systems and Software*, 65(2):115–126, 2003.

[PV98] A. Porter und L. Votta. Comparing Detection Methods for Software Requirements Inspections: A Replication Using Professional Subjects. *Empirical Software Engineering*, 3(4):355–379, 1998.

[SCC02] W. Shadish, T. Cook und D. Campbell. *Experimental and Quasi-Experimental Designs for Generalized Causal Inference*. Houhgton Mifflin, 2002.

[SHH+05] D.I.K. Sjøberg, J.E. Hannay, O. Hansen, V.B. Kampenes, A. Karahasanovic, N.-K. Liborg und A.C. Rekdal. A Survey of Controlled Experiments in Software Engineering. *IEEE Transactions on Software Engineering*, 31(9):733–753, 2005.

[SJLY00] D. Sauer, R. Jeffery, L. Land und P. Yetton. The Effectiveness of Software Development Technical Reviews. *IEEE Transactions on Software Engineering*, 26(1):1–14, 2000.

[Tic00] W. Tichy. Hints for Reviewing Empirical Work in Software Engineering. *Empirical Software Engineering*, 5(4):309–312, 2000.