Universität Hannover
Fachgebiet Software Engineering
Institut für Angewandte Systeme
Fachbereich Informatik

Konzept und Realisierung eines Werkzeuges zur Charakterisierung und Analyse von Software Engineering-Experimenten

Bachelorarbeit

im Studiengang Informatik

von

Jörg Eggermann

Prüfer: Prof. Dr. Kurt Schneider
Zweitprüfer: Prof. Dr. Nicola Henze

Hannover, den 01. September 2005

Erklärung

Hiermit versichere ich, dass ich diese Arbeit selbständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet wurden.

Hannover, den 01. September 2005

Jörg Eggermann

Zusammenfassung

Eine aktuelle Entwicklung des Software-Engineerings ist der verstärke Einsatz von Experimenten. Diese haben sich in anderen Fachbereichen bereits als ein fundamentales Mittel behauptet, um aufgestellte Thesen zu beweisen oder zu widerlegen. Ein großes Problem der Experimente im Software-Engineering stellt ihre Planung und ihre Analyse dar. Denn anders als in den klassischen Einsatzgebieten - der Physik und Chemie - ist hierbei nichtdeterministisches menschliches Verhalten eine entscheidende Experimentvariable.

In dieser Arbeit wird ein Programm vorgestellt, das Experimente im Software-Engineering anhand ausgewählter Metriken vergleichbar und somit präzise planbar macht.

Inhaltsverzeichnis

1 Einleitung

Software findet man heutzutage in immer mehr Produkten. Das bedeutet, dass eine große Vielzahl an Software entwickelt wurde und auch noch entwickelt wird. In diesem Prozess können viele Probleme auftreten.

Größere Software Projekte benötigen viel Zeit, sind sehr komplex, involvieren viele Entwickler und kosten viel Geld. Daraus resultiert die Notwendigkeit Prozesse zu optimieren bzw. die richtigen für ein Projekt zu finden, um Kosten zu sparen, Produkte verbessern zu können und konkurrenzfähig zu bleiben. Dafür gibt es eine Vielzahl von Methoden und Werkzeugen zur Unterstützung der Softwareentwicklungsprozesse.

Gängige Annahmen dabei sind:

- wir verwenden Techniken, denen wir vertrauen, dass sie ein bestimmtes Ergebnis liefern
- wir erwarten, dass sich die Entwicklungszeit unter Verwendung von Werkzeug XY verkürzt
- wir nehmen an, dass die Qualität des finalen Produkts besser sein wird, wenn bestimmte Entwicklungsprozesse angewendet werden
- etc.

Aber müssen diese Annahmen immer wahr sein? Und unter welchen Rahmenbedingungen müssen sie angewendet werden, um zum Erfolg zu führen? Welche der vielen Veröffentlichungen für Prozessverbesserungen sind für das aktuelle Projekt relevant und könnten einen Nutzen bieten?

Bisher haben sich über längere Zeit der Anwendung die brauchbaren herauskristallisiert. Wie aber wird dieser Zeitpunkt ermittelt?

Es ergibt sich der Wunsch nach bewährten wissenschaftlichen Methoden, um konkrete Bewertungen vornehmen zu können, unter welchen exakten Rahmenbedingungen welches Tool oder welche Methode am effizientesten ist, um sie dann für das eigene Projekt anwenden zu können. Des Weiteren ist es schwierig, einen Prozess im Vergleich zu einem Produkt auszuwerten oder zu beurteilen. Für Produkte lassen sich Prototypen bauen, anhand derer sich die weitere Verwendbarkeit bestimmen lässt. Problematisch für Prozesse

ist, dass sich keine Prototypen erstellen lassen, da sie bis zur Anwendung nur Beschreibungen darstellen.

Genau hier setzen Experimente an. Bekannt aus den Naturwissenschaften bieten sie eine systematische, quantifizierbare und kontrollierbare Bewertung für menschlich basierte Aktivitäten. Experimente im Software Engineering ermöglichen neue Methoden, Prozesse etc. mit existierenden zu vergleichen und unter genau festgelegten Rahmenbedingungen und wissenschaftlichen Aspekten eine Bewertung vorzunehmen, um dann gezielt die benötigten einzusetzen. Diese Gründe führen dazu, dass sich Experimente im Bereich Software Engineering immer größerer Beliebtheit erfreuen.

Auch im Fachgebiet Software Engineering des Instituts für Angewandte Systeme der Universität Hannover beschäftigt man sich mit Experimenten aus diesem Bereich. Im Wintersemester 2004/2005 wurde beispielsweise ein Experiment durchgeführt, um verschiedene Techniken der testgetriebenen Softwareentwicklung zu vergleichen. Im Einzelnen waren dies TestFirst und automatisiertes Testen. Während der Planung und Durchführung des Experiments traten einige Probleme auf, die zu dem Wunsch nach einem Werkzeug zur Erleichterung der Experimentplanung führten.

Hier knüpft das Ziel dieser Arbeit an: Konzept und Realisierung eines Werkzeuges zur Charakterisierung und Analyse von Software Engineering Experimenten.

1.1 Zielsetzung

Das Ziel dieser Arbeit ist die Entwicklung eines Werkzeugs, welches bei der Planung, Ausführung und Auswertung von Experimenten im Bereich Software Engineering behilflich sowie in der Lage ist, Experimente zu analysieren und zu charakterisieren. Ferner soll das Programm durch eine bereitgestellte PlugIn Lösung erweiterbar sein.

U. Beck, welche ebenfalls an der Entwicklung des Werkzeugs beteiligt ist, beschäftigt sich mit dem Schwerpunkt der Komplexitätsabschätzung und –bestimmung von Experimenten.

1.2 Gliederung

Das zweite Kapitel wirft einen genaueren Blick auf Experimente. Die dazu folgenden theoretischen Betrachtungen erläutern die Einsatzgebiete von Experimenten, Unterschiede der Anwendungsbereiche, generelle Struktur, sowie eine detaillierte Betrachtung der einzelnen Experimentphasen und endet mit der Anwendung auf ein Beispielexperiment.

Im dritten Kapitel geht es um die Analyse und Charakterisierung von Experimenten. Es werden mathematische Methoden vorgestellt und deren Einsetzbarkeit für dieses Werkzeug geprüft. Anschließend werden für die zu erreichenden Ziele mittels einer im Software Engineering etablierten Methode Metriken zum konkreten Messen entwickelt.

Kapitel vier beschreibt die verschiedenen Anforderungen an das Werkzeug. Dabei werden gängige Dokumentationsmethoden aus dem Software Engineering Bereich verwendet, zum Beispiel Use Cases.

Das fünfte Kapitel zeigt den Entwurf sowie Implementierung und geht auf die Gründe der einzelnen Entscheidungen ein.

Im letzten Kapitel befinden sich die Zusammenfassung, sowie Möglichkeiten der Erweiterung.

2 Grundlagen der Experimente

Ein Experiment (lateinisch: experimentum = Versuch, Beweis, Prüfung, Probe) im Sinne der Wissenschaft ist ein methodisch aufgebauter Versuch zur zielgerichteten Untersuchung einer unter definierten Bedingungen reproduzierbar hervorgerufenen Erscheinung.

(Definition nach Wikipedia)

2.1 Einsatzgebiete von Experimenten

Experimente werden genutzt um Theorien zu testen, Hypothesen zu be- / widerlegen, Vor- und Nachteile von Methoden zu erkennen, zum Optimieren von Arbeitstechniken sowie effiziente Nutzung von Werkzeugen zu ermitteln. Die Stärke von Experimenten liegt darin, dass genau untersucht werden kann, in welchen Situationen die getroffenen Annahmen etc. wahr sind und den Kontext bieten, unter welchen Rahmenbedingungen Methoden und Werkzeuge am effizientesten anzuwenden sind. Ferner kann die Genauigkeit von Modellen bestimmt, bzw. untersucht werden, ob die Genauigkeit den Erwartungen entspricht. Des Weiteren können Schlüsse über kausale Beziehungen aufgestellt werden. Entscheidend dafür sind genaue Messungen der dem Experiment wichtigen Eingangs- und Ausgangsgrößen.

Zitat DeMarco: „*You cannot control what you cannot measure*“. [DeM82]

Experimente werden eingesetzt, wenn Verhaltensweisen direkt, gezielt und systematisch manipuliert werden sollen unter kontrollierten Bedingungen. Kontrolle über externe Faktoren kann auf mehreren Wegen erfolgen. In traditioneller Laboratoriumsforschung wird dies durch Regelung der physikalischen Umgebung, in der das Experiment durchgeführt wird, erreicht, beispielsweise durch Konstanthalten der Temperatur, des Luftdrucks oder der Luftfeuchtigkeit. Schwieriger wird es da bei Menschen, denn das Kontrollieren der externen Faktoren ist dort nur begrenzt möglich. Falls sich das Experiment über einen langen Zeitraum erstreckt, kann nicht erwartet werden, dass die Testpersonen über die gesamte Zeit von ihren Freunden oder Familien ferngehalten werden, sollte dies eine für das Experiment notwendige Bedingung sein. Gerade der außerexperimentelle Kontakt zu anderen

Menschen, die eventuell auch an dem Experiment teilnehmen, oder die eigene Familie kann die Testpersonen negativ beeinflussen und das Experiment verfälschen. Fenton [Fen91] behauptet beispielsweise, dass Ergebnisse aus dem Labor nicht unbedingt auf die Außenwelt übertragbar sind wegen des hohen Grades an Kontrolle. Ergebnisse, welche im Labor Gültigkeit besitzen, müssen dementsprechend in der Außenwelt nicht unbedingt zutreffen. Ferner besteht die Gefahr von Bedingungen auszugehen, die so nicht existieren. In der Chemie oder Physik kann man sich meist auf identische Materialien verlassen, welche nicht voneinander abweichen. Bei Menschen sieht das allerdings anders aus. Sie sind verschieden, besitzen unterschiedliche Erfahrungen und zu viele Eigenschaften, als das sich eine Reihe identischer Testpersonen finden lassen. Dementsprechend schwierig gestaltet sich auch das Messen bei Menschen, denn wie soll beispielsweise Motivation quantifizierbar gemessen werden? Um diese Gegebenheiten auszugleichen werden Experimente mit sehr vielen Testpersonen durchgeführt. Die Forschung mit Menschen ist daher sehr teuer und erschwert experimentelle Forschung teilweise erheblich im Bereich Software Engineering.

2.1.1 Relevante Maße im Software Engineering

Die für Software Engineering relevanten Objekte können in drei Klassen unterteilt werden. Der Prozess beschreibt, welche Aktivitäten notwendig sind, um die Software zu produzieren. Das Produkt bezeichnet durchzuführende Arbeiten oder Dokumente, welche durch einen Prozess entstehen. Die Ressourcen benennen beispielsweise das Personal, Hardware oder Software, welche für die Prozessaktivitäten benötigt werden. In jeder dieser drei Klassen wird zwischen internen und externen Merkmalen unterschieden. Externe Merkmale können dabei nur in Relation zu anderen Objekten gemessen werden, interne lassen sich üblicherweise direkt messen. Ziel ist es oft, Aussagen über externe Eigenschaften zu treffen, die meist nur indirekt zu messen sind, bzw. über interne Eigenschaften hergeleitet werden.

Klasse	Objektbeispiel	Merkmalstyp	Beispiel der Messgröße
Produkt	Code	intern	Umfang
		extern	Zuverlässigkeit
Prozess	Testen	intern	Aufwand
		extern	Kosten
Ressourcen	Personal	intern	Alter
		extern	Produktivität

(Abbildung 1: Beispiele für Maße im Software Engineering)

2.2 Kontrollierte Experimente

Nach [Pre01] ist ein kontrolliertes Experiment eine Studie, bei der alle voraussichtlich für das Ergebnis relevanten Umstände (*Störvariablen*) konstant gehalten werden (Kontrolle), mit Ausnahme von einem oder wenigen, die den Gegenstand der Untersuchung bilden (*Experimentvariablen*). Die Beobachtungen (*abhängige Variablen*) für verschiedene, gezielt ausgesuchte Werte der Experimentvariablen (*unabhängige Variablen*) werden miteinander verglichen, um so zu reproduzierbaren Aussagen zu kommen, die eine vor dem Experiment definierte Experimentfrage beantworten.

Unabhängige oder *Experimentvariablen* bezeichnen die im Laufe des Experimentes manipulierte Größe. Im Beispiel für das im folgenden Fall angenommene Experiment, einer Untersuchung der Unterschiede der Verständlichkeit zwischen Programmcode und Flussdiagrammen für unterschiedliche Programmgrößen, sind unabhängige Variablen die Betrachtung des Darstellungstyps und die Programmgröße in verschiedenen Versuchsbedingungen.

Eine *abhängige Variable* bezeichnet die im Laufe des Experiments gemessene Größe. Als Beispiel für die Untersuchung, welches von zwei Entwurfsverfahren für ein gegebenes Entwurfsproblem geeigneter ist, ist die gemessene Zeit des Entwurfsaufwandes oder die Fehleranzahl der entstehenden Entwürfe anzuführen.

Störvariablen bezeichnet die im Experiment konstant gehaltene Variablen. Im Prinzip sollen alle Störvariablen in einem kontrollierten Experiment gleichgehalten werden. Dieser Anspruch ist aber unrealistisch, denn man kann niemals den exakt gleichen Zustand der Welt herstellen. Glücklicherweise lassen sich fast alle Störvariablen mit derselben Technik kontrollieren, die in einem softwaretechnischen Experiment ohnehin angewendet wird. Gemeint sind *Randomisierung* und *Replikation*. Dabei wird nicht das Verhalten einer einzelnen Versuchsperson für jeden Wert der unabhängigen Variablen verglichen, sondern eine ganze Gruppe unter Betrachtung ihres durchschnittlichen Verhaltens. *Randomisieren* bezeichnet dabei das zufällige Auswählen der Mitglieder jeder Gruppe aus den verfügbaren Versuchspersonen. Somit wird ein möglicher systematischer Fehler (die Störvariable Z verändert unbekannterweise das beobachtete Ergebnis) in einen statistischen Fehler verwandelt (zufällig war die Störvariable Z für die verglichenen Gruppen nicht identisch), welcher sich mittels statistischer Methoden quantifizieren und beherrschen lässt. Jede solche Gruppe wird mit Versuchsgruppe oder Experimentgruppe bezeichnet. Zusätzlich kann es eine zweite Gruppe geben, die auf klassische Weise arbeitet, diese wird als Kontrollgruppe oder Vergleichsgruppe bezeichnet. Eine hohe Reproduzierbarkeit (*Replikation*) der Ergebnisse wird durch das genaue Festlegen und auch die Dokumentation aller Randbedingungen bzw. das Konstanthalten vieler Aspekte zur Kontrolle der übrigen Variablen erreicht.

Ein wichtiger Aspekt ist auch das *Vertrauen* in die Beobachtungen. Durch einen hohen Grad an Kontrolle über die Beobachtungsbedingungen, also die Festlegung und Überwachung der Variablen und Arbeitsbedingungen, wird ein hoher Grad an Vertrauen in die Beobachtung erzielt.

Die *innere Gültigkeit* eines kontrollierten Experiments ist der Grad, in dem die Änderungen in den Werten der abhängigen Variablen, tatsächlich wie gewünscht, nur auf Änderungen in den unabhängigen Variablen zurückzuführen sind. Das bedeutet, wie gut alle relevanten Störvariablen kontrolliert wurden, und dass das Ergebnis korrekt ist.

Die *äußere Gültigkeit* bezeichnet in einem kontrollierten Experiment den Grad, in dem sich seine Resultate korrekt auf andere Anwendungsfälle übertragen lassen, insbesondere auf solche, die in der Praxis häufig vorkommen. Dies betrifft zum Beispiel die Qualifikation der Versuchspersonen, Art und Größe der Software sowie Randbedingungen, wie sonstige softwaretechnische Methoden, räumliches Arbeitsumfeld etc. Die äußere Gültigkeit besagt also, ob das Ergebnis verallgemeinerbar ist.

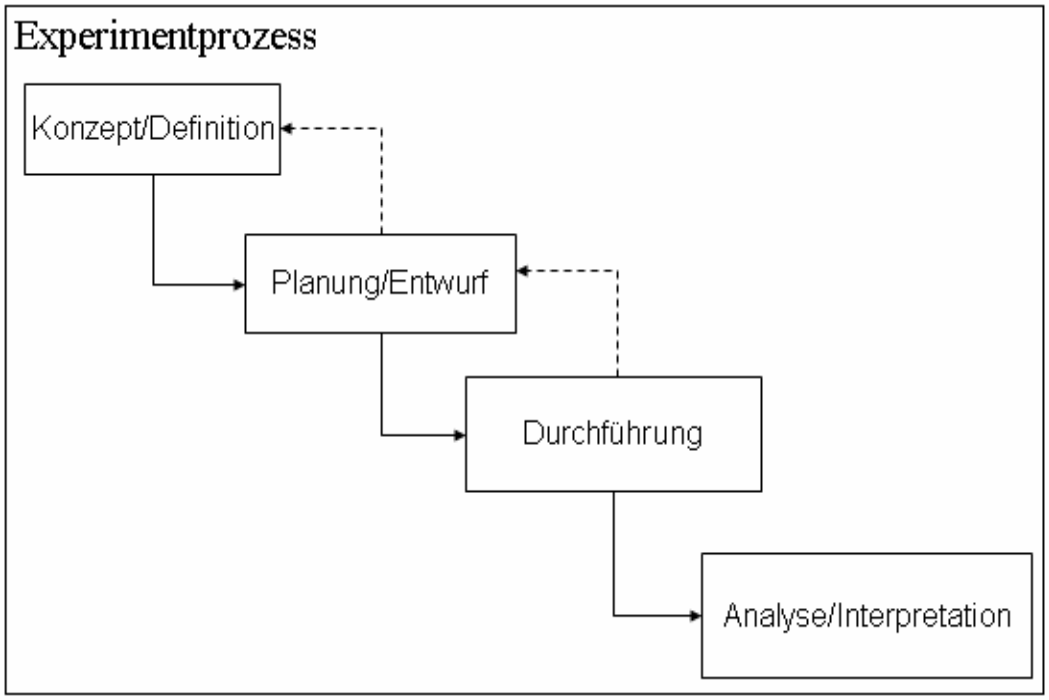
Der Zweck eines kontrollierten Experiments besteht darin, Beobachtungen zu machen, deren Ursachen eindeutig feststehen. Wenn etwas zweimal gemacht wird und dabei alle Umstände bis auf einen gleich sind, dann müssen eventuell auftretende Unterschiede in den Ergebnissen von der Änderung dieses einen Umstandes herrühren. Die bislang häufigste Rolle kontrollierter Experimente in der Softwaretechnik ist die Prüfung einer konkreten Vermutung, in der Regel von der Form „A ist besser als B bezüglich Eigenschaft E“. Dies wird meist mit Hilfe eines Hypothesentests erfolgen, in dem aufgestellte Hypothesen auf ihre Gültigkeit geprüft werden.

Sobald durch einen Hypothesentest eine Einflussgröße als solche nachgewiesen sowie die Größe ihrer Wirkung für konkrete Einzelfälle quantifiziert wurde, ergeben sich weiterführende Fragen: *Wie kommt diese Wirkung zustande und inwiefern würde sie sich unter anderen Umständen verändern?* Zur Beantwortung wird ein quantitatives Modell gesucht, welches den Einfluss und das Zusammenspiel aller beteiligten Variablen bei der Softwareentwicklung beschreibt. Das sind zum Beispiel Entwicklungsmethoden oder die Eigenschaften der Menschen. Die Zahl der Einflussgrößen ist in der Softwaretechnik sehr groß und ihre Wirkungen und Wechselwirkungen sind wenig bekannt, weswegen sich die meisten Modelle in der Softwaretechnik Feldstudien bedienen, um ihre Daten zu entnehmen. Es gibt aber auch Ausnahmen, in denen aufgrund von kontrollierten Experimenten Modelle gebildet wurden.

Abschließend stellt sich noch die Frage, welche Punkte wichtig sind, damit ein Experiment erfolgreich ist. Das resultierende Ergebnis sollte nicht das Ergebnis eines unbekannten Einflusses sein, das bedeutet, die Fehlerquote der Interpretation sollte optimiert werden. Das Ergebnis sollte wichtige Informationen zur Softwaretechnik liefern und den praktischen Einsatz oder die Forschung beeinflussen. Die Sicht auf ein Experiment wird geprägt durch die Relevanz der vom Experiment bearbeiteten Fragestellung, welche sich aus zwei Aspekten zusammensetzt: Wie bedeutsam ist die Forschungsfrage, zu deren Beantwortung das Experiment durchgeführt wird, und wie nützlich ist die konkret bearbeitete Experimentfrage zur Beantwortung der Forschungsfrage. Als begrenzende Faktoren fungieren dabei die zur Verfügung stehenden Ressourcen.

2.3 Phasen eines Experiments

Basili [Bas86] liefert ein Framework für den Aufbau von Experimenten. Damit lassen sich wichtige Entscheidungspunkte definieren sowie das Klassifizieren von Experimenten erleichtern. Diese Unterteilung in mehrere Phasen wurde im Laufe der Zeit von weiteren Personen aufgegriffen und teils verändert, siehe Manso [Man01] oder Pfleger [Pfl95], die grundlegende Beschreibung blieb dabei stets erhalten. Diese Arbeit beschäftigt sich weitgehend mit dem Entwurf von Basili.



(Abbildung 2: Phasen eines Experiments)

Dieser Prozess soll kein Wasserfallmodell darstellen, da nicht notwendigerweise eine Aktivität beendet sein muss, bevor die nächst folgende beginnt. Die Reihenfolge der Aktivitäten im Prozess soll die Startreihenfolge verdeutlichen, beginnend mit der Definition über die Planung und Durchführung bis letztendlich zur Interpretation. Dieser Prozess ist teils iterativ, und manchmal ist es notwendig, einen Schritt zurückzugehen und die vorherige Aktivität umzugestalten, in der Abbildung dargestellt anhand der gestrichelten Pfeile. Eine wichtige Ausnahme dabei ist, sobald die eigentliche Experimentdurchführung in der dritten Phase gestartet wurde, kann nicht mehr zurück zur Definition oder Planung gewechselt werden, da die Versuchspersonen durch aktive Teilnahme an dem Experiment beeinflusst werden. Würde nun abgebrochen und zurück zur Definition oder Planung gegangen werden, bestünde das Risiko, dass es unmöglich wäre dieselben Testpersonen für das geänderte Experiment zu nutzen und damit aussagekräftige Ergebnisse zu erzielen. Bevor die Phasen genauer betrachtet werden, folgt nun ein kurzer Überblick der einzelnen Aufgaben.

Konzept/Definition

- Ziele festlegen
- Experimentfrage aufstellen
- Hypothesen aufstellen

Planung/Entwurf

- Experimentvariablen festlegen
- Personenanzahl bestimmen
- Arbeitsumfeld festlegen

Durchführung

- Personen festlegen/ Randomisieren
- Schulung
- Aufgaben ausgeben
- Daten erheben

Analyse/Interpretation

- Datenvollständigkeit/ -konsistenz prüfen
- Datenauswertung
- Hypothesen be-/ widerlegen

2.3.2 Konzept / Definition

Ein Experiment durchzuführen ist eine arbeitsintensive Aufgabe. Um einen Nutzen im Vergleich zu den aufgewendeten Kosten ziehen zu können, muss sichergestellt werden, dass das Ziel des Experiments auch wirklich zufriedenstellend erfüllt wird. In der Definitionsphase wird das Fundament des Experiments gelegt. Hierbei aufgetretene Fehler bedürfen der Nacharbeit oder können sogar das Experiment unbrauchbar machen.

Startpunkt eines jeden Experiments ist eine Idee oder Vorstellung eines kausalen Bezugs. Diese Beziehung wird dann in einer Hypothese formuliert, welche durch das Experiment bewertet werden soll. Die Hypothese muss klar ausgedrückt werden, aber an diesem Punkt noch nicht zwingend formal korrekt ausgearbeitet sein. Es muss aber hervorgehen, was genau gemessen wird und welche Phänomene untersucht werden. Beispielsweise soll herausgefunden werden, wie effizient zwei individuelle Testmethoden in Bezug auf das Entdecken eines bestimmten Fehlertyps sind. Ein Experiment wird nun definiert, um diese Idee in eine quantifizierbare Hypothese zu überführen. Für das obige Beispiel denkbar:

Technik A ist in der Lage mehr Typ-1 Fehler zu entdecken als Technik B.

Dies ist eine quantifizierbare Hypothese in dem Sinne, dass sie messbar ist. Die Anzahl der Fehler wiederum kann auf verschiedene Art und Weise ermittelt werden. Die genaue Vorgehensweise wird in der Entwurfsphase festgelegt.

2.3.2.1 Ziele des Experiments definieren

Nach Basili [Bas95] eignet sich zur Zieldefinition das Goal-Question-Metric Goal Template [BaRo88]. Dieses Template wurde zur Definition von Messzielen beim Finden von Metriken entwickelt. Es bietet einen systematischen Ansatz zum Erstellen und Integrieren von Zielen, welche für Softwareprozesse, Produkte und Qualitätsaspekte von Relevanz sind bezogen auf die speziellen Anforderungen und Bedürfnisse des Projekts oder der Organisation. Von Bedeutung sind dabei nachfolgende Parameter:

Objekt der Studie

Gegenstand des Experiments, welches untersucht werden soll. Dabei kann es sich zum Beispiel um das Softwareprodukt, einen Entwicklungsprozess, Ressourcen oder ein Modell handeln.

Zweck der Studie

Definiert, was die Intention des Experiments sein soll, beispielsweise den Einfluss verschiedener Techniken zu evaluieren oder Effizienz von Prozessen zu bestimmen.

Fokus

Mit welcher Sichtweise untersucht wird, beispielsweise um Fehler zu finden

Perspektive

Die Perspektive stellt den Gesichtspunkt dar, unter dem das Experiment interpretiert wird, also die Personen, welche von den Informationen profitieren. Beispielsweise könnten Programmierer wohl weniger mit einer Interpretation des Experiments aus Sicht eines Managers anfangen, im Vergleich mit der Interpretation aus der Sicht eines Programmierers.

Domäne

Für Experimente im Bereich Software Engineering sind Programmierer oder Programmierteams sowie die Programme oder Projekte von besonderer Bedeutung. Erstere arbeiten getrennt voneinander und können Gruppen oder einzelne Personen sein, charakterisiert durch Größe, Erfahrung, etc., letztere sind die Programme oder Probleme, an denen erstere arbeiten, charakterisiert beispielsweise durch Komplexität, Größe oder Anwendung. Generell lässt sich der Gültigkeitsbereich durch die Größe der zwei Domänen ermitteln.

Einzelprojektstudien werden mit einem einzelnen Team oder einer Person, angewendet auf ein einzelnes Projekt, durchgeführt. Bei der Multiprojektstudie entsprechend mehrere Projekte mit einem einzelnen Team. Bearbeiten mehrere Teams unabhängig voneinander dasselbe Projekt spricht man von wiederholten Projekten, dadurch ergeben sich gute Ver-

gleichsmöglichkeiten, und die statistische Auswertbarkeit lässt sich verbessern. Ferner gibt es noch den Fall, dass mehrere Teams auf unterschiedliche Projekte angesetzt werden, beispielsweise um verschiedene Arbeitsmethoden vergleichen zu können. Dabei werden die einzelnen Teams voneinander abgegrenzt, so dass keinerlei Kommunikation unter ihnen möglich ist. All diese Experimenttypen können als Experiment oder Quasi-Experiment durchgeführt werden. Unterschieden wird dabei, ob bei der Einteilung der Personen Randomisierung angewendet oder eine gezielte Einteilung anhand spezieller Kriterien durchgeführt wird.

2.3.3 Planung / Entwurf

Nachdem in der Definitionsphase des Experiments das Fundament gelegt und erläutert wurde, aus welchen Gründen das Experiment durchzuführen ist, beschäftigt sich die Entwurfsphase damit, wie es durchzuführen ist. Diese Phase ist besonders wichtig, da das Ergebnis des Experiments verfälscht oder unbrauchbar gemacht werden kann, falls hierbei Fehler gemacht werden. Die Planungsphase besteht aus folgenden Unterbereichen: Umfeld festlegen, Hypothesen ausformulieren, Variablen bestimmen, Versuchsobjekte auswählen, Experiment designen und Gültigkeit der Experiments bestimmen.

Zuerst wird das Umfeld, in dem das Experiment ausgeführt werden soll, festgelegt. Findet das Experiment in einem normal laufenden Projekt statt oder alternativ als parallel laufendes offline Projekt, um die Risiken zu mindern, wofür aber mit zusätzlichen Kosten zu rechnen ist? Werden professionelle Mitarbeiter eingesetzt, oder wird das Experiment mit Studenten durchgeführt? Letzteres bietet eine günstige, leicht zu kontrollierende Möglichkeit, aber eher direkter auf einen speziellen Kontext bezogen, sowie schwerer in die Industrie zu überführen, verglichen mit dem Einsatz erfahrener, spezialisierter Mitarbeiter. Des Weiteren können mit letzteren, entsprechendes Budget vorausgesetzt, eher reale Probleme mit globaler Aussagekraft für den Software Engineering Bereich untersucht werden.

Als zweites werden die Hypothesen ausformuliert. Sie dienen als Basis für spätere statistische Auswertungen. Dabei werden die während des Experiments erhobenen Daten genutzt um, falls möglich, die Hypothese zu beurteilen und entsprechende Schlüsse zu ziehen. Dafür werden zwei Hypothesenarten aufgestellt. Die Nullhypothese und die alternative Hypothese. Erstere besagt, dass durch das Experiment keine Unterschiede im Vergleich zu der Effizienz des klassischen Vorgehens auftreten und falls doch, diese nur zufällig sind. Diese Hypothese gilt es zu widerlegen. Ein Beispiel einer Nullhypothese ist: *Die neue Methode der Fehlererkennung liefert im Vergleich zur momentan eingesetzten dieselbe Anzahl an Fehlern*. Alternative Hypothesen hingegen favorisieren was die Nullhypothese zurückweist. Um beim Beispiel zu bleiben: *Die neue Methode findet durchschnittlich mehr Fehler als die alte.*

Als nächstes werden die abhängigen und unabhängigen Variablen benannt. Unabhängige Variablen sind dabei die kontrollier- und änderbaren Variablen des Experiments, abhängige Variablen hingegen sollen direkt messbar und von der Hypothese ableitbar sein. Die Wahl der abhängigen Variablen legt außerdem Maßeinheit und Maßstab fest.

Anschließend werden die Versuchsobjekte ausgewählt, welche entscheidend für die Verallgemeinerbarkeit der Aussagekraft der Experimentergebnisse sind. Von Bedeutung ist dabei welchem Profil sie entsprechen müssen, wie hoch die benötigte Anzahl ist, sowie eventuelle Separierungskriterien.

Beim Gestalten des Experiments müssen die durchzuführenden Aufgaben sorgfältig geplant werden. Dabei von Bedeutung sind die Art und Anzahl der Tests, welche notwendig sind, um den beobachtenden Effekt sichtbar machen zu können, sowie in der Lage sind, einen Bezug zu den vorhandenen Hypothesen und Messgrößen herzustellen. Außerdem sollte die Aufgabe so angelegt werden, dass die Möglichkeit besteht, das Experiment wiederholen zu können. Weitere Designprinzipien, die es zu beachten gilt, sind Randomisierung, Blocken und Balancieren. Randomisieren beschreibt den Vorgang die Personen zufällig den Gruppen zuzuordnen. Blocken bedeutet eine klare Abgrenzung der Gruppen, so dass keine Interaktion unter ihnen stattfinden kann und Balancieren heißt, dass die gebildeten Gruppen die gleiche Personenstärke aufweisen. Besonders wichtig ist das Randomisieren, da es für statistische Auswertmethoden eine Vorraussetzung darstellt. Blocken wird angewendet, um teilweise auftretende bekannte Störfaktoren zu eliminieren, beispielsweise durch Aufteilen von erfahrenen und unerfahrenen Personen in zwei Gruppen mit klarer Abgrenzung untereinander durch bestimmte Anordnung des Arbeitsumfeldes. Durch Balancieren lässt sich die statistische Auswertung vereinfachen.

Abschließend bleibt noch die wichtige Frage, wie stichhaltig die Ergebnisse des Experiments sein werden. Dieser umfangreiche Aspekt sollte bereits in der Planung berücksichtigt werden [WRH+00].

2.3.4 Durchführung

Nachdem die ersten beiden Phasen des Experimentprozesses abgeschlossen wurden steht die Durchführung an, um die Daten zu erheben und auswerten zu können. Diese besteht aus drei Schritten, der Vorbereitungsphase, der Ausführung des Experiments und der Gültigkeitsprüfung der erhobenen Daten

Bevor das Experiment ausgeführt werden kann, müssen einige Vorbereitungen getroffen werden. Dazu gehört, die passenden Personen für das Experiment zu finden, welche motiviert und gewillt sind bis zum Ende teilzunehmen, sowie eventuelle Schulungen für sie abzuhalten. Die benötigten Materialien, Dokumente und Arten der Datenerhebung müssen vorbereitet, sowie das Arbeitsumfeld und Utensilien entsprechend der Planung eingerichtet werden. Falls beispielsweise Daten in Form eines Interviews erhoben werden, müssen die verschiedenen Fragen vor Experimentbeginn ausgearbeitet werden.

Nach der Vorbereitungsphase folgt die Durchführung des Experiments. An diesem Punkt kommen die Versuchspersonen mit dem Experiment direkt in Berührung und werden so beeinflusst. Es ist nun nicht mehr möglich, in die Definitions- oder Entwurfsphase zurückzukehren, um dort Änderungen vorzunehmen, da danach die Versuchspersonen nicht mehr eingesetzt werden können, um aussagekräftige Ergebnisse zu erhalten, weil sie bereits vom Experiment beeinflusst worden sind und sich entsprechend anders verhalten. In der Ausführung wird genau beschrieben, wie die einzelnen Schritte des Experiments ablaufen

sollen, und wie die Daten erhoben werden. Letzteres kann manuell durch Ausfüllen von Fragebögen durch die Teilnehmenden des Experiments oder automatisiert durch Werkzeuge erfolgen.

Im letzen Schritt der Durchführung werden die gesammelten Daten auf ihre Gültigkeit geprüft. Dabei wird betrachtet, ob sich die Messwerte in den vorgegebenen Bereichen befinden, ob die Teilnehmer die Fragebögen richtig verstanden und ernsthaft ausgefüllt haben und das Experiment entsprechend der Planung korrekt durchgeführt wurde. Falls einige Teilnehmer nicht ernsthaft mitgearbeitet haben, sollten diese Daten vor der Auswertung entfernt werden, so dass keine unbrauchbaren Daten das Ergebnis nachteilig beeinflussen, beziehungsweise verfälschen.

2.3.5 Analyse / Interpretation

Das Ziel dieser Phase ist es, die gesammelten Daten auszuwerten und daraus Schlussfolgerungen zu ziehen. Dazu müssen die Experimentdaten entsprechend interpretiert werden. Im ersten Schritt werden die Daten charakterisiert und grafisch aufbereitet, um interessante Aspekte sichtbar zu machen. Dies erfolgt beispielsweise über Streuungs- oder Kuchendiagramme, um eine Übersicht für die Daten zu bekommen, sie besser verstehen zu können und anormale oder falsche Messwerte zu identifizieren.

Bevor die Daten zum Überprüfen der Hypothesen eingesetzt werden können, folgt die Untersuchung auf Fehler. Dabei kann es sich um falsche Messwerte oder um Ausreißer handeln. Im Abschluss der Durchführungsphase wurden bereits fehlerhafte oder unbrauchbare Daten, zum Beispiel erzeugt durch fehlende Ernsthaftigkeit einiger Teilnehmer, identifiziert und entfernt, bleiben noch die Ausreißermesswerte übrig. Hier muss ergründet werden, warum es zu diesen Werten kam. Falls es sich durch ein ungewöhnliches oder seltenes Ereignis, welches sich nicht wiederholt, handelt, kann der Messwert herausgenommen werden. Falls sich aber das Ereignis, welches zu dem Wert geführt hat, wiederholen kann, ist es nicht sinnvoll, den Punkt herauszunehmen, sondern stattdessen sollte überlegt werden, wie mit ihm zu verfahren ist. Außer den fehlerhaften oder anormalen Daten sollten noch die redundanten betrachtet und vor dem Hypothesentest entsprechend bearbeitet werden.

Die Aufgabe des Hypothesentests ist es herauszufinden, ob es möglich ist, die Nullhypothese zu widerlegen, da andernfalls keine Schlussfolgerungen gezogen werden können. Die Nullhypothese hängt von einer abhängigen Variablen ab, welche im Experiment direkt gemessen wurde und über die eine Aussage getroffen werden kann. Es gibt eine Vielzahl statistischer Tests, die angewendet werden können, entsprechend dem Experimentdesigns. Sie lassen sich in zwei Bereiche unterteilen, parametrisierte und parameterlose Tests. Letztere sind allgemeiner und können, falls erstere nicht anwendbar sind, stattdessen eingesetzt werden. Bekanntere Beispiele für parameterlose Tests sind der Chi^2- oder der Binomial Test, für den anderen Bereich t-Test oder F-Test. Einen Überblick über weitere Tests sowie die Funktionsweise der einzelnen Tests findet sich unter [WRH+00]. Für die einzusetzenden Tests ist ferner die Eignung zu prüfen, also ob sie überhaupt anwendbar sind. Dabei von Bedeutung sind die Normalverteilung, Abhängigkeit und der statisti-

sche Fehler. Falls beispielsweise die Testdaten von mehreren unabhängigen Variablen stammen ist es notwendig zu prüfen, dass keine Korrelation zwischen den einzelnen Variablen besteht.

Nachdem die Experimentdaten analysiert und interpretiert wurden, werden Schlussfolgerungen gezogen. Falls die Nullhypothese nicht widerlegt werden kann, liegt kein kausaler Zusammenhang zwischen den abhängigen und unabhängigen Variablen vor. Es wurde lediglich gezeigt, dass kein signifikanter statistischer Unterschied zwischen den einzelnen untersuchten Verfahren besteht. Dies wäre beispielsweise ein Ergebnis der Art, dass Methode B 1% Kosten sparend gegenüber Methode A ist.

Falls aber die Nullhypothese widerlegt werden konnte, lassen sich Schlussfolgerungen über den Einfluss der Variablen ziehen. Beachtet werden sollte dabei auch die Möglichkeit einer nicht beachteten Variablen, welche für den kausalen Zusammenhang verantwortlich ist. In allen Fällen muss die praxisnahe Bedeutung untersucht werden, sowie sämtliche Aspekte, die eine Wiederholung erleichtern, beschrieben werden.

2.4 Beispiel für ein Experiment

Im Wintersemester 2004/2005 wurde an der Universität Hannover im Fachgebiet Software Engineering ein Experiment mit Studenten von T. Flohr und T. Schneider durchgeführt. [FlSc04] Dabei wurden zwei Entwicklungsmethoden auf ein Problem angewandt, um festzustellen, welche der beiden effizienter ist. Untersucht wurde dabei Test-First und klassisches Testen. Dieses Experiment, im weiteren als Test-First Experiment bezeichnet, erfüllt im Wesentlichen den im Kapitel 2.3 vorgestellten Experimentprozess und soll als Beispiel eines Experiments aus dem Bereich Software Engineering dienen. Die Experimentfrage lautet:

> *„Welche Vorteile bietet Test First gegenüber einem Szenario, in dem zwar nicht zuerst die Testfälle geschrieben werden, aber eine automatisierte Testausführung möglich ist?“*

In der Planungsphase wurden mehrere Hypothesen aufgestellt, von denen letztendlich vier untersucht und ausgewertet wurden. Sie lauten:

- Die Entwicklung nach dem Test-First Verfahren produziert mehr Testfälle als durch klassisches Testen.
- Die Testabdeckung liegt bei Test-First zwischen 90 und 100%, also deutlich höher als beim klassischen Programmieren.
- Die Entwicklungsgeschwindigkeit beim klassischen Testen liegt höher, verglichen mit der Entwicklung nach Test-First, da beim klassischen Testen weniger Zeit für Testfälle aufgebracht wird.
- Die Akzeptanz der Studenten für das Test-First Verfahren lässt sich in einem Kurvenverlauf darstellen (siehe hierzu Abbildung 1 aus [FlSc04])

Für jede Hypothese wurden mittels der GQM Methode Metriken entwickelt. Für das Experiment wurden 19 Studenten eingesetzt, welche zuvor durch Fragebögen auf ihre Eignung überprüft wurden. Um die Unterschiede der zwei Entwicklungsverfahren zu bestimmen, wurden zwei Gruppen gebildet, eine Kontrollgruppe, welche nach herkömmlicher Art programmiert, sowie eine Testgruppe die nach dem Test-First Verfahren entwickelt. Dabei wurde auf Balancierte Gruppen geachtet, welche mittels Randomisierung zugewiesen wurden. Außerdem wurde jede Gruppe noch einmal in Untergruppen zu zwei Personen eingeteilt, entsprechend zugewiesen per Randomisierung. Durch diese Zweiergruppen konnte Extreme Programming [Bec00] eingesetzt werden. Beide Gruppen erhielten die gleiche Aufgabe, das Implementieren einer Javabibliothek zur Verwaltung und Analyse der semantischen Beschreibungssprache FLOW innerhalb von zehn Wochen, wobei diese noch in Teilaufgaben anhand von Story Cards unterteilt wurden.

In der Planungsphase wurden Störvariablen identifiziert und das Arbeitsumfeld entsprechend gestaltet. Die Entwicklungsumgebung wurde mit Eclipse festgelegt, sämtliche Materialien wurden nur während der Experimentlaufzeit ausgegeben, die Computer vom Netz getrennt, feste Programmierzeiten vorgegeben mit dem Verbot, außerhalb dieser Zeiten weiterzuarbeiten, sowie ein Kommunikationsverbot innerhalb der Studenten erlassen. Für die Test-First Gruppe sowie die nach klassischem Testen entwickelnde Gruppe wurden spezielle Schulungen durchgeführt, wobei die jeweiligen Entwicklungsverfahren erläutert wurden. Als Art der Datenerhebung wurden Logfiles eingesetzt, welche während des gesamten Experiments gesammelt wurden, sowie Fragebögen eingesetzt und die Entwicklungszeit der jeweiligen Teilaufgaben gemessen. Während des gesamten Entwicklungsprozesses übernahm jeweils einer der Experimentatoren die Aufgabe des On-site Customers sowie des technischen Beraters. Diese beiden Ansprechpartner standen den Versuchspersonen während der Experimentlaufzeit zur Verfügung.

Die Datenauswertung erfolgte in zwei Phasen. Ein Teil der erhobenen Daten wurde unmittelbar nach der Experimentdurchführung ausgewertet, um eine erste Beurteilung der Hypothesen durchzuführen. Da diese Bewertung noch während des Semesters mit den Studenten erörtert werden sollte, war der verfügbare Zeitrahmen zu klein, um eine komplette Datenauswertung durchzuführen. Bislang ist die Auswertung der erhobenen Daten noch nicht abgeschlossen.

Jedoch wurde bereits Paper veröffentlicht, in dem der Experimentaufbau sowie das Experimentdesign beschrieben wurden. Eine ausführliche Bewertung der Hypothesen soll nach vollständiger Datenauswertung folgen.

3 Analyse von Experimenten

Eine Analyse (griechisch ανάλυση, vom altgriechischen Verb ἀναλύειν, analyein – auflösen) ist eine systematische Untersuchung, bei der das untersuchte Objekt oder Subjekt zergliedert und in seine Bestandteile zerlegt wird und diese anschließend geordnet und ausgewertet werden.

(Definition nach Wikipedia)

3.1 Mathematische Betrachtung

In vielen Anwendungsgebieten existieren Datenbestände, die komplexe Sachverhalte und Objekte beschreiben, wie beispielsweise bei dem in dieser Arbeit betrachteten Feld der Experimente. Gesucht sind Methoden, die es erlauben, strukturelle Gemeinsamkeiten solcher komplexen Objekte zu finden.

Experimente werden in dieser Arbeit als markierte, gerichtete Graphen dargestellt, genauer erläutert in Kapitel 4.2. Die Repräsentation komplexer Objekte durch markierte Graphen ist in vielen Anwendungsgebieten üblich und sinnvoll, wie z.B. in der Chemie. Sie zeichnen sich dadurch aus, komplexe Sachverhalte in ihrer vollen Struktur komprimiert und verständlich darzustellen.

Problematisch bei der Wahl von Graphen zur Repräsentation von Strukturen in der Anwendung ist bisher, dass wichtige Begriffe, wie Abstand oder Ähnlichkeit, die für logische Repräsentationen definiert sind, keine einheitlich gebrauchte Entsprechung auf dem Gebiet der markierten Graphen haben, sowie darauf aufbauende Verfahren zum Klassifizieren für Graphen weniger verbreitet sind.

3.1.1 Ähnlichkeit

Die Ermittlung struktureller Gemeinsamkeiten und Ähnlichkeiten ist die Grundlage für eine Reihe von Methoden, die das enthaltene Wissen über die Zusammenhänge von Strukturen und Verhalten bzw. Funktion der Objekte nutzbar machen sollen. Das Isomorphieproblem, die Möglichkeit für ein und dieselbe Struktur mehrere verschiedene

Darstellungen anzugeben, ist bei der Verarbeitung von Strukturen von besonderer Bedeutung. Isomorphe Graphen besitzen dieselben Eigenschaften, also dieselbe Anzahl von Knoten, Kanten, sowie Knoten eines bestimmten Grades. Die Ermittlung struktureller Übereinstimmungen und Ähnlichkeiten ist das Kernstück vieler Verfahren zur Verarbeitung struktureller Beschreibungen. Die strukturellen Gemeinsamkeiten von Objekten entsprechen in der Graphenrepräsentation isomorphe Subgraphen, also übereinstimmenden Teilen von Graphen in Hinblick auf gleiche Knoten. Die Ermittlung größter isomorpher Subgraphen ist im Allgemeinen ein NP-vollständiges Problem.

In[TSS+95] werden Ansätze zum analogen Schließen vorgestellt, die alle auf der Suche nach gemeinsamen Untergraphen zwischen den betrachteten Strukturen beruhen, wobei meist das Problem der Subgraphensuche, besonders für allgemeine und große Graphen, in adäquater Zeit ungelöst bleibt. Im fallbasierten und analogen Schließen wird Wissen über bekannte Probleme auf strukturell ähnliche neue Aufgaben, unter Ausnutzung struktureller Übereinstimmungen, übertragen. Dabei werden Fälle durch komplexe Strukturen, beispielsweise markierte Graphen, beschrieben. Strukturelle Übereinstimmung zwischen Problemen oder Problemlösungen sind die Basis. [Wir98]

Knoten- und kantenmarkierte Graphen bilden eine verbreitete, intuitive Repräsentation strukturierter Objekte, ausgezeichnet dadurch, dass jedem Objekt genau ein Knoten und jedem Paar von Knoten genau eine Kante zugeordnet werden kann. Ein gemeinsamer oder isomorpher Subgraph zweier interpretierter Graphen ist ein Graph, der isomorph zu je einem Subgraphen beider Graphen ist, und der größte gemeinsame Subgraph bezeichnet denjenigen mit der größten gemeinsamen Knotenanzahl. Graphen werden als umso ähnlicher angesehen, je größer die ihnen gemeinsamen Subgraphen sind. Strukturelle Ähnlichkeit einer Menge von Graphen wird in [BaTa94] definiert als deren knoten- oder kantenmaximaler isomorpher Subgraph. Quantitatives Maß dabei ist die Zahl der Knoten, beziehungsweise Kanten des Subgraphen. Die Bestimmung der Ähnlichkeit von Graphen über gemeinsame bzw. unterschiedliche Subgraphen ist jedoch sehr aufwändig wie beispielsweise [Knö71] zeigt.

3.1.2 Abstandsmaß

Die Ähnlichkeit von Strukturen lässt sich beispielsweise durch den Abstand quantifizierbar ausdrücken. Ein Maß des Abstands bei Strukturen, beruhend auf der Ermittlung größter gemeinsamer Subgraphen, bietet der Zelinka-Abstand. In [Zel75] definiert Zelinka eine Metrik über die Menge aller Isomorphieklassen von unmarkierten Graphen mit gleicher Knotenzahl. Die Graphen können dabei gerichtet oder ungerichtet sein, dürfen aber keine multiplen Kanten oder Schleifen besitzen. Die Definition zur Berechnung lautet folgendermaßen:

> *Seien G und H zwei Graphen ohne Schlingen und multiple Kanten mit jeweils $|N|$ Knoten, wobei $|N_T|$ die Knotenzahl des größten gemeinsamen induzierten Teilgraphen von G und H bezeichnet. Durch $|N| - |N_T|$ wird eine Metrik im Raum der Isomorphieklassen von Graphen mit $|N|$ Knoten definiert.*

Die Definition des Zelinka-Abstandes wurde von Kaden in [KaSo82] auf allgemeine Graphen beliebiger Größe erweitert. Sie lautet:

> *Seien G und H zwei beliebige Graphen mit $|N_G|$ beziehungsweise $|N_H|$ Knoten und $|N_T|$ die Anzahl der Knoten des größten isomorphen Teilgraphen von G und H.*
>
> *Durch $max\{|N_G|, |N_H|\} - |N_T|$ wird eine Metrik im Raum der Isomorphieklassen von Graphen definiert.*

Der erweiterte Zelinka-Abstand kann für die, in dieser Arbeit betrachteten Experimentgraphen und deren Analyse eingesetzt werden, da die geforderten Bedingungen erfüllt werden. Die Metrik gibt ein Maß für die Ähnlichkeit zweier Graphen an, wobei der erweiterte Zelinka-Abstand als Ergebnis Null liefert, wenn die betrachteten Graphen identisch sind. Je größer der Wert ist, desto verschiedener sind sie zueinander.

Bisher wurden nur Knoten und Kanten gleicher Markierung betrachtet, die Paare der aufeinander abgebildeten tragen dann gleichermaßen zur Ähnlichkeit bei. Notwendig ist es nun, die Güte der Ähnlichkeit zu quantifizieren, die nicht nur gleich, sondern auch unterschiedlich markierte Knoten und Kanten aufeinander abbildet. Nach [WoYo85] wird dies erreicht, indem bei der Bewertung des Abstandes bzw. der Ähnlichkeit Substitutionskosten für die Überführung der einen Markierung in die andere berücksichtigt werden.

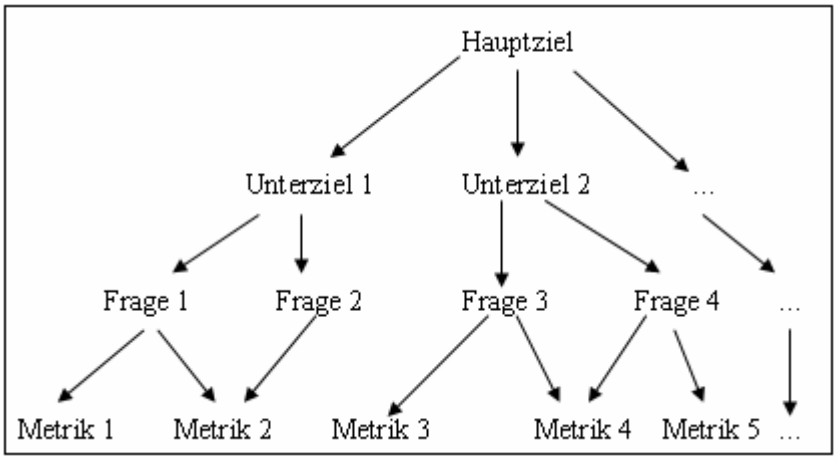
Eine andere Möglichkeit besteht darin, die Ähnlichkeit der Knoten und Kanten von Anfang an in einem Term zu verrechnen, also eine kombinierte Ähnlichkeit der Kanten mit ihren beteiligten Knoten zu bestimmen. Da die Bewertung von Ähnlichkeiten abhängig vom Zweck der Ähnlichkeitsbetrachtung ist, können problemabhängige Gewichtungen und Kosten für Knoten- und Kantenmerkmale über die Definition der Knoten- und Kantenähnlichkeiten die Bewertung der Graphähnlichkeiten beeinflussen. Im Verfahren der Multidimensionalen Skalierung [BoLi87] wird versucht, ausgehend von einer Matrix, die den Abstand bzw. die Ähnlichkeit zwischen jeweils zwei Objekten enthält, die Objekte in einen Vektorraum einzubetten und Zusammenhänge zwischen Bereichen dieser Räume zu formulieren. Dieser Ansatz ist schwer zu realisieren, und es war nicht möglich, für die in dieser Arbeit betrachteten Experimente eine Ähnlichkeitsmatrix aufzustellen. Es wird sehr viel Zeit und Erfahrung im Umgang mit Experimenten im Bereich Software Engineering benötigt, um eine solche Matrix aufzustellen, so dass im Rahmen dieser Arbeit auf eine Umsetzung verzichtet wurde.

Eine weitere Möglichkeit, zusammengehörige Teilmengen von Objekten zu repräsentieren, besteht in der Definition von Konzepten, auch Begriffe genannt [Sch00]. Ein Begriff fasst eine Menge von Objekten mit gemeinsamen Merkmalen unter einem Namen zusammen. Konzeptbeschreibungen charakterisieren die Gemeinsamkeiten der zugehörigen Objekte. Sie sind also generalisierte, abstrahierende Beschreibungen. Wichtig hierbei ist, dass zu einer Konzept-Beschreibungssprache eine Entscheidungsfunktion gegeben ist, so dass für jedes Objekt und jede Konzeptbeschreibung entschieden werden kann, ob das Objekt zu der von der Konzeptbeschreibung definierten Menge von Objekten gehört. Problematisch für die Umsetzung dieser Möglichkeit ist das Erstellen der Entscheidungsfunktion, welche für jedes Objekt eine genaue Zuordnung zu den Begriffen liefert. Auch die

Einbeziehung dieses Punktes war in dem zeitlichen Rahmen dieser Arbeit nicht möglich gewesen.

3.2 Goal Question Metric Methode

Um Metriken zu entwickeln, die die Analyse und Charakterisierung von Experimenten betreffen, bedient man sich im Software Engineering der Goal Question Metric Methode [BCR94]. Die GQM Methode ist eine systematische Vorgehensweise zur Erstellung spezifischer Qualitätsmodelle. Der Vorteil dieser Methode besteht darin, dass die Messziele aus den Projektzielen abgeleitet werden. Unnütze Messungen ohne Zielbezug entfallen vollständig. Die GQM Metrik lässt sich grafisch als Baum darstellen, wobei das Ziel (*Goal*) als Wurzel, die Knoten die Fragen (*Questions*) und die Blätter die Metriken darstellen. Das Ziel kann dabei in mehrere Unterziele unterteilt werden, welche über die Fragen zu den Metriken verfeinert werden, wobei auf diesem Weg Softwaremetriken abgeleitet werden. Nachfolgende Abbildung veranschaulicht diesen Vorgang.



(Abbildung 3 : Schematischer Ablauf der GQM Methode)

Mit Hilfe der GQM Methode lassen sich für spezifische Ziele über die dazugehörigen Fragen Metriken ableiten, mit denen eine Bewertung möglich ist. Der Begriff der Software-Einheit bezeichnet dabei nicht nur den Quellcode, sondern vielmehr fallen darunter auch Prozesse, Dokumente, Eigenschaften der Entwickler (z.B. Erfahrung) etc. Unter letztere fällt auch der Experimentgraph, was eine Anwendung von Softwaremetriken auf die in dieser Arbeit betrachtete Repräsentation von Experimenten durch den Experimentgraph rechtfertigt. Eine Softwaremetrik ist nach IEEE Standard 1061[IEE98] folgendermaßen definiert:

> *Eine Softwaremetrik ist eine Funktion, die eine Software-Einheit in einen Zahlenwert abbildet. Dieser berechnete Wert ist interpretierbar als der Erfüllungsgrad einer Qualitätseigenschaft der Software-Einheit.*

Das Ziel, zu dem Metriken gefunden werden sollen lautet:

> *„Möglichkeiten der Charakterisierung und Analyse von Experimenten im Software Engineering“*

Dabei ergaben sich mehrere Unterziele, für die Fragen formuliert und Metriken abgeleitet wurden. Sie bilden eine Hilfestellung für Charakterisierungs- und Analyseszenarien. Es folgt eine genaue Betrachtung der Unterziele und die Anwendung des GQM Verfahrens.

3.3 Metriken zur Analyse von Experimenten

Für die Ähnlichkeitsbestimmung von Experimenten wurde unter anderem der zuvor erläuterte erweiterte Zelinka-Abstand herangezogen.

3.3.1 Ähnlichkeitsbestimmung von zwei Experimenten

Ziel: Ähnlichkeit von zwei Experimenten bestimmen

Fragen:

Wie viele Arbeitsschritte enthalten die Experimente?

Wie viele identische Arbeitsschritte enthalten die Experimente?

Wie viele Hypothesen müssen ausgewertet werden?

Wie viele Versuchspersonen nehmen an den Experimenten teil?

Metriken:

1.1 Anzahl der markierten Knoten im Experiment

1.2 Anzahl der identischen, markierten Knoten von zwei Experimenten

1.3 Bestimmung des erweiterten Zelinka-Abstands

1.4 Anzahl der Hypothesen pro Experiment

1.5 Anzahl der Versuchspersonen pro Experiment

Erläuterung der einzelnen Metriken:

Metrik 1.1 (Wie viele Arbeitsschritte enthalten die Experimente?)

Beschreibung: Es werden alle Knoten der zwei zu vergleichenden Experimente gezählt und als eine Zahl dargestellt.

Berechnung: Von jedem Experiment wird die Anzahl der Knoten durch die Länge der Knotenliste, in der jeder Knoten des Experiments gespeichert ist, ermittelt.

Mögliche Werte: Der kleinste angenommene Wert ist Null und wird nach oben begrenzt durch die maximal erreichbare Anzahl von Knoten in einem Experiment.

Einschränkungen: Es müssen mindestens zwei Experimente für einen Vergleich vorhanden sind.

Metrik 1.2 (Wie viele identische Arbeitsschritte enthalten die Experimente?)

Beschreibung: Es wird die Anzahl identischer Knoten der zwei zu vergleichenden Experimenten ermittelt und als eine Zahl dargestellt.

Berechnung: Jeden Knoten des einen Experiments wird mit den Knoten des zweiten Experiments verglichen und die Übereinstimmungen gezählt.

Mögliche Werte: Der kleinste angenommene Wert ist Null und wird nach oben begrenzt durch die maximal erreichbare Anzahl von Knoten in einem Experiment.

Einschränkungen: Es müssen mindestens zwei Experimente für einen Vergleich vorhanden sind.

Metrik 1.3 (Wie viele identische Arbeitsschritte enthalten die Experimente?)

Beschreibung: Als ein Maß für die Ähnlichkeit zweier Experimente wird deren Abstand anhand des erweiterten Zelinka-Abstands ermittelt. Je geringer der ermittelte Wert ist, desto größer ist die Ähnlichkeit der zwei Experimente.

Berechnung: Der erweiterte Zelinka-Abstand wird durch folgende Formel ermittelt: max{# Knoten Experiment 1, # Knoten Experiment 2} – # identischer Knoten

Mögliche Werte: Der kleinste angenommene Wert ist Null und wird nach oben begrenzt durch die maximal erreichbare Anzahl von Knoten in einem Experiment.

Einschränkungen: Es müssen mindestens zwei Experimente für einen Vergleich vorhanden sind.

Metrik 1.4 (Wie viele Hypothesen müssen ausgewertet werden?)

Beschreibung: Es wird die Anzahl der auszuwertenden Hypothesen ermittelt und als eine Zahl dargestellt.

Berechnung: Die Hypothesen an dem entsprechenden Knoten werden gezählt.

Mögliche Werte: Der kleinste angenommene Wert ist eins und ist nach oben hin offen. Aus dem Test-First Experiment ging hervor, dass von neun aufgestellten Hypothesen nur vier ausgewertet wurden, weshalb eine sinnvolle Obergrenze beispielsweise mit 10 oder 15 angesetzt werden könnte.

Einschränkungen: Die Hypothesen müssen für das Experiment angegeben sein.

Metrik 1.5 (Wie viele Versuchspersonen nehmen an den Experimenten teil?)

Beschreibung: Es wird die Anzahl der Versuchspersonen in dem Experiment ermittelt und als eine Zahl dargestellt.

Berechnung: Die eingegebene Anzahl der Versuchspersonen des Experiments wird anhand der entsprechend angelegten Variable ausgelesen.

Mögliche Werte: Der kleinste angenommene Wert ist eins und nach oben hin unbeschränkt.

Einschränkungen: Die die Versuchspersonen für das Experiment müssen angegeben sein.

3.4 Metriken zur Charakterisierung von Experimenten

Neben der Analyse beschäftigt sich diese Arbeit mit der Charakterisierung von Experimenten im Bereich Software Engineering. Demnach stellt sich die Frage, durch welche Eigenschaften sich Experimente charakterisieren lassen. Während sich bei Softwareprodukten beispielsweise der Sourcecode in Klassen, Methoden, Funktionen etc. zerlegen lässt, ist diese Zerlegung bei Experimenten nur schwer möglich. Denkbare Komponenten, die in allen Experimenten auftauchen und zum Vergleich herangezogen werden können, sind beispielsweise:

- Templates
- Bearbeitete eigene Dokumente
- Hypothesen
- Versuchspersonen

3.4.1 Abschließbarkeit von Experimenten

Ziel: Experiment abschließen

Fragen:

Wurde ein Template bearbeitet?

Wurden sämtliche Templates eines Knoten bearbeitet?

Wurden alle Knoten des Experiments vollständig bearbeitet?

Endet das Experiment mit dem Experimentendknoten?

Ist das Experiment abschließbar?

Metriken:

2.1 Bearbeitungszustand der Templates

2.2 Bearbeitungszustand der Knoten

2.3 Bearbeitungszustand des Experiments

2.4 Endknotenzustand

2.5 Abschließbarkeit des Experiments

Erläuterung der einzelnen Metriken:

Metrik 2.1 (Wurde ein Template bearbeitet?)

Beschreibung: Es wird geprüft ob ein Template bearbeitet wurde.

Berechnung: Zu einem Template wird geprüft, ob ein zugehöriges, bearbeitetes Dokument hochgeladen wurde. Trifft dies zu, wird der Wert *wahr*, andernfalls der Wert *falsch* für den Bearbeitungszustand des Templates angenommen.

Mögliche Werte: Als Werte können *wahr* oder *falsch* angenommen werden.

Einschränkungen: Das aktuelle Experiment muss mindestens ein Knoten mit einem Template enthalten.

Metrik 2.2 (Wurden sämtliche Templates eines Knoten bearbeitet?)

Beschreibung: Für sämtliche Templates eines Knotens wird mittels Metrik 2.1 der Bearbeitungszustand geprüft.

Berechnung: Zu einem Knoten wird mittels Metrik 2.1 für jedes Template der Bearbeitungszustand geprüft. Falls bei allen Templates der Wert *wahr* ausgegeben wurde, wird der Bearbeitungszustand des Knotens auf *wahr* gesetzt, andernfalls wird der Wert *falsch* angenommen.

Mögliche Werte: Als Werte können *wahr* oder *falsch* angenommen werden.

Einschränkungen: Das aktuelle Experiment muss mindestens einen Knoten enthalten.

Metrik 2.3 (Wurden alle Knoten des Experiments vollständig bearbeitet?)

Beschreibung: Es wird für sämtliche Knoten des Experiments mittels Metrik 2.2 der Bearbeitungszustand ermittelt.

Berechnung: Zu jedem Knoten des Experiments wird mittels Metrik 2.2 der Bearbeitungszustand ermittelt. Falls bei allen Knoten der Wert *wahr* ausgegeben wurde, wird der Bearbeitungszustand des Experiments auf *wahr* gesetzt, andernfalls wird der Wert *falsch* angenommen.

Mögliche Werte: Als Werte können *wahr* oder *falsch* angenommen werden.

Einschränkungen: Das aktuelle Experiment muss mindestens einen Knoten enthalten.

Metrik 2.4 (Endet das Experiment mit dem Experimentendknoten?)

Beschreibung: Es wird ermittelt, ob der letzte Knoten des Experiments der Experimentendknoten ist.

Berechnung: Der letzte Knoten des Experiments wird überprüft. Handelt es sich dabei um den speziellen Experimentendknoten, wird als Wert *wahr* ausgegeben, andernfalls wird der Wert *falsch* angenommen.

Mögliche Werte: Als Werte können *wahr* oder *falsch* angenommen werden.

Einschränkungen: Das aktuelle Experiment muss mindestens einen Knoten enthalten.

Metrik 2.5 (Ist das Experiment abschließbar?)

Beschreibung: Es wird geprüft, ob sich das Experiment abschließen lässt.

Berechnung: Metriken 2.3 und 2.4 werden auf das Experiment angewandt. Liefern beide den Wert *wahr*, so wird die Abschließbarkeit des Experiments auf *wahr* gesetzt, andernfalls wird der Wert *falsch* angenommen.

Mögliche Werte: Als Werte können *wahr* oder *falsch* angenommen werden.

Einschränkungen: Das aktuelle Experiment muss mindestens einen Knoten enthalten.

3.4.2 Standardmaß von Experimenten

Ziel: Experiment anhand der ausgewählten Knoten charakterisieren

Fragen:

Wie viel verschiedene Pfade innerhalb des Experimentgraphen können gebildet werden?

Wie hoch ist der Anteil der Standardkomponenten im Experiment?

Metriken:

3.1 Alle möglichen Pfade ermitteln

3.2 Standardmaß ermitteln

Erläuterung der einzelnen Metriken:

Metrik 3.1 (Wie viel verschiedene Pfade innerhalb des Experimentgraphen können gebildet werden?)

Beschreibung: Es wird die Anzahl aller möglichen Pfade, die von Experimenten angenommen werden können, ermittelt.

Berechnung: Zur Berechnung wird eine Tiefensuche verwendet, die von der Wurzel des Experimentgraphen ausgehend den Wert um eins erhöht, wenn sie an dem Endknoten angekommen ist.

Mögliche Werte: Solange der Experimentgraph nicht geändert wird, wird durch diese Metrik immer derselbe Wert erzeugt, welcher abhängig vom Inhalt des Experimentgraphs ist.

Einschränkungen: Es muss ein gültiger Experimentgraph vorhanden sein.

Metrik 3.2 (Wie hoch ist der Anteil der Standardkomponenten im Experiment?)

Beschreibung: Unter der Annahme, dass jeder Pfad im Experimentgraph gleich häufig verwendet wird, ergibt sich, dass Knoten, die auf sehr vielen Pfaden liegen, sehr häufig in Experimenten vorkommen und entsprechend Standardkomponenten eines Experiments sind. Darauf aufbauend ist ein Experiment ein Standardexperiment, wenn es Standardkomponenten enthält. Diese Metrik berechnet den Anteil der Standardkomponenten im aktuellen Experiment und gibt dieses Standardmaß als Prozentzahl aus.

Berechnung: Anhand Metrik 3.1 wird die Anzahl der möglichen Experimentpfade ermittelt. Anschließend wird eine Tabelle angelegt, in der sämtliche Knoten gelistet sind mit der Anzahl ihres Auftretens in allen Experimenten. Dieser Wert wird normalisiert. Für das aktuelle Experiment wird für jeden Knoten der Wert aus der Tabelle ausgelesen, addiert und durch die Anzahl der Knoten des aktuellen Experiments (ermittelt durch Metrik 1.1) dividiert und ergibt somit das Standardmaß des Experiments.

Mögliche Werte: Der theoretisch kleinste annehmbare Wert ist Null, der größte hundert, da das Standardmaß als Prozentanteil ausgedrückt wird.

Einschränkungen: Das aktuelle Experiment muss mindestens einen Knoten enthalten.

4 Realisierung des Experimentplanungswerkzeugs

Das im Wintersemester 2004/2005 durchgeführte Test-First Experiment an der Universität Hannover des Fachgebiets Software Engineering war Auslöser für die Idee dieses Werkzeugs. Folglich stammen die Anforderungen auch hauptsächlich von den an dem Experiment beteiligten Mitarbeitern. Ihre Erfahrungen bei der Planung, Durchführung und Auswertung spielten eine wichtige Rolle bei der Anforderungserhebung. In den ersten Wochen dieser Arbeit wurde durch mehrere Gespräche sowie Durchsicht der damals angefallenen Dokumente und Berichte versucht, die grundlegenden Funktionalitäten, die das Werkzeug leisten soll, herauszuarbeiten. Das durch intensives Einarbeiten in die Domäne der Experimente im Software Engineering erhaltene Wissen wurde genutzt, um die Anforderungen um eigene Ideen zu ergänzen. In der Entwicklungsphase wurden die Anforderungen iterativ konkretisiert.

4.1 Anforderungen

Der folgende Abschnitt betrachtet die verschiedenen Anforderungen, unterteilt in technische, funktionale sowie gestellte Anforderungen an die Benutzeroberfläche. Eine Beschreibung in Textform dient der Übersicht, gefolgt von der detaillierten Erläuterung einiger grundlegender funktionaler Anforderungen in Form von Use Cases.

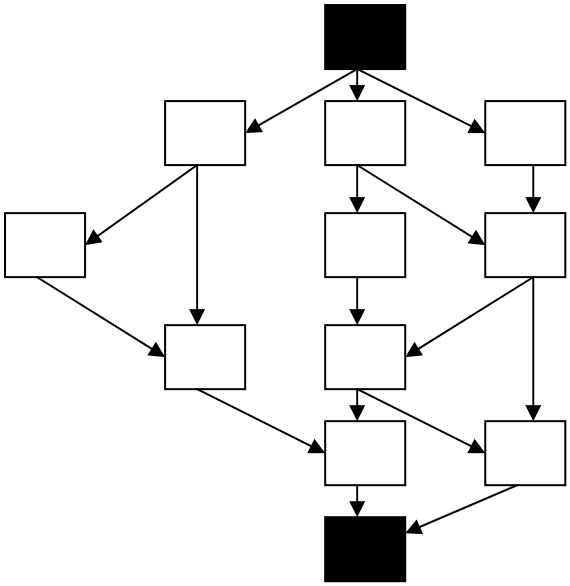
4.1.1 Technische, allgemeine Anforderungen

Erstellt werden soll eine Java Applikation, mit der Anwender in der Lage sind, Experimente planen, verwalten und analysieren zu können. Ein Experimentgraph dient dabei als ein Gerüst, in dem sich durch Markieren von Knoten ein spezifisches Experiment anhand eines eindeutigen Pfades darstellen lässt. Die Knoten stellen dabei Arbeitsschritte dar, welche durchgeführt werden müssen. Zur besseren Veranschaulichung folgt ein kleiner Auszug von Arbeitsschritten aus dem Test-First Experiment:

- Experimentfrage definieren
- Aufstellen der Hypothesen

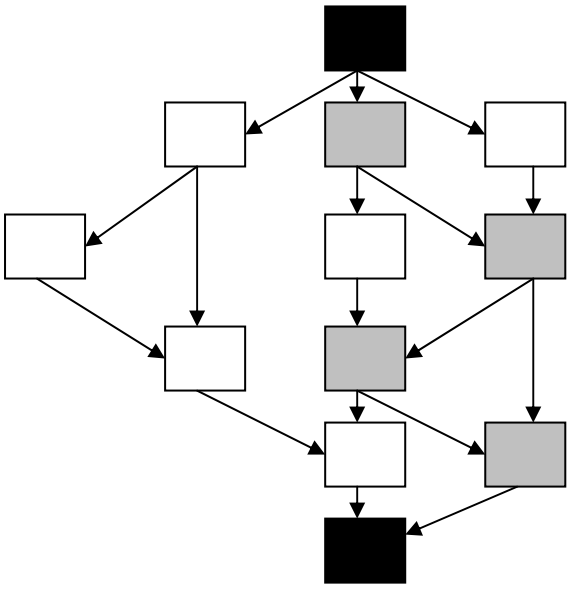
- Anzahl der teilnehmenden Versuchspersonen festlegen (dargestellt durch mehrere Intervalle, die jeweils durch einen Knoten dargestellt werden. Es kann nur ein Knoten, bzw. Intervall gleichzeitig gewählt werden, beispielhaft durch drei Knoten, weniger als 10 Personen, zwischen 10 und 30 Personen und mehr als 30 Personen, repräsentiert)
- Die Einteilung der Testpersonen in Gruppen, dabei unterschieden in ausschließlich Testgruppe oder Testgruppe und Kontrollgruppe. Dargestellt anhand von zwei Knoten, von denen jeweils nur einer selektiert werden kann.
- Art der Datenerhebung (dargestellt durch mehrere Knoten, wobei jeder Erhebungsart (z.B. per Logfiles, Fragebögen oder durch Messen) ein Knoten entspricht, welche in beliebiger Kombination ausgewählt werden können.)

Es existieren für jeden Arbeitsschritt mehrere Knoten, die als Auswahlmöglichkeit bei Entscheidungspunkten im Experiment dienen. Ein Startknoten sowie ein Endknoten repräsentieren dabei Experimentbeginn und –ende, dargestellt in nachfolgender Abbildung durch die schwarz gefärbten Knoten.



(Abbildung 4: Schematischer Aufbau des Experimentgraphen)

Ein Experiment wird durch die markierten Knoten spezifiziert, nachfolgend dargestellt durch graue Einfärbung der ausgewählten Knoten.



(Abbildung 5: Ein Experiment im Experimentgraph)

Der Experimentgraph, auf dem die Experimente abgebildet werden, muss gerichtet und azyklisch sein. Ferner soll gewährleistet werden, dass vom Startknoten in endlich vielen Schritten der Zielknoten erreicht werden kann. Auf die Möglichkeit, das Test-First Experiment in dem Experimentgraphen abbilden zu können, wird besonderer Wert gelegt. Außerdem soll nur ein Experiment gleichzeitig auf dem Experimentgraphen dargestellt werden.

Eine weitere Anforderung sieht die Möglichkeit der einfachen Erweiterung oder Veränderung der Entscheidungspunkte vor, dazu müssen die Daten entsprechend abgelegt werden. Siehe hierzu den Abschnitt über Datenhaltung im Entwurfsteil unter Kapitel 5.1.

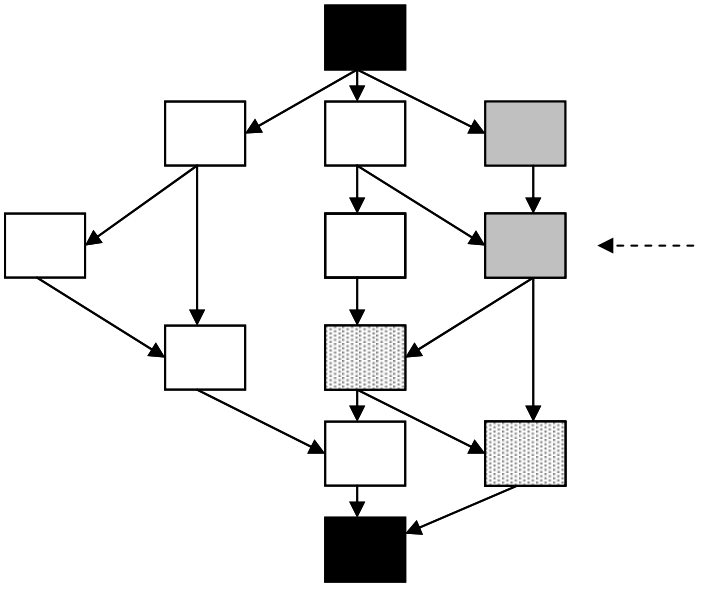
Die Knoten sollen verschiedene Informationen verwalten, dazu gehört eine Liste von Templates. Dies sind Dokumente, die beim Voranschreiten des Experiments nach und nach ausgefüllt werden müssen. Beispiele aus dem Test-First Experiment wären die Teilnehmerliste oder die Gruppenzusammensetzung der Testpersonen. Die Templates sollen in einem separaten Ordner untergebracht werden. Das Test-First Experiment, beziehungsweise die daraus resultierenden Dokumente, dienen als Grundlage für die Templates. Jeder Knoten enthält außerdem einen Beschreibungstext, der Informationen über den Arbeitsschritt liefert.

Das Experimentplanungswerkzeug soll durch PlugIns erweiterbar sein, eine einheitliche Schnittstelle für die PlugIns soll dies ermöglichen. In dieser Arbeit wird ein PlugIn entwickelt, um Experimente analysieren und charakterisieren zu können.

4.1.2 Funktionale Anforderungen des Hauptprogramms

Benutzer sind in der Lage Experimente anzulegen, dies wird in Use Case Nr.1 beschrieben. Ein neues Experiment wird anhand seines Namens identifiziert, welcher durch den Benutzer eingegeben wird. In einem, dem Namen entsprechenden, erstellten Verzeichnis wird das leere Experiment abgelegt. Es besitzt anfangs jeweils eine leere Liste der Knoten sowie der eigenen Dokumente.

Use Case Nr.2 zeigt, wie der Benutzer dieses leere Experiment füllen kann. Dazu wird ein Knoten ausgewählt und markiert. Der anfangs leeren Knotenliste wird der markierte Knoten hinzugefügt. Das Löschen, beziehungsweise Demarkieren eines Knotens mit entsprechendem Entfernen aus der Knotenliste zeigt Use Case Nr.3. Um zu gewährleisten, dass durch willkürliches Markieren sowie Löschen der Knoten kein unbrauchbares Experiment entsteht, beispielsweise ein lückenhaftes, wenn der Pfad unterbrochen wird, soll ein entsprechender Markierungsalgorithmus den Vorgang überwachen und entsprechende Warnhinweise liefern. Dabei ist beim Markieren zu beachten, dass der erste Knoten jedes Experiments der Startknoten und der letzte der Endknoten ist. Weiter dürfen nur Knoten markiert werden, die mindestens einen markierten Vorgängerknoten besitzen. Da der markierte Weg einem Experimentaufbau entspricht, dürfen keine Knoten auf gleicher Ebene markiert werden. Das Löschen der Knoten darf nur erfolgen solange dadurch keine Lücken in dem Pfad entstehen. Nachfolgende Abbildung dient der Veranschaulichung, wobei die grau schattierten Knoten mögliche selektierbare darstellen, ausgehend von dem grau hinterlegten, aktuell gewählten Knoten. Dabei kann nur einer der beiden möglichen gewählt werden, da sie sich beide auf derselben Ebene befinden. Ausgehend von den zwei markierten Knoten kann nur der untere gelöscht werden, gekennzeichnet durch den gestrichelten Pfeil, da ansonsten eine Lücke in dem Experimentpfad entstehen würde.



(Abbildung 6: Knoten markieren und löschen)

Im Experimentgraph kann ein Benutzer jeden Knoten aktivieren. Dies geschieht unabhängig vom Markieren und dient dem Anzeigen der Detailinformationen des aktivierten Knotens. Dies beinhaltet Templates, eigene Dokumente sowie Notizen. Templates und eigene Dokumente werden durch den Namen repräsentiert und Notizen, falls vorhanden, innerhalb eines Textfeldes angezeigt. Der Benutzer hat nun, ausgehend vom Zustand des Knotens, verschiedene Möglichkeiten. Unabhängig vom Auswahlzustand können Templates des aktivierten Knotens heruntergeladen, also an einer beliebigen, wählbaren Stelle abgespeichert werden. Use Case Nr.4 verdeutlicht diesen Vorgang. Des Weiteren besitzt jeder Knoten ein Notizfeld, in dem der Benutzer Notizen eingeben, sowie für den Knoten übernehmen kann. Ist der aktivierte Knoten gleichzeitig markiert, so steht dem Benutzer die Möglichkeit zur Verfügung, eigene Dokumente hochzuladen und einem Template des Knotens zuzuordnen. Use Case Nr.5 zeigt diesen Vorgang, der bei einer Zuordnung eines Dokuments zu einem Template dieses als ausgefüllt interpretiert. Beim Hochladen wird eine Kopie des Dokuments angefertigt und im Experimentordner abgelegt. Das Löschen hochgeladener Dokumente ist ebenfalls über einen entsprechenden Knopf möglich. Die Templates können nicht über die Oberfläche gelöscht werden.

Ein Experiment kann gespeichert und geladen werden. Um die Speicherfunktion nutzen zu können, muss ein Experiment mindestens einen Eintrag in der Knotenliste vorweisen. Abgelegt werden dabei alle markierten Knoten mit zugehörigen Dokumenten und vorhandenen Notizen. Kapitel 5.1 bietet einen Einblick in die Datenhaltung und zeigt, wie Experimente gespeichert werden.

Das Laden von Experimenten geschieht über eine Auswahlliste, in der alle verfügbaren Experimente dargestellt werden. Nachdem der Benutzer eins ausgewählt hat, wird es im Experimentgraph visualisiert. Da immer nur ein Experiment zur selben Zeit dargestellt werden soll, wird ein vorher angezeigtes Experiment aus der Anzeige entfernt. Use Case Nr. 6 veranschaulicht diesen Vorgang.

PlugIns können über einen Menüauswahlpunkt dem Programm hinzugefügt, also geladen oder entfernt werden. Use Case Nr.7 zeigt wie ein PlugIn geladen wird.

4.1.3 Anforderungen an das PlugIn

Das PlugIn soll die in Kapitel 3.3 und 3.4 betrachteten Aufgaben implementieren. Sie sollen dabei auf einer grafischen Oberfläche visualisiert werden. Spezifische Anforderungen an die Benutzeroberfläche beschreibt Kapitel 4.2.6. Der Benutzer bekommt die Möglichkeit, das aktuelle Experiment mit gespeicherten zu vergleichen, siehe Use Case Nr. 8, oder zu prüfen, ob das aktuelle Experiment vollständig bearbeitet und somit abgeschlossen werden kann, beschrieben in Use Case Nr. 9. Des Weiteren kann der Benutzer ein Standardmaß des momentan geplanten Experiments anzeigen lassen.

Die Ähnlichkeitsbestimmung und die Prüfung auf Abschließbarkeit des Experiments bedürfen einer entsprechenden Aktion des Benutzers, das Standardmaß wird automatisch angezeigt, sobald das Experiment mindestens einen Knoten besitzt. Bei jeder Änderung des Experiments im Hauptprogramm wird der angezeigte Wert erneut berechnet. Dazu zählen Löschen und Markieren von Knoten, sowie Laden von Experimenten.

4.1.4 Use Cases

Anhand von Use Cases werden die grundlegenden Funktionalitäten des Werkzeugs genauer spezifiziert. Es folgt eine detaillierte Darstellung der Use Cases.

Use Case Nr.1	Neues Experiment anlegen
Stakeholder und Interesse	Benutzer möchte ein neues Experiment planen
Voraussetzung	Programm wurde erfolgreich gestartet
Garantie	Benutzer weiß, ob neues Experiment angelegt wurde Es wird kein vorhandenes Experiment überschrieben
Erfolgsfall	Neues Experiment-Verzeichnis wird angelegt Im Experimentgraph wird das neue, leere Experiment dargestellt
Auslöser	Benutzer wählt „new Experiment“
Beschreibung	System fragt nach Experimentnamen Benutzer gibt Namen ein System legt neues Verzeichnis mit diesem Namen im Experiment-Verzeichnis an System entfernt sämtliche markierten Einträge im Experimentgraph System zeigt angelegtes Experiment als aktuelles Experiment an
Erweiterungen	2a Wenn bereits ein Experiment mit dem eingegebenen Namen existiert, fragt das System nach einem alternativen Namen 3a Wenn Experiment-Verzeichnis nicht existiert, wird kein neues Experiment angelegt

Use Case Nr.2	Hinzufügen eines Knotens aus dem Experiment
Stakeholder und Interesse	Benutzer möchte einen Knoten zum aktuellen Experiment hinzufügen
Voraussetzung	Es ist ein Experiment angelegt oder geladen
Garantie	Es entsteht kein lückenhafter Experimentpfad
Erfolgsfall	Knoten wurde Experiment hinzugefügt und wird im Experimentgraph als markiert dargestellt
Auslöser	Benutzer führt Doppelklick auf unmarkierten Knoten aus
Beschreibung	System prüft, ob ein Experiment existiert System prüft, ob Knoten markiert werden kann System fügt Knoten in die Knotenliste des Experiments ein System markiert entsprechenden Knoten im Experimentgraph
Erweiterungen	2a Wenn Knoten nicht markiert werden kann, liefert das System eine entsprechende Meldung, die Schritte 3 und 4 entfallen

Use Case Nr.3	Löschen eines Knotens aus dem Experiment
Stakeholder und Interesse	Benutzer möchte einen Knoten aus dem aktuellen Experiment entfernen
Voraussetzung	Es existiert ein Experiment, dessen Knotenliste mindestens einen Eintrag enthält
Garantie	Es entstehen keine Lücken im Experimentpfad
Erfolgsfall	Knoten wurde aus Experiment entfernt Im Experimentgraph wird die Knotenmarkierung gelöscht
Auslöser	Benutzer führt Doppelklick auf markierten Knoten aus
Beschreibung	System prüft, ob Knotenmarkierung gelöscht werden kann System löscht Knoten aus der Knotenliste des Experiments System löscht Notizen und Dokumente von diesem Knoten System löscht Markierung des entsprechenden Knoten im Experimentgraph
Erweiterungen	1a Wenn Knotenmarkierung nicht gelöscht werden kann, zeigt das System eine entsprechende Meldung, die Schritte 2, 3 und 4 entfallen

Use Case Nr.4	Herunterladen eines Templates
Stakeholder und Interesse	Benutzer möchte ein Template im einem Verzeichnis seiner Wahl speichern
Voraussetzung	In der Templateliste des aktiven Knotens ist ein Eintrag vorhanden und selektiert
Garantie	Benutzer wird über Erfolg oder Misserfolg informiert
Erfolgsfall	Eine Kopie des Templates wird im Dateisystem abgelegt
Auslöser	Benutzer wählt „download Template“
Beschreibung	System überprüft, ob das gewählte Template im Template-Verzeichnis liegt System zeigt Dateiauswahldialog Benutzer wählt ein Verzeichnis aus System kopiert Template und speichert die Kopie im ausgewählten Verzeichnis System meldet erfolgreichen Download
Erweiterungen	1a Wenn Template nicht im Template-Verzeichnis vorhanden, zeigt System entsprechende Meldung, alle restlichen Schritte entfallen

Use Case Nr.5	Hochladen eines Dokuments
Stakeholder und Interesse	Benutzer möchte ein eigenes Dokument dem Experiment hinzufügen
Voraussetzung	Es existiert ein aktuelles Experiment mit einem markierten, aktivierten Knoten
Garantie	Benutzer wird über Erfolg oder Misserfolg informiert
Erfolgsfall	Eine Kopie des Dokuments wird im Experimentordner abgelegt Das Dokument wird einem Knoten und einem Template zugeordnet
Auslöser	Benutzer wählt „upload Document“
Beschreibung	System zeigt Dateiauswahldialog Benutzer wählt ein Dokument aus System zeigt Dialog für Zuordnung des Dokuments mit vorhandenen Templates des aktiven Knotens Benutzer nimmt Zuordnung vor System fragt in einem weiteren Eingabedialog nach der benötigten Zeit zum Ausfüllen des Dokuments Benutzer gibt Zeit ein System kopiert Dokument und speichert sie im Experimentverzeichnis System speichert Dokumentname mit Templatezuordnung und entsprechender Zeit in dem selektierten Knoten
Erweiterungen	4a Benutzer hat auch die Möglichkeit das Dokument explizit keinem Template zuzuordnen, weiter mit 5

Use Case Nr.6	Bestehendes Experiment laden
Stakeholder und Interesse	Benutzer möchte ein vorhandenes Experiment ansehen Benutzer möchte ein Experiment bearbeiten
Voraussetzung	Im Experiment-Verzeichnis befindet sich mindestens ein Experiment
Garantie	Experiment wird komplett oder gar nicht geladen
Erfolgsfall	Das gewählte Experiment wird mit sämtlichen Einträgen dargestellt (Knotenmarkierungen, Dokumente, Notizen)
Auslöser	Benutzer wählt „open Experiment“

Beschreibung	System zeigt Auswahldialog mit allen sich im Experiment-Verzeichnis befindlichen Experimenten Benutzer wählt ein Experiment aus System liest alle Einträge dieses Experiments aus System löscht alle Einträge des bisher dargestellten Experiments System markiert Experimentknoten Zu jedem markierten Knoten werden die Dokument- und Zeitinformationen sowie eventuell vorhandene Notizen übernommen
Erweiterungen	3a Wenn beim Auslesen ein Fehler auftritt, entfallen die folgenden Schritte und das System liefert eine entsprechende Meldung

Use Case Nr.7	PlugIn laden
Stakeholder und Interesse	Benutzer möchte das Programm um zusätzliche Funktionalitäten erweitern
Voraussetzung	Plugin-Verzeichnis enthält mindestens ein PlugIn
Garantie	PlugIn wird komplett oder gar nicht geladen
Erfolgsfall	Die Funktionalität des PlugIns kann im vollen Umfang genutzt werden Auf einem im Hauptprogramm vorgesehenen Bereich wird die grafische Oberfläche des PlugIns visualisiert
Auslöser	Benutzer wählt „load PlugIn“
Beschreibung	System zeigt Liste aller verfügbarer PlugIns Benutzer wählt ein PlugIn aus System lädt PlugIn System initialisiert PlugIn mit dem aktuell geladenen Experiment System stellt Oberfläche des PlugIns im dafür vorgesehenen Bereich dar
Erweiterungen	3a Wenn Probleme auftreten und das PlugIn nicht korrekt geladen werden kann, bricht das System den Vorgang ab und informiert den Benutzer

Use Case Nr.8	Experiment vergleichen
Stakeholder und Interesse	Benutzer möchte aktuelles Experiment mit abgespeicherten vergleichen
Voraussetzung	PlugIn ist geladen Ein Experiment ist angelegt und die Knotenliste enthält mindestens einen Eintrag Es existiert mindestens ein abgespeichertes Experiment
Garantie	
Erfolgsfall	Aktuelles Experiment wird mit abgespeicherten verglichen und Ähnlichkeit dargestellt

Auslöser	Benutzer wählt auf der PlugIn-Oberfläche „Experiment vergleichen“
Beschreibung	PlugIn berechnet Ähnlichkeit von aktuellen Experiment mit sämtlichen abgespeicherten In einer Liste werden alle gespeicherten Experimente anhand des Namens mit ihrer Ähnlichkeit zum aktuellen aufgeführt
Erweiterungen	1a Wenn kein abgespeichertes Experiment existiert, liefert das System eine entsprechende Meldung, sämtliche Schritte entfallen

Use Case Nr.9	Experiment abschließen
Stakeholder und Interesse	Benutzer möchte erfahren, ob aktuelles Experiment vollständig bearbeitet wurde
Voraussetzung	PlugIn ist geladen Ein Experiment ist angelegt, und die Knotenliste enthält mindestens einen Eintrag
Garantie	Benutzer wird über Erfolg oder Misserfolg informiert
Erfolgsfall	Benutzer erhält Meldung über Abschließbarkeit des aktuellen Experiments
Auslöser	Benutzer wählt auf der PlugIn-Oberfläche „Experiment abschließen“
Beschreibung	PlugIn überprüft, ob sämtliche Bedingungen für das Abschließen von Experiment erfüllt sind Benutzer erhält Meldung, ob das Experiment abgeschlossen werden kann
Erweiterungen	

4.1.5 Qualitätsanforderungen

Durch die Wahl von Java als Programmiersprache ist die Portierbarkeit des Programms bereits gegeben. Der Fokus der Qualitätsanforderungen liegt auf einer einfachen Erweiterungsmöglichkeit des Experimentgraphen. Das Hinzufügen neuer oder Löschen bestehender Knoten und Templates soll von den Benutzern, bei denen es sich um erfahrene Personen aus dem Bereich des Software Engineering handelt, mit geringem Aufwand durchführbar sein.

4.1.6 Anforderungen an die Benutzeroberfläche

Auf einer grafischen Oberfläche, deren Hauptkomponente der Experimentgraph bildet, können sämtliche Benutzereingaben durchgeführt werden. Weiterhin gibt es noch zwei, inhaltlich getrennte Bereiche. Einer dient der Anzeige von Detailinformationen über den gerade aktiven Knoten, der andere ist für die PlugIns vorgesehen. Zu dem Knoten soll je eine Liste mit Templates und Dokumenten sowie ein Notizfeld dargestellt werden. Über

entsprechende Knöpfe sollen sich Templates runter- sowie Dokumente hoch- oder runterladen lassen.

Der PlugIn Bereich soll anfangs zweigeteilt werden, entsprechend den vorhergesehenen zwei PlugIns. Dabei ist der Platz der beiden gleich groß. Eine spätere Erweiterung in eine dynamische Anpassung in Abhängigkeit der Anzahl geladenen PlugIns ist denkbar.

5 Entwurf des Experimentplanungswerkzeugs

Dieses Kapitel widmet sich dem Entwurf des Werkzeugs. Es folgt ein Blick auf die Datenhaltung, die Art der Visualisierung des Graphen, verwendete Design Pattern, Gestaltung der Grafischen Benutzeroberfläche sowie ein Überblick der Packages mit kurzer Aufgabenbeschreibung der Klassen.

5.1 Datenhaltung

Zur Speicherung der Informationen des Experimentgraph sowie der Experimente gab es zwei Möglichkeiten. Zum einen der Einsatz einer Datenbank sowie das Ablegen der Informationen in Extensible Markup Language Dateien [Qui05], kurz XML genannt. Auf den Einsatz einer Datenbank sowie dem dadurch entstehenden Aufwand der Konfiguration und Verwaltung wurde, auch aus zeitlichen Aspekten, verzichtet. Dementsprechend wurde XML zur Datenhaltung eingesetzt. Die Vorteile der Plattform- und Herstellerunabhängigkeit gekoppelt mit dem Einsatz von Java bietet ein hohes Maß an Flexibilität.

XML ist ein Standard zur Erstellung maschinenlesbarer Dokumente in Form einer Baumstruktur. Die Details der jeweiligen Dokumente, insbesondere die Festlegung der Strukturelemente sowie ihrer Anordnung müssen für einen konkreten Anwendungsfall spezifiziert werden. Dabei lassen sich die Namen der einzelnen Strukturelemente frei wählen. Eine Grundidee von XML ist die Trennung von Daten und Repräsentation. In XML abgelegte Daten lassen sich somit auf verschiedene Arten verarbeiten. Die Struktur von XML Dokumenten lässt sich über Schemasprachen festlegen. Dabei gibt es zwei Arten, Dokument Type Definitions, kurz DTD genannt, sowie XML Schema. Erstere ist älter und hat den Nachteil, dass nur vordefinierte Datentypen verwendet werden können, und sie sind nicht XML konform. Die Wahl fiel auf XML Schema, da sie nicht nur moderner, sondern auch mächtiger sind. Es lassen sich eigene Datentypen frei definieren sowie der mögliche Inhalt von Elementen und Attributen beschränken. Für diese Arbeit wurden zwei XML Schema definiert, jeweils für den Experimentgraph und für ein Experiment, welche im Anhang aufgeführt sind.

Zum Auslesen und Verarbeiten, dem so genannten Parsen der XML Dokumente bietet Java verschiedene Möglichkeiten. Zum einem die SAX Parser, welche eine XML Datei sequenziell abarbeiten und dabei nicht die komplette XML Datei im Speicher halten. Zum

anderen die DOM Parser, welche die komplette XML Datei in den Speicher einlesen und daraus einen Objektbaum aufbauen. Vorteil davon ist, dass sämtliche Elemente in einer hierarchischen Struktur vorliegen, auf die gleichermaßen zugegriffen werden kann. Außerdem lässt sich der Objektbaum manipulieren und kann später erneut als XML Datei abgespeichert werden. Die Wahl fiel auf den DOM Parser, da der Nachteil des intensiveren Speicherbedarfes durch die Vorteile des Objektbaumes mehr als kompensiert wird. Gerade die Möglichkeit diesen Baum zu manipulieren und wieder als XML Datei abzulegen, ermöglicht ein komfortables Speichern der Experimente.

Die gewünschte Möglichkeit des einfachen Änderns und Erweiterns des Experimentgraphen wird durch den Einsatz von XML Dateien zur Datenhaltung gewährleistet. XML Dateien sind Textdateien, die mit dem nötigen Wissen über Aufbau und Struktur verhältnismäßig leicht verändert oder erweitert werden können. Das ist gewährleistet, da es sich bei den Benutzern des Programms um erfahrene Personen aus dem Bereich handelt.

5.1.1 Experimentgraph

Der Experimentgraph wird bei jedem Programmstart ausgelesen und bildet die Hauptkomponente des Programms. Ein Speichern ist nicht vorgesehen, da etwaige Änderungen über die entsprechende XML Datei vorgenommen werden sollen anstatt im Programm selbst. Um den Graph erstellen zu können, enthält die XML Datei Informationen über die Knoten Dazu zählen Beschreibungstext und Templates, sowie Koordinaten zur grafischen Anordnung und ein Bezug zum Vorgänger, um den Graph eindeutig konstruieren zu können. Die Templates enthalten zusätzlich Informationen über die geschätzte Ausfüllzeit, ob die benötigte Anzahl des Dokuments von der Personenzahl abhängt, wie viele Exemplare des Templates ausgefüllt werden müssen, ob die benötigte Anzahl des Dokuments von der Anzahl der Datenerhebungen abhängt, und wie oft das Dokument pro Datenerhebung bearbeitet werden muss.

Erwähnenswert ist noch, dass kein eigenes Element für Kanten existiert. Der Grund für die Entscheidung ist, dass Kanten in dem Graphen über keine zusätzlichen Funktionen verfügen und die Beziehung zwischen den Knoten über das Vorgängerfeld aufgestellt werden kann.

Das Schema für den Experimentgraph definiert als Wurzelelement ein Element *graph*, mit der Bedingung mindestens zwei Unterelemente *node* zu besitzen. Damit sind der Start- und Zielknoten gemeint. Eine maximale Begrenzung der *node*-Elemente ist nicht vorgesehen.

Jedes *node* Element erhält eine eindeutige *ID* in Form eines Attributs, womit eine Referenzierung auf die Vorgängerknoten möglich ist. Anhand dieser Informationen können sämtliche Kanten erstellt werden. Der Titel des Knotens wird im Element *name* gespeichert, welches jeder Knoten enthalten muss. Des Weiteren gibt es noch zwei Attribute mit Namen *xwert* und *ywert*. Aus ihnen ergeben sich die Koordinaten für die Anordnung des Knotens auf der grafischen Oberfläche. Dieses Festlegen der Koordinaten erfolgt, da für die zum Visualisieren benutzte Bibliothek JGraph kein entsprechender Layoutalgorithmus gefunden wurde. Aus Zeitgründen wurde auf die Entwicklung eines eigenen Algorithmus

verzichtet. Die Beschreibung des Arbeitsschrittes erfolgt über das in jedem Knoten genau einmal enthaltene Unterelement *description*, welches einen kurzen Beschreibungstext enthält und in jedem Knoten genau einmal enthalten ist.

Jeder Knoten enthält beliebig viele Unterelemente *template*, deren Inhalt dem Templatenamen entspricht und über folgende Attribute verfügt:

- *expTime*: Gibt die geschätzte Ausfüllzeit in Personenstunden an
- *numberOfDataCollection:* Benötigte Anzahl des Dokuments pro Datenerhebung
- *numberOfPerson:* Benötigte Anzahl des Dokuments pro Versuchsperson

Das vollständige XML Schema findet sich im Anhang A1.

5.1.2 Experimente

Ein zweites verwendetes XML Schema wurde für die Experimente definiert. Zwecks Redundanzvermeidung werden nur Informationen abgelegt, die nicht aus dem Experimentgraph entnommen werden können. Es genügt beispielsweise die eindeutige ID der Knoten, um auf die Knoteninformationen des Experimentgraphen zugreifen zu können. Als zusätzliche Information für die Knoten werden die Dokumente in der Experimentdatei abgespeichert, welche den Templates des betrachteten Knotens zugeordnet werden.

Als Wurzelelement für das XML Schema der Experimente dient das Element mit Namen *experiment*. Es enthält an Unterelementen *name*, *numberOfSubjects*, *numberOfDataCollection* und *scheduledTime*, welche jeweils genau einmal vorkommen. Sie repräsentieren den Namen, die Anzahl der Versuchspersonen, die Häufigkeit der Datenerhebung sowie die für das Experiment eingeplante Zeit. Die letzen drei werden Anfangs mit Null initialisiert und können im Programm verändert werden.

Die im Experiment enthaltenen Knoten, wobei mindestens einer im Experiment existieren muss, werden über das Element *node*, welches ein Attribut *ID* besitzt, dargestellt. Anhand der ID lässt sich der entsprechende Knoten dem des Experimentgraphen zuordnen. Die Knotenelemente können *document* Elemente als Unterelemente besitzen, welche die Attribute *template* und *attendedTime* enthalten. Ersteres enthält den Namen des Templates, letzteres die zum Ausfüllen des Dokuments benötigte Zeit. Im Anhang A2 befindet sich das vollständige XML Schema.

5.2 Darstellung des Graphen

Für die Visualisierung der Graphinformationen fiel die Wahl auf eine vorhandene, frei verfügbare Bibliothek. Dabei handelt es sich um eine Kombination der opensource Javabibliotheken JGraph [Ald05] und JGraphT [Nav05], in den zum Zeitpunkt der Arbeit aktuellen Versionen. JGraph ist Java und swingbasiert und ermöglicht die Darstellung von Graphen. Die sehr umfangreichen Möglichkeiten, dessen Verwendungen sich trotz einer umfassenden API Dokumentation recht schwierig gestalteten, waren der Grund zusätzlich JGraphT zu verwenden. Es benutzt JGraph zur Darstellung, erlaubt aber eine einfache

Verwaltung der Knoteninformationen. Somit konnten durch die Kombination beider Bibliotheken die jeweiligen Vorteile ausgenutzt werden.

5.3 Eingesetze Design Pattern

Eingesetzt wurden das *Model View Controller* Pattern, kurz MVC Pattern genannt, das *Singleton* Pattern, sowie das *Observer* Pattern. Es folgt ein Überblick der Funktionsweise, sowie Vorteile der eingesetzten Entwurfsmuster. Eine ausführliche Beschreibungen sämtlicher Design Pattern bietet unter anderem [FRF03].

5.3.1 MVC Pattern

Das mittlerweile sehr bekannte MVC Pattern bietet den Vorteil einer strikten Trennung bestimmter Programmeigenschaften zwischen Datenmodell(*Model*), Präsentation(*View*) und Programmsteuerung (*Controller*). Dadurch wird ein flexibles Programmdesign erreicht, spätere Änderungen oder Erweiterungen, beispielsweise der Austausch der Darstellungsform, sind somit leicht durchzuführen und ermöglicht die Widerverwendbarkeit einzelner Komponenten. Die Trennung bietet außerdem eine bessere Übersicht und Ordnung.

5.3.2 Singleton Pattern

Ebenfalls sehr bekannt ist das Singleton Pattern. Es stellt sicher, dass zu einer Klasse immer nur genau ein Objekt erzeugt werden kann und ermöglicht einen globalen Zugriff auf dieses Objekt.

Da es nur einen Experimentgraphen gibt, wurde dieser als Singleton realisiert. Aufgrund der Möglichkeit des globalen Zugriffs wurde der Experimentgraph als Zugriffsklasse des aktuell geladenen Experiments eingesetzt. Durch den Singleton Einsatz wird somit die Anforderung, dass immer nur ein Experiment gleichzeitig angezeigt wird, erfüllt.

5.3.3 Observer Pattern

Eingesetzt wird das Observer Pattern, falls Objektänderungen weitergegeben werden müssen. Im Fall einer grafischen Benutzeroberfläche ist es notwendig, die Komponenten, welche für die Visualisierung zuständig sind, über Änderungen des darzustellenden Objektes zu informieren. Beispielsweise die Darstellung von sich ständig ändernden Messergebnissen in Diagrammen. Bei jeder Änderung muss der aktuelle Wert angezeigt werden, eine Benachrichtigung über das Ändern des Wertes ist demnach zwingend notwendig. Ferner ist dieses Pattern ein Bestandteil des MVC Entwurfmusters, nämlich die Verbindung zwischen Datenmodell und Präsentation. Das Observer Pattern bietet die Möglichkeit, dass sich Beobachter(*Observer*) an einem zu beobachtenden Objekt anmelden können und anschließend über jede Änderung informiert werden, um entsprechend zu reagieren.

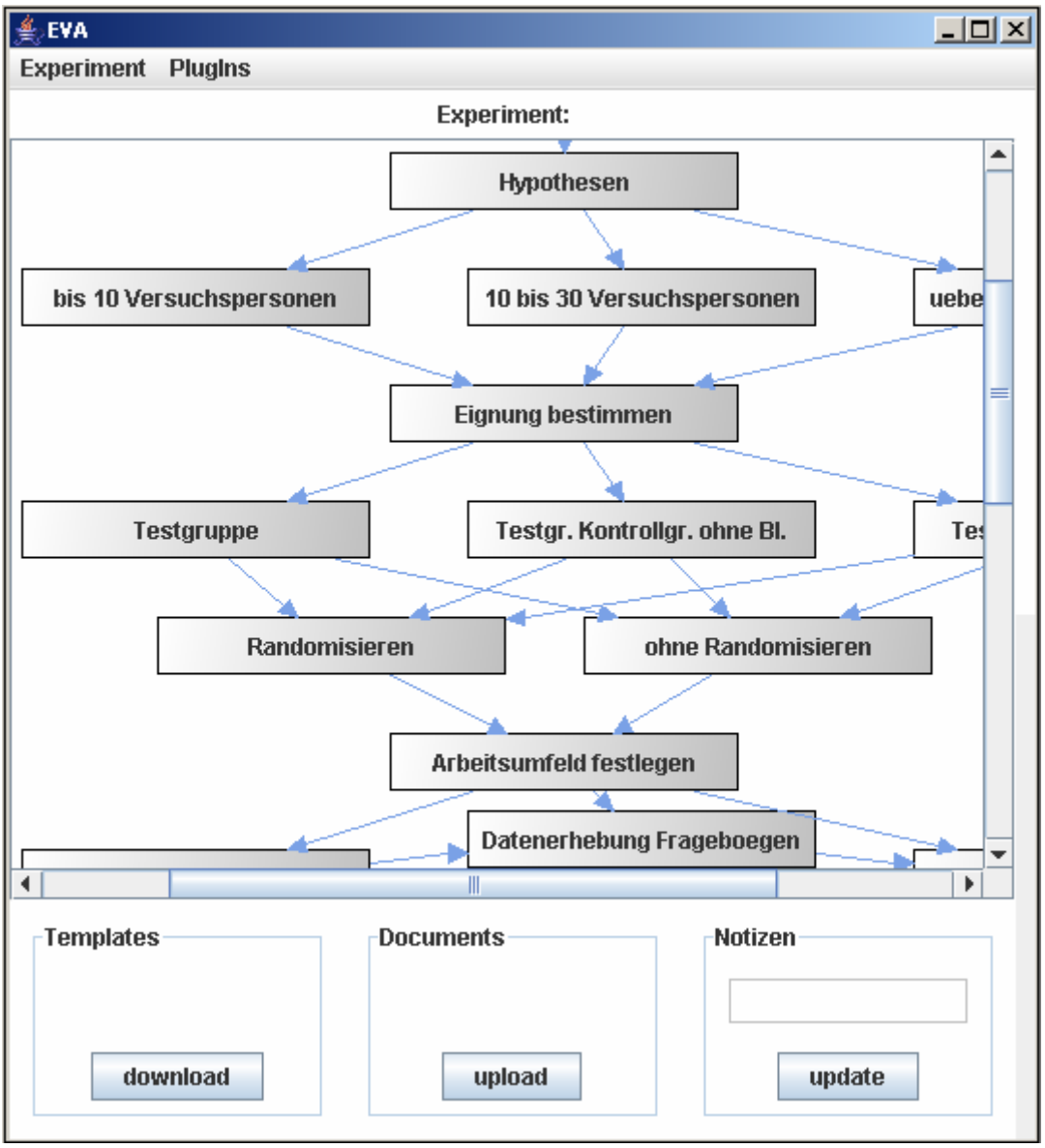
Mit Hilfe des Observer Patterns wurde die PlugIn Schnittstelle gestaltet. Dabei muss jedes PlugIn in einer Klasse das Interface *IGraphChangedListener* implementieren und sich beim aktuell geladenen Experiment anmelden. Im PlugIn übernimmt diese Funktion die Klasse *PlugInListener*.

5.4 Grafische Benutzeroberfläche

Das Hauptfenster der Oberfläche wurde entsprechend der Anforderungen dreigeteilt. Über die Menüleiste lassen sich die Experimente neu angelegen, laden oder speichern. Ferner ist es möglich, PlugIns zu starten und zu beenden.

Den größten Bereich nimmt der Experimentgraph ein, der im linken oberen Bereich beginnt. Unterhalb befindet sich ein Bereich zum Anzeigen der Detailinformationen des aktiven Knotens. Es existieren zwei Listen, in denen jeweils vorhandene Templates und Dokumente angezeigt werden, sowie ein Textfeld, in dem vorhandene Notizen zu dem aktivierten Knoten angezeigt und verändert werden können. Ein entsprechender Knopf „update“ befindet sich unterhalb des Notizfeldes. Ein „download“ Knopf befindet sich unterhalb der Templateliste und der Dokumente. Dieser ermöglicht das Abspeichern der selektierten Datei in ein beliebiges Verzeichnis. Neben diesem gibt es beim Dokumentbereich noch zwei weitere Knöpfe, „upload“ und „delete“, über die sich eigene Dokumente hochladen, beziehungsweise vorhandene, ausgewählte Dokumente löschen lassen.

Der dritte Bereich ist für die Visualisierung der PlugIns vorgesehen und ermöglicht die gleichzeitige Anzeige zweier PlugIns. Dementsprechend wurde der Raum in eine obere sowie untere Hälfte geteilt. Dabei wird dem erstgeladenen PlugIn die obere und dem als zweites geladenen PlugIn die untere Hälfte zugewiesen. Falls nur eins oder gar kein PlugIn geladen wurde, bleiben die entsprechenden Bereiche frei. Nachstehende Abbildung zeigt das Programm nach dem Start, ohne geladene PlugIns.

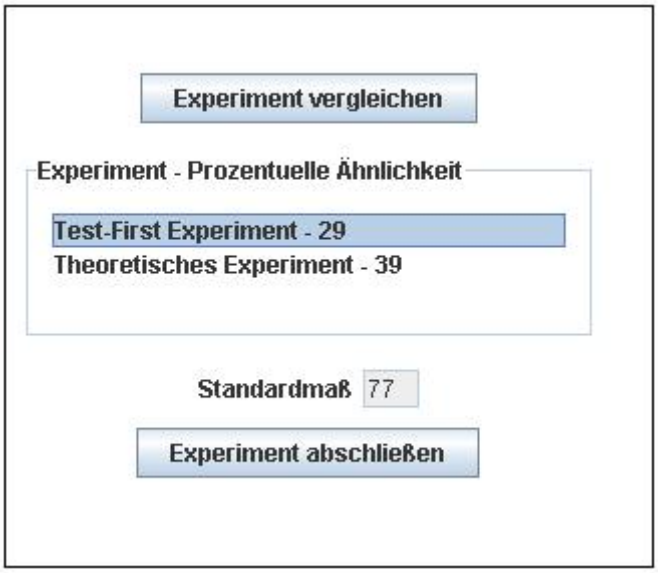


(Abbildung 7: Experimentplanungwerkzeug ohne geladene PlugIns)

Die PlugIn Schnittstelle stellt als Hauptkomponente ein JPanel Objekt zur Verfügung, in dem eine freie Anordnung der darzustellenden Informationen des PlugIns möglich ist. Diese Fläche enthält einen Knopf „Experiment vergleichen“, mit dem sich die darunter befindliche Liste laden lässt, in der die Experimente mit Namen und entsprechend berechneter Ähnlichkeit dargestellt werden. Bei jedem Betätigen des Vergleichsknopfes wird ein entsprechender Vergleich vorgenommen und die Anzeige aktualisiert.

Unterhalb befindet sich die Anzeige für das Standardmaß, welche bei jeglicher Änderung der Entscheidungspunkte, also durch das Markieren oder Demarkieren eines Knotens, berechnet und in der Anzeige aktualisiert wird.

Im unteren Bereich befindet sich der Knopf „Experiment abschließen“. Damit lässt sich eine Prüfung durchführen, ob sämtliche Bedingungen erfüllt sind, um das Experiment komplett bearbeitet zu haben. Eine entsprechende Meldung informiert den Benutzer.



(Abbildung 8: PlugIn-Oberfläche)

5.5 Packages des Hauptprogramms

Das Hauptprogramm wurde in sechs Packages unterteilt. Es folgt die Erläuterung der Funktionen sowie eine auszugsweise Betrachtung der Klassen mit den jeweiligen Aufgaben.

5.5.1 Package interfaces

Die Implementierung des Programms orientiert sich an dem Grundsatz „Program to an interface, not an implementation“ [GHJ98]. Die Grundidee dabei ist, im Programm nur Schnittstellen zu verwenden und die Implementierung der Interfaces in ein anderes Package auszulagern. Dadurch wird eine geringe Abhängigkeit zu der Implementierung erreicht, und es lassen sich konkrete Klassen leicht austauschen. Es muss lediglich das Interface neu implementiert werden.

Im Interface Package befinden sich alle im Hauptprogramm verwendete Interfaces sowie die Schnittstellen für die PlugIns. Es folgt eine Auflistung mit kurzer Erläuterung der Aufgaben der einzelnen Interfaces, um einen Überblick erhalten zu können. Dabei werden funktional zusammengehörige Interfaces in Gruppen eingeteilt.

Für den Datenzugriff zuständige Interfaces:

IGraphAccess - Parsen der Experimentgraphinformationen und Erstellen des Experimentgraphen

IExperimentAccess - Auslesen der Experimentinformationen und Laden des Experiments

IDocumentAccess - Speichern und Löschen sowie Kopieren von Dokumentdateien

Für die Controllerfunktionen zuständigen Interfaces:

IGraphListener - Erbt von Mouselistener, für Klassen, die auf das Selektieren von Graphelementen reagieren

IButtonListener - Erbt von ActionListener, für Klassen, die auf Benutzerinteraktionen der Knöpfe reagieren

IMenuListener - Erbt von ActionListener, für Klassen, die auf Menüereignisse reagieren

Für die grafische Oberfläche zuständige Interfaces:

IMainFrame - Interface für das Hauptfenster

IGraphView - Zuständig für Visualisierung des Experimentgraphen

Für die Datenhaltung zuständige Interfaces:

IExperimentGraph
- Setzen und Abfragen des aktuellen Experiments
- Markieren von Knoten
- Positionieren der Knoten
- Schnittstelle, an der sich die Darstellungsklassen anmelden können

IExperiment
- Methoden zum Entfernen und Hinzufügen der Knoten
- Schnittstelle zum Anmelden von PlugIns
- Methoden, um angemeldete PlugIns über Änderungen am Experiment zu informieren

INode
- Repräsentation der Inhalte von Knoten
- Methoden zur Zustandsabfrage und -änderung (Notizen, Templates, Dokumente, Markierung) von Knoten

IDocument	- Ermöglicht Setzen und Abfragen von Zeiten und Templates zu einem Objekt einer Klasse, die dieses Interface implementiert
ITemplate	- Erlaubt Abfrage der Templateinformationen

Für die PlugIn Schnittstelle zuständige Interfaces:

IPlugIn	- Hauptklasse von jedem PlugIn muss dieses Interface implementieren - Methoden zum Initialisieren des PlugIns mit dem Experimentgraph, einem Experiment und der verfügbaren Oberfläche
IGraphChangedListener	- Implementierte Klasse kann sich beim Experiment als Observer anmelden und wird über Änderungen bezüglich des Experiments informiert

5.5.2 Pacakge dataAccess

Hier befinden sich die implementierten Klassen zum Auslesen der Experimentgraph- und Experimentinformationen, als auch zum Hoch- und Runterladen von Dokumenten.

GraphAccess	- Liest XML Datei des Experimentgraphen aus und erstellt den Graphen
ExperimentAccess	- Zuständig für Auslesen und Abspeichern von Experimenten in Form entsprechender XML Dateien
DokumentAccess	- Zuständig für das Speichern und Laden von Dokumenten

5.5.3 Package data

Hier befinden sich Klassen, welche zur Datenhaltung benötigt werden.

Experimentgraph	- Realisiert als Singleton, da es nur einen Experimentgraphen gibt - Dient als Zugriffsklasse auf das geladene Experiment - Entspricht dem *Model* im Sinne des MVC Pattern
Experiment	- Enthält Variablen zum Speichern der Knoten, experimentspezifischer Informationen sowie entsprechende Zugriffsmethoden
Node	- Hält Referenz auf Dokumente und Templates des geladenen Experiments

	- Bietet Methoden zum Setzen oder Entfernen der Knotenmarkierung sowie dem Arbeiten mit Dokumenten und Templates
Edge	- Repräsentiert eine Kante
Template	- Repräsentiert ein Template mit entsprechenden Methoden
MyDocument	- Repräsentiert ein Dokument mit entsprechenden Methoden

5.5.4 Package handler

Enthält die Klassen, welche im MVC Pattern die Controllerfunktionen übernehmen.

ButtonListener	- Reagiert auf Benutzereingaben, die auf der Oberfläche vorgenommen werden
GraphListener	- Reagiert auf Benutzerinteraktionen mit dem Experimentgraphen - Stellt Methoden zum Selektieren von Knoten zur Verfügung (Knoten markieren, aktivieren etc.)
MenuListener	- Überwacht Auswahl der einzelnen Menüpunkte

5.5.5 Package gui

Hier befinden sich Klassen zur Erzeugung der Oberflächenobjekte sowie Dialogklassen. Das Hauptfenster setzt sich aus mehreren Einzelkomponenten zusammen und erleichtert somit das mögliche Umgestalten der Oberfläche.

MainFrame	- Hauptfenster
EMenuBar	- Menüleiste vom Hauptfenster inklusive der Einträge
EInfoPanel	- Unterer Teil vom Hauptfenster, dient der Knoteninformationsanzeige
GraphScrollPane	- Anzeigeplatz des Experimentgraphs, größter Teil des Fensters
PlugInPanel	- Für PlugIns verfügbarer Anzeigeteil
TempDocDialog	- Dialog um Dokumenten einem Template zuzuordnen
TimeDialog	- Dialog zur Eingabe der Ausfüllzeiten

5.5.6 Package main

Dieses Package enthält die Klasse *MainGraph*. Sie enthält die einzige *main()*- Methode, die zum Starten des Programms erforderlich ist.

5.6 Packages des Analyse-PlugIns

Das PlugIn wurde in mehrere Packages unterteilt. Es folgt eine kurze Beschreibung der jeweiligen Aufgaben sowie eine Betrachtung der eingesetzten Klassen.

5.6.1 Package gui

Hier befinden sich Klassen zum Erstellen der Oberfläche des PlugIns. Da nur ein begrenzter Bereich zur Verfügung steht wurde die Anzeige übersichtlich gehalten

PlugInAnzeige - Anzeigeplatz des PlugIns, beinhaltet alle Elemente zum Visualisieren der durch das PlugIn bereitgestellten Metriken

5.6.2 Package handler

Das Package beinhaltet die Klassen, welche die Controllerfunktionen übernehmen.

PlugInButtonListener - Reagiert auf Benutzereingaben der PlugIn Oberfläche

PlugInListener - Reagiert auf Benutzerinteraktionen mit dem Experimentgraphen

5.6.3 Package metrics

Hier befinden sich die Klassen des PlugIns mit denen die implementierten Metriken berechnet werden.

ExperimentAbschliessen - Enthält den Algorithmus zum Prüfen ob das Experiment abschließbar ist

ExperimentVergleich - Beinhaltet den Algorithmus zum Vergleichen zweier Experimente

ExperimentStandardmass - Berechnet das Standardmaß des aktuellen Experiments

5.6.4 Package main

Beinhaltet die Hauptklasse, mit der das PlugIn aufgerufen wird und instanziiert den Bereichs des Hauptprogramms, der dem PlugIn zur Verfügung steht.

5.7 Verzeichnisse des Programms

Es existieren die drei Verzeichnisse „*Experimente*“, „*Templates*“ und „*PlugIns*“, deren Bedeutung kurz erläutert wird.

5.7.1 Verzeichnis Experimente

Hier befindet sich die XML Datei des Experimentgraphs, die beim Programmstart ausgelesen und verarbeitet wird. Des Weiteren wird hier für jedes zu speichernde Experiment

ein Unterverzeichnis mit dem Namen des Experiments erstellt, indem die zugehörige XML Datei sowie experimentspezifische Dokumente abgespeichert werden.

5.7.2 Verzeichnis Templates

Sämtliche Templates, die der Experimentgraph enthält, werden hier abgelegt. Ein Herunterladen dieser Dokumente ist nur aus diesem Verzeichnis möglich.

5.7.3 Verzeichnis PlugIns

Hier befinden sich die einzelnen PlugIns, die aus dem Hauptprogramm geladen werden können. Nur wenn die PlugIn Schnittstelle implementiert wurde, ist ein korrektes Laden möglich.

6 Zusammenfassung und Ausblick

Das Ziel dieser Arbeit war es, ein Werkzeug zur Charakterisierung und Analyse von Software Engineering Experimenten zu konzipieren und zu realisieren. Mit diesem sollte die Planung und Analyse von Experimenten erleichtert werden.

Hierzu war zunächst eine theoretische Betrachtung von Experimenten erforderlich. So wurden in Kapitel 2 die Grundlagen von Experimenten mit spezieller Betrachtung des Software Engineering Bereichs erläutert. Ferner wurden die einzelnen Phasen des Experimentaufbaus erläutert und auf ein an dieser Universität durchgeführtes Experiment als Beispiel angewandt. Dieses Test-First Experiment diente als Grundlage der Arbeit. Das entwickelte Werkzeug enthält einen Experimentgraph, in dem sich das Test-First Experiment abbilden lässt sowie weitere Experimente planen lassen.

In Kapitel 3 wurden verschiedene mathematische Möglichkeiten der Graphenanalyse vorgestellt und mittels der GQM Methode spezielle Metriken für die Analyse und Charakterisierung von Experimenten aus dem Software Engineering Bereich entwickelt.

In Kapitel 4 und 5 wurden die Anforderungen an das Werkzeug aufgezeigt sowie die Realisierung im Detail betrachtet. Das Werkzeug bietet die Möglichkeit, Experimente miteinander vergleichen zu können und implementiert die in Kapitel 3 entwickelten Metriken zur Charakterisierung.

Zusammenfassend betrachtet wurde mit der Entwicklung dieses Werkzeugs das Ziel der Arbeit erreicht. Es liegt ein Softwareprodukt vor, mit dem Experimente geplant und analysiert werden können und sich das durchgeführte Test-First Experiment abbilden lässt.

6.1 Kritische Betrachtung der implementierten Metriken

Eine Validierung der eingesetzten Metriken konnte nur ansatzweise erfolgen, da außer dem Test-First Experiment keine weiteren Experimentdaten vorlagen. Des Weiteren konnte nicht genau untersucht werden, wie sinnvoll die durch die eingesetzten Metriken erhaltenen Aussagen in Hinblick auf die Praxistauglichkeit sind.

Speziell beim Vergleich von Experimenten konnte nur mit theoretisch erstellten Experimenten gearbeitet werden. Über das Standardmaß lässt sich auch erst eine genaue Aussage

treffen sobald mehrere Experimentdaten vorliegen und die ermittelten Standardmaße untereinander verglichen werden können, um die Aussagekraft dieser Metrik ermitteln zu können. Die implementierten Metriken sollen deswegen einen ersten Anhaltspunkt liefern und können, wenn weitere Experimentdaten vorliegen, entsprechend verfeinert werden.

6.2 Ausblick

Ausgehend von der vorliegenden Arbeit lassen sich noch folgende Bereiche näher betrachten:

- Der Experimentgraph kann anhand wachsender Erfahrungen mit Experimenten im Software Engineering erweitert werden. Dies beinhaltet zum einen die grundlegende Verfeinerung der Knoten im Experimentgraph, zum anderen könnten sie durch Hinweise ergänzt werden.
- Zur genaueren Ähnlichkeitsbestimmung kann eine Ähnlichkeitsmatrix entwickelt werden, in dem die Ähnlichkeitsbeziehungen zwischen sämtlichen Knoten spezifiziert werden können. Diese Aufgabe lässt sich nur durch sehr viel Erfahrung im Umgang mit Experimenten aus dem Software Engineering Bereich lösen, liefert damit aber präzisere Ergebnisse.

Literaturverzeichnis

[DeM82] T. DeMarco: Controlling Software Projects, Yourdon Press, New York, USA 1982.

[Fen91] N. Fenton: Software Metrics - A Rigorous Approach, Chapman & Hall, London 1991.

[Pre01] L. Prechelt: Kontrollierte Experimente in der Softwaretechnik, Springer, Berlin 2001.

[Bas86] V.R. Basili: Experimentation in Software Engineering, 1986.

[Man01] E. Manso: Experimentation in Software Engineering, 2001.

[Pfl95] S. Pfleger: Experimental Design and Analysis in Software Engineering, 1995.

[Bas95] V.R. Basili: The Role of Experimentation in Software Engineering, 1995.

[BaRo88] V.R. Basili, H. Rombach: The TAME Project: Towards Improvement-Oriented Software Environments, 1988.

[WRH+00] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, A. Wesslen: Experimentation in Software Engineering, Kluwer Academic Publishers, London 2000.

[FlSc04] T. Flohr, T. Schneider: An XP Experiment with Studends - Setup and Problems, Hannover 2004.

[Bec00] K. Beck: Extreme Programming Explained, Addison-Wesley, 2000.

[TSS+95] E. Tammer, K. Steinhöfel, S. Schönherr, D. Matuschek: Anwendung des Konzepts der Strukturellen Ähnlichkeit zum Fallvergleich mittels Term- und Graph-Repräsentationen, 1995.

[Wir98] J. Wirth: Die Verwendung von unterschiedlich strukturierten Analogien beim Problemlösen, 1998.

[BaTa94] B. Bartsch-Spörl, E. Tammer: Graph-based Approach to Structural Similarity, 1994.

[Knö71] W. Knödel: Ein Verfahren zur Feststellung der Isomorphie von endlichen, zusammenhängenden Graphen, 1971.

[Zel75] B. Zelinka: On a certain distance between isomorphism classes of graphs, 1975.

[KaSo82] F. Kaden, F. Sobik: Beiträge zur angewandten Graphentheorie, 1982.

[WoYo85] A.K.C. Wong, M. You: Entropy and distance of random graphs with application to structural pattern recognition, 1985.

[BoLi87] I. Borg, J. Lingoes: Multidimensional Similarity Structure Analysis, 1987.

[Sch00] K. Schädler: Die Ermittlung struktureller Ähnlichkeit und struktureller Merkmale bei komplexen Objekten, 2000.

[BCR94] V.R. Basili, G. Caldiera, H. Rombach : Goal question metric paradigm, in Encyclopedia of Software Engineering, 1994.

[IEE98] IEEE Standard 1061 for a Software Quality Metrics Methodology, IEEE Standard Glossary of Software Engineering Terminology, 1998.

[Qui05] L. Quin: Extensible Markup Language http://www.w3.org/XML/, 2005.

[Ald05] G. Alder: Java Graph Visualization and Layout http://www.jgraph.com/, 2005.

[Nav05] B. Naveh: JGraphT http://jgrapht.sourceforge.net/, 2005.

[FRF03] M. Fowler, D. Rice, M. Foemmel: Patterns für Enterprise Application-Architekturen, 2003.

[GHJ98] E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software: Addison-Wesley, 1998.

Abbildungsverzeichnis

Anhang

A.1 XML Schema des Experimentgraphen

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:element name="graph">
            <element name="node" type="node_type" minOccurs="2"
                maxOccurs="unbounded" />
            <complexType name="node_type">
                  <attribute name="id" type="ID" use="required" />
                  <attribute name="xkoor" type="integer" use="required" />
                  <attribute name="ykoor" type="integer" use="required" />
                  <all>
                  <element name="name" type="string" />
                  <element name="description" type="string" />
                  <element name="predecessor" type="predecessor_type"
                        minOccurs="0" maxOccurs="unbounded"/>
                  <element name="template" type="template_type"
                        minOccurs="0" maxOccurs="unbounded"/>
                  </all>
            </complexType>
            <complexType name="predecessor_type" type="string">
                  <attribute name="knoten" type="IDREF"
                        use="required" />
            </complexType>
            <complexType name="template_type" type="string">
                  <attribute name="expTime" type="integer"
                        use="required" />
                  <attribute name="numberOfDataCollection" type="integer"
                        use="required" />
                  <attribute name="numberOfPerson" type="integer"
                        use="required" />
            </complexType>
      </xs:element>
</xs:schema>
```

A.2 XML Schema eines Experiments

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:element name="graph">
            <element name="subjects" type="integer"/>
            <element name="dataCollection" type="integer"/>
            <element name="node" type="node_type" minOccurs="1"
                  maxOccurs="unbounded" />
            <complexType name="node_type">
                  <attribute name="id" type="ID" use="required" />
                  <all>
                        <element name="document" type="document_type"
                              minOccurs="0" maxOccurs="unbounded"/>
                  </all>
            </complexType>
            <complexType name="document_type" type="string">
                  <attribute name="attendedTime" type="integer"
                        use="required" />
                  <attribute name="template" type="string"
                        use="required" />
            </complexType>
      </xs:element>
</xs:schema>
```