Universität Hannover
Fachgebiet Software Engineering
Institut für Angewandte Systeme
Fachbereich Informatik

Entwurf und Implementierung eines Werkzeuges zur Abschätzung der Komplexität von Experimenten im Software Engineering

Bachelorarbeit

im Studiengang Informatik

von

Ursula Beck

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

Hannover, 01. September 2005

Zusammenfassung

Im Software Engineering erfolgt der Wissenszuwachs in erster Linie durch Empirie, da die betrachteten Prozesse zu komplex für mathematische Analysen sind. Eine Forschungsmethode, die ein großes Maß an Kontrolle und damit ein hohes Maß an Vertrauen in ihre Ergebnisse bietet, ist das Experiment. Experimente erfordern eine sorgfältige und detaillierte Planung, um diese Kontrolle über die Beobachtungsbedingungen zu erzielen.

Die vorliegende Arbeit befasst sich mit der Entwicklung eines Werkzeuges, das die Planung von Experimenten unterstützt und Metriken zur Verfügung stellt mit denen die Komplexität von Experimenten abgeschätzt werden kann.

Erklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und ohne fremde Hilfe verfasst habe und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe.

Hannover, den 01.September 2005

Ursula Beck

Inhaltsverzeichnis

1 Einleitung

1.1 Problemstellung, Motivation

In den letzten Jahren hat sich die Erkenntnis, dass Experimente eine für das Software Engineering relevante Forschungsmethode sind, immer mehr durchgesetzt [1].

Das Software Engineering beschäftigt sich unter anderem mit der Suche nach Möglichkeiten, den Entwicklungsprozess von Software sowie die zu erstellenden Produkte zu verbessern. Mit Hilfe von empirischen Studien werden Methoden, Techniken und Werkzeuge auf ihre Leistungsfähigkeit hin überprüft und weiterentwickelt. Hierzu dienen eine Reihe von unterschiedlichen Forschungsmethoden, wie z.B. Fall-/Feldstudien oder kontrollierte Experimente [2].

Kontrollierte Experimente sind eine klassische Methode der Forschung, die vor allem in den naturwissenschaftlichen Fächern zum Einsatz kommt. Von Experimenten wird behauptet, dass sie die einzige Möglichkeit darstellen, Kausalzusammenhänge aufzuzeigen, da man beim Experimentaufbau gezielt bestimmte Faktoren manipulieren oder konstant halten und so deren Relevanz auf das Ergebnis untersuchen kann.

Im Software Engineering ist es wichtig zu wissen, wann und wie ein Entwicklungsmodell funktioniert, wo seine Vorteile aber auch Grenzen liegen. Trotz dieser offensichtlichen Notwendigkeit des Einsatzes von Experimenten hat diese Forschungsmethode sich im Software Engineering noch nicht durchgesetzt. Eine Studie von Tichy [3] ergab, dass 40% von Papern, die eine empirische Auswertung benötigt hätten, sogar ganz auf diese verzichten.

Ein möglicher Grund für diesen Mangel an experimenteller Arbeit ist, dass es häufig schwer fällt, geeignete Fragestellungen zu finden, die sich mit Hilfe von kontrollierten Experimenten in einem angemessenen Zeit- und Finanzierungsrahmen beantworten lassen. So findet man in der Forschungsliteratur vor allem Experimente, die sich mit der Lesbarkeit und Verständlichkeit von Dokumenten beschäftigen oder aber solche, die die Effektivität von Inspektionen beantworten, aber kaum welche zum Thema der Objektorientierung.

Am Institut für Software Engineering der Universität Hannover wurde im Wintersemester 2004/05 ein Experiment mit Studenten durchgeführt, welches die Effektivität von Test-First im Vergleich zum klassischen Testen im Hinblick auf die Entwicklungsgeschwindigkeit und Testabdeckung untersuchte [4].

Schon bei der Planung und später auch im Verlauf des Experiments (im Folgenden als Test-First Experiment bezeichnet) stellte sich heraus, dass die Durchführung eines Experiments weitaus umfangreicher ist als ohnehin schon angenommen. Da man auch in der Literatur nur wenig Erfahrung und Unterstützung fand, entstand die Idee der Entwicklung eines Werkzeuges, das beim Entwurf, bei der Durchführung und letztlich auch bei der Analyse von Experimenten hilft.

1.2 Ziel dieser Arbeit

Diese Arbeit beschäftigt sich mit dem Konzept und der Umsetzung eines solchen Werkzeuges, das Unterstützung bei Planung, Durchführung und Auswertung von Experimenten liefert.

Zusätzlich soll dieses Werkzeug eine Möglichkeit bieten, die Komplexität von durchgeführten aber auch in der Planung befindlichen Experimenten abzuschätzen. Zu diesem Zweck ist es erforderlich, den Komplexitätsbegriff für Experimente zu definieren und geeignete Metriken zu entwickeln, mit deren Hilfe sich diese Komplexität messen lässt.

J. Eggermann, der ebenfalls an der Realisierung dieses Werkzeuges beteiligt ist, beschäftigt sich mit dem Schwerpunkt der Charakterisierung und des Vergleichs von Experimenten.

1.3 Gliederung

Die nachfolgenden Kapitel dieser Arbeit gliedern sich wie folgt:

Im zweiten, theoretischen Teil werden Grundlagen zu Experimenten im Software Engineering erarbeitet. Es wird darauf eingegangen, für welche Fragestellungen sich Experimente eignen und welche Vor- und Nachteile sie gegenüber anderen Forschungsmethoden bieten. Der grundlegende Aufbau von Experimenten wird vorgestellt und auf das Test-First Experiment abgebildet. Außerdem werden allgemeine Grundlagen zum Komplexitätsbegriff sowie zu Metriken erläutert.

Das dritte Kapitel beschreibt das Vorgehen bei der Anforderungserhebung und enthält technische, funktionale und Qualitätsanforderungen sowie Anforderungen an die Benutzeroberfläche des zu entwickelnden Werkzeuges. In diesem Kapitel sowie in dem Entwurfskapitel wird auf bewährte Dokumentationsmethoden im Software Engineering [5] - wie beispielsweise Use Cases - zurückgegriffen.

Im vierten Kapitel dieser Arbeit wird der Komplexitätsbegriff für Experimente, wie er in dem zu entwickelnden Werkzeug verstanden wird, definiert. Anschließend werden mit Hilfe der Goal-Question-Metric-Methode [6] eine Reihe von Metriken entwickelt, mit denen sich diese Komplexität messen lässt.

Im fünften Kapitel werden die grundlegenden Entwurfsentscheidungen, die beim Konzept des Werkzeuges entstanden, vorgestellt. Dabei wird sowohl auf die Datenspeicherung als auch auf die Gestaltung der Oberfläche sowie die Strukturierung des Quellcodes eingegangen.

In einer abschließenden Diskussion wird das gesamte Vorgehen noch einmal zusammengefasst. Darüber hinaus werden die entwickelten Metriken kritisch betrachtet und mögliche Änderungen und Weiterentwicklungen vorgestellt.

2 Grundlagen

2.1 Was sind kontrollierte Experimente?

Kontrollierte Experimente sind eine empirische Forschungsmethode mit deren Hilfe Kausalzusammenhänge zwischen gemachten Beobachtungen und deren Ursachen aufgedeckt werden können. Sie ermöglichen es, Theorien zu bestätigen oder zu widerlegen und die Effektivität von Methoden und Techniken zu untersuchen [7].

> *„Ein kontrolliertes Experiment ist eine Studie bei der alle voraussichtlich für das Ergebnis relevanten Umstände konstant gehalten werden, mit Ausnahme von einem oder wenigen, die den Gegenstand der Untersuchung bilden (Experimentvariablen). Die Beobachtungen für verschieden gezielt ausgesuchte Werte der Experimentvariablen werden miteinander verglichen, um so zu reproduzierbaren Aussagen zu kommen, die eine vor dem Experiment definierte Experimentfrage beantworten. Die Experimentfrage ist ein genügend enger Aspekt einer relevanten Forschungsfrage.“* [8]

Durch den hohen Grad an Kontrolle über die Versuchsbedingungen lassen sich die Ergebnisse und deren Ursachen besonders gut verstehen.

2.2 Terminologie von Experimenten

In diesem Kapitel werden spezifische Begriffe, die im Zusammenhang mit Experimenten eine Rolle spielen kurz vorgestellt und erläutert. Die Ausdrücke „Experiment“ und „kontrolliertes Experiment“ werden dabei synonym verwendet, auf eine Differenzierung wird verzichtet, da sie für diese Arbeit nicht von Bedeutung ist.

In einem Experiment werden *Subjekte* und *Objekte* unterschieden. Ein *Objekt* ist der Gegenstand der Untersuchung (z.B. ein Dokument oder eine Technik), als *Subjekte* bezeichnet man die Versuchspersonen.

Es gibt drei verschiedene Typen von Variablen - *abhängige Variablen, unabhängige Variablen* und *zu kontrollierenden Variablen* - die in jedem Experiment vorkommen.

Abhängige Variablen sind diejenigen, deren Werte im Experiment gemessen werden, sie werden häufig auch als *Experimentvariablen* bezeichnet.

Unabhängige Variablen sind all jene Faktoren, die gezielt manipuliert werden, um ihren Einfluss auf die *abhängigen Variablen* zu beobachten.

Alle anderen Faktoren, die im Experiment konstant gehalten werden (müssen), bezeichnet man als die *zu kontrollierenden Variablen* oder *Störvariablen*.

Die *innere Gültigkeit* eines Experiments ist ein Maß dafür, wie gut die Kontrolle über die *Störvariablen* tatsächlich ist.

> *„Die innere Gültigkeit eines kontrollierten Experiments ist der Grad, in dem die Änderung in den Werten der abhängigen Variablen tatsächlich wie gewünscht nur auf Änderungen in den unabhängigen Variablen zurückzuführen sind, d.h wie gut letztlich die relevanten Störvariablen kontrolliert wurden.“* [8]

Die *innere Gültigkeit* ist ein wichtiges Maß, wenn es um die Aussagekraft von Experimenten geht. Es gibt mehrere Faktoren, welche die innere Gültigkeit bedrohen. Die sind unter anderem *Reifung*, *Instrumentation*, *Sterblichkeit* oder *Auswahleffekte.*

Die *Reifung* betrifft Veränderungen des Verhaltens von Versuchspersonen. Dies können Ermüdungserscheinungen bei fortschreitender Experimentdauer aber auch Lern- und Reihenfolgeeffekte sein.

Veränderungen im Verhalten der Experimentatoren oder am Experimentaufbau werden unter dem Begriff der *Instrumentation* zusammengefasst.

Als *Sterblichkeit* bezeichnet man das freiwillige Ausscheiden von Versuchspersonen aus einem laufenden Experiment.

Die so genannten *Auswahleffekte* betreffen die Kriterien, welche bei der Einteilung der Versuchspersonen in Gruppen zum Tragen kommen. In einem idealen Experiment sollte die Gruppenbildung durch *Randomisieren* erfolgen, d.h. die Personen werden zufällig aufgeteilt. Häufig ist ein solches Vorgehen aufgrund charakteristischer Personeneigenschaften, Erfahrungen oder benötigter Kenntnisse für eine Aufgabe nicht möglich. Werden Versuchspersonen nach bestimmten Kriterien in Gruppen aufgeteilt, so können diese Kriterien unter Umständen Auswirkungen auf das Ergebnis haben und damit die *inner Gültigkeit* bedrohen.

Eine Alternative zum *Randomisieren* stellt das Verfahren der *Blockung* dar. *Blockung* bedeutet, dass die Versuchspersonen zunächst aufgrund verschiedener, nicht manipulierbarer Eigenschaften (z.B. Berufserfahrung) in Klassen eingeteilt werden. Auf Grundlage dieser Klassenbildung erfolgt dann die *Randomisierung*.

Neben dem Begriff der *inneren Gültigkeit* existiert auch der Begriff der *äußeren Gültigkeit*.

> *„Die äußere Gültigkeit eines kontrollierten Experiments ist der Grad, in dem sich seine Resultate korrekt auf andere Anwendungsfälle übertragen lassen - insbesondere auf solche, die in der Praxis häufig vorkommen.*“ [8]

Die Grundlage eines jeden Experiments ist eine definierte Fragestellung, die *Experimentfrage*, welche sich aus einem bestimmten Forschungsschwerpunkt ergibt. Auf Basis der Experimentfrage werden *Hypothesen* aufgestellt, die es im Experiment zu beweisen oder zu widerlegen gilt.

2.3 Vergleich von Experimenten im Software Engineering mit Experimenten in naturwissenschaftlichen Fächern

Experimente haben sich mittlerweile zu einer Forschungsmethode entwickelt, die in vielen verschiedenen Wissenschaftsgebieten zum Einsatz kommt. Der Ursprung des klassischen Experiments liegt allerdings in der Naturwissenschaft. Vor allem in der Physik, Chemie und Medizin sind Experimente eine anerkannte und häufig genutzte Forschungsmethode. Die Methodik des Experimentierens in diesen Gebieten lässt sich aber nicht ohne weiteres auf das Software Engineering übertragen.

Das Ziel der Physik ist es, Vorgänge in der Natur durch Gesetze zu beschreiben. Grundlage von Experimenten sind daher häufig theoretisch entwickelte, mathematische Formel oder Gesetzmäßigkeiten. Im Software Engineering hingegen will man Eigenschaften von Entwicklungsprozessen, bzw. Vor- oder Nachteile bestimmter Methodiken oder Werkzeugen experimentell untersuchen. Die dabei aufgestellten Hypothesen lassen sich meistens aber nicht als mathematische Gesetze formulieren.

Die Chemie untersucht die Eigenschaften von Stoffen und Verbindungen. Ihre zwei Hauptaufgaben sind die Analyse und die Synthese von Stoffen. Diese zwei Anwendungen lassen sich auf das Software Engineering übertragen.

Als Analyse kann man die Untersuchung eines fertig gestellten Stück Softwares oder Dokuments betrachten während die Synthese dem Entwicklungsprozess entspricht.

Betrachtet man diese Aufgaben und ihren Zweck in den beiden Wissenschaftsgebieten jedoch genauer, so stellt man Unterschiede in ihrem Bestreben fest.

Während die Analyse in der Chemie die Zusammensetzung von Stoffe untersucht, also danach fragt, welche Elemente in welcher Form und zu welchem Anteil in einem Stoff enthalten sind, besteht das Anliegen der Analyse von Softwareprodukten darin, bestimmte Eigenschaften wie Testbarkeit, Verständlichkeit oder Wartbarkeit dieses Produkts zu benennen. Diese Eigenschaften lassen sich zwar mit Hilfe von Metriken messen und vergleichen, es sind aber keine absoluten Angaben z.B. im Hinblick auf die Wartbarkeit eines Programms möglich.

Die Synthese in der Chemie untersucht, welche Elemente und Verbindungen, in welcher Menge auf was für eine Art verbunden werden müssen, um neue Stoffe herzustellen. Die Forschung im Software Engineering dagegen beschäftigt sich eher mit der Frage, welche Methoden und Werkzeuge den Menschen bei der Entwicklung helfen können. Diese Vor- oder Nachteile bestimmter Werkzeuge und Techniken variieren jedoch, sie hängen vom Umfeld und von den Personen, die sie benutzen, ab.

Während man in der Chemie nach einem erfolgreichen Experiment z.B. behaupten kann: „immer wenn ich die Menge x des Stoffes A mit der Menge y des Stoffes B mische erhalte ich C“ sind analoge Aussagen wie „der Einsatz des Werkzeuges z verdoppelt die Entwicklungsgeschwindigkeit“ im Software Engineering nicht möglich.

Ein weiterer Unterschied zwischen Experimenten in naturwissenschaftlichen Fächern und im Software Engineering besteht in der Menge der zur Auswertung zur Verfügung stehenden Datensätze.

Viele Versuche in der Chemie oder Physik laufen automatisiert ab, mit ein paar Handgriffen lassen sich gezielt bestimmte Faktoren variieren und mögliche Einflüsse dieser Veränderungen auf das Ergebnis beobachten. Durch diese Automatisierung erhält man einen großen Satz an Datenmengen. Wenn wie im Software Engineering hingegen eine Entwicklungsmethode, die von Menschen genutzt wird, der Gegenstand der Untersuchung ist, so muss man sich fast immer mit sehr viel kleineren Datenmengen zu Frieden geben. Dieser Mangel an Daten erschwert die Auswertung im Hinblick auf Aussagekraft und äußere Gültigkeit der Ergebnisse.

Über den menschlichen Faktor könnte man versuchen, eine Verbindung zwischen der experimentellen Arbeit im Software Engineering und in der Medizin herzustellen.

In der Medizin werden z.B. bestimmte Therapiemethoden untersucht. Anders als im Software Engineering sind die Versuchspersonen bei diesen Experimenten jedoch keine aktiven Teilnehmer. Vielmehr lassen sie eine Behandlung über sich ergehen und obliegen dabei der Beobachtung der Experimentatoren.

Im Software Engineering haben Versuchspersonen durch ihre Erfahrungen, dem daraus resultierenden Arbeitsverhalten sowie persönlichen Eigenschaften, einen weitaus größeren Einfluss auf die Ergebnisse von Experimenten.

Ein weiterer Unterschied besteht in der Art der durchzuführenden Experimente. Eine häufig verwendete Technik in der Medizin ist die Verblindung oder Doppelverblindung. Ein derartiges Vorgehen, bei dem die Versuchspersonen (und Experimentatoren) nicht darüber informiert sind, welcher konkreten Behandlungsmethode sie unterzogen sind, ist im Software Engineering kaum möglich.

Die hier herausgearbeiteten Unterschiede tragen vermutlich dazu bei, dass Experimente im Software Engineering immer noch verhältnismäßig wenig zum Einsatz kommen.

Es ist erforderlich, eine eigene Methodik für die Planung, Durchführung und Auswertung zu entwickeln. Dabei sollte man soweit es geht, verschiedene, erprobte Verfahren aus den oben genannten Gebieten übernehmen und sie an die gegebene Umstände im Software Engineering anpassen. Dies ist ein aktueller Forschungsschwerpunkt im Bereich des experimentellen Software Engineerings [1, 9].

2.4 Kontrollierte Experimente im Vergleich mit anderen Forschungsmethoden

Die Auswahl einer Forschungsmethode beim empirischen Arbeiten beinhaltet die Überlegung, welche Alternativen zur Verfügung stehen und welche Vor- und Nachteile die einzelnen Methoden gegenüber anderen bieten.

Im Software Engineering gibt es eine Reihe von Techniken [2], welche zu Forschungszwecken genutzt werden.

Fall- und Feldstudien sind dabei zwei Methoden, die besonders häufig zum Einsatz kommen. An dieser Stelle werden deshalb die Vor- und Nachteile von kontrollierten Experimenten gegenüber diesen beiden Methoden herausgestellt.

Fallstudien werden ähnlich wie Experimente in einer künstlichen Umgebung durchgeführt. Der Unterschied besteht in der geringeren Kontrolle über die unabhängigen Variablen und Störvariablen. Es wird kein Wert darauf gelegt, diese konstant zu halten. So liefern Fallstudien zwar ebenfalls Aussagen über Eigenschaften und Effizienz von Werkzeugen und Techniken, können aber im Gegensatz zu Experimenten keine gesicherten Angaben zu deren Ursachen machen. Der Vorteil von Fallstudien gegenüber Experimenten ist ihre Einfachheit. Es müssen viel weniger Faktoren berücksichtigt werden. Dies wirkt sich natürlich auch positiv auf die Kosten aus.

Fallstudien sollten verwendet werden, wenn es zunächst darum geht, bestimmte Eigenschaften von Methoden herauszuarbeiten. Darauf aufbauend können dann Experimente eingesetzt werden, um die Ursachen für diese Eigenschaften zu finden.

Eine Feldstudie ist eine systematische wissenschaftliche Beobachtung unter natürlichen Bedingungen, d.h. Feldstudien ziehen ihre Beobachtungen aus realen Projekten. In einem noch geringeren Maße als bei Fallstudien sind hier gesicherte Aussagen über Ursache und Wirkung möglich.

In ein reales Projekt hat man als Experimentator viel weniger Eingriffsmöglichkeiten, die Reproduzierbarkeit solcher Untersuchungen ist praktisch gleich null. Der große Vorteil dieser Methode ist, dass die erzielten Ergebnisse zumindest in einem konkreten Anwendungsfall schon einmal zutrafen. Dies muss bei Fallstudien oder Experimenten, welche in einer künstlichen Umgebung stattfinden, erst noch bewiesen werden.

Feldstudien dienen in erster Linie dazu, die Verallgemeinerbarkeit von Ergebnissen aus Experimenten nachzuweisen.

Die drei genannten Forschungsmethoden zusammen bilden die Grundlage des empirischen Software Engineerings. In Abhängigkeit von der zu beantwortenden Fragestellung sollte man abwägen, welche von ihnen die besten Ergebnisse liefern. Gerade bei größeren Forschungsfragen ist es sinnvoll, Untersuchungen mit mehreren verschiedenen Methoden durchzuführen.

	Fallstudie	Feldstudie	Experiment
Aufwand für Forscher	-/+	+	-
Aufwand für Versuchspersonen	+	+	-
Kontrolle, Eingriffsstärke	-/+	-	+
Reproduzierbarkeit	-/+	-	+
Verallgemeinerbarkeit der Ergebnisse	-/+	-/+	+

+ : Vorteile gegenüber den anderen Methoden

- : Nachteile gegenüber den anderen Methoden

Abb. 1: Vergleich von Fallstudie, Feldstudie und Experimenten

2.5 Der Aufbau von Experimenten

In [1, 9] existieren Ansätze, ein Framework für den Aufbau von Experimenten zu definieren. Ein solches Framework soll das Design von Experimenten erleichtern, indem es wichtige Kriterien und Entscheidungspunkte benennt, die beim Entwurf eines Experiments zu beachten sind. Ein weiterer Nutzen liegt in der Möglichkeit, Experimente mit Hilfe eines Frameworks zu klassifizieren und so Vergleichsgrundlagen zu schaffen.

Die Ansätze zur Definition eines solchen Frameworks variieren zum einen in der Anzahl der Phasen, in die ein Experiment unterteilt wird, zum anderen in der Auswahl der Kriterien, die zur Charakterisierung der einzelnen Phasen herangezogen werden.

So findet man bei Basili [1]beispielsweise einen vierstufigen Ansatz, welcher die Kategorien „definition“, „planning“, „operation“ und „interpretation“ enthält, während Pfleeger [9]sein Konzept auf sechs unterschiedliche Phasen („conception“, „design“, „preparation“, „execution“, „analysis“, „dissemination and decision-making“) stützt. Die Unterschiede dieser verschiedenen Ansätze sind allerdings nicht so gravierend, dass man von völlig verschiedenen Frameworks sprechen muss.

In dieser Arbeit soll der Entwurf von Basili [1] zur grundlegenden Beschreibung des Aufbaus von Experimenten herangezogen werden.

Das Test-First-Experiment, welches den Anstoß zur Entwicklung dieses Experiment-Planungwerkzeugs gegeben hat, lässt sich diesem Experimentprozess zuordnen.

Im Folgenden werden die einzelnen Phasen dieses Konzepts mit ihren grundlegenden Entscheidungskriterien und Arbeitsschritten vorgestellt

2.5.1 Definition

In dieser ersten Phase des Experimentprozess wird die Experimentfrage benannt und festgehalten.

Basili schlägt vor, diese Experimentfrage mit Hilfe des Goal-Question-Metric-Goal-Templates [6] zu spezifizieren. Dieses Template wurde ursprünglich zur Definition von Messzielen beim Finden von Metriken entwickelt. Es beinhaltet fünf Parameter, die mit Werten belegt werden müssen. Diese Parameter werden hierarchisch zerlegt, um eine möglichst detaillierte Experimentfrage zu erhalten. Die Zerlegung wird mit Hilfe von so genannten Klassifizierungsbäumen vorgenommen.

Die fünf Parameter im Einzelnen:

- ***Objekt der Studie***: was wird untersucht, z.B. ein Prozess, Produkt, …
- ***Zweck***: mit welcher Intention wird die Untersuchung vorgenommen;
  charakterisieren, evaluieren, verbessern, …
- ***Fokus***: mit welcher Sichtweise wird das Objekt untersucht;
  z.B. Fehler finden, Zuverlässigkeit, …
- ***Perspektive***: für wen sind die Informationen nützlich
- ***Kontext***: Spezifikation der Umgebung, Störvariablen,…

Die folgende Abbildung zeigt einen Ausschnitt aus einem Klassifizierungsbaum, wie er für das ***Objekt der Studie*** aussehen könnte.

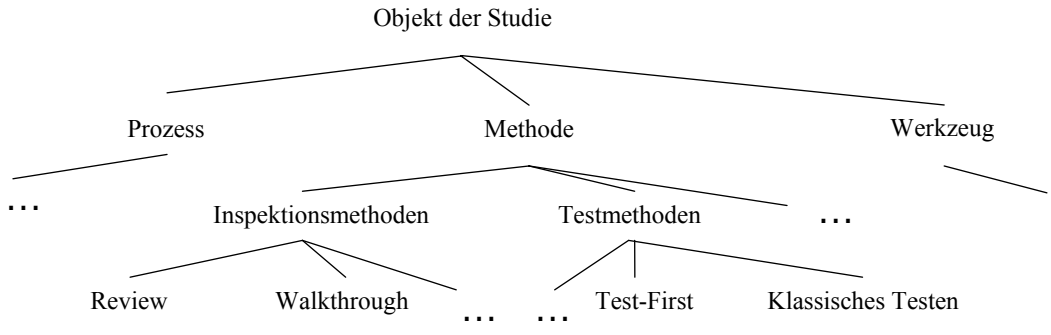

Abb. 2: Klassifizierungsbaum für das Objekt der Studie

Häufig werden beim Entwickeln von Experimentfragen mit Hilfe dieser Methode nur die ersten drei bis vier Parameter spezifiziert. Die Definition des Kontextes ist relativ umfangreich, sie wird erst nach und nach im Laufe der zweiten Phase des Experimentprozesses entwickelt.

Eine Experimentfrage, die mit Hilfe des Goal-Question-Metric-Goal-Templates entwickelt wurde, könnte wie folgt lauten:

> *"Analyze reading techniques(Objekt) to evaluate(Zweck) their ability to detect defects(Fokus) on natural-language requirements documents"* [1]

Ein positiver Nebeneffekt der Hierarchiebildung bei der Spezifikation der Faktoren ist die Schaffung einer Vergleichsgrundlage von Experimenten anhand ihrer Experimentfragen. Um einen solchen Vergleich zu ermöglichen muss ein einheitliches Model von Klassifizierungsbäumen für alle fünf Parameter geschaffen werden. Das Erstellen dieser Klassifizierungen könnte ein Schwerpunkt für weitere Arbeiten auf dem Gebiet des experimentellen Software Engineerings sein.

Die Definitionsphase umfasst also in erster Linie das Festlegen der Experimentfrage. Diese stellt allerdings das Hauptkriterium für die weitere Planung dar. Deshalb sollte die Formulierung der Experimentfrage mit großer Sorgfalt erfolgen.

2.5.2 Planung

Die Planungsphase beinhaltet das Aufstellen von Hypothesen, die Spezifikation des Versuchsaufbaus sowie das Erstellen eines Zeitplans über den Ablauf des Experiments. Der Versuchsaufbau wird durch eine Menge von Faktoren charakterisiert, die wichtigsten werden im Folgenden kurz erläutert.

Mit dem Festlegen der Hypothesen, die mit Hilfe des Experimentes bewiesen werden sollen, werden die unabhängigen Variablen sowie die Störvariablen identifiziert. Beim Aufstellen von Hypothesen muss bedacht werden, welche Daten zu ihrer Auswertung benötigt werden, ob es möglich ist diese Daten zu beschaffen und mit Hilfe von welchen Metriken oder statistischen Methoden die Auswertung erfolgen soll. Hierbei muss auch der zeitliche Rahmen des Experiments berücksichtigt werden, denn es bringt keinen Nutzen, am Anfang der Planung eine Reihe von Hypothesen aufzustellen, die am Ende aus zeitlichen Gründen oder aus Mangel an Daten nicht ausgewertet werden können.

Einen wichtigen Punkt bei der Planung stellt der Entwurf der Aufgabenstellung dar. Die Aufgabe muss die Experimentfrage beantworten können, Zeitrahmen, Wissen und Erfahrung der Versuchspersonen, sowie die Möglichkeiten, genügend relevante Daten aus den Ergebnissen zu ziehen, berücksichtigen.

Ein weiteres Entscheidungskriterium in der Planungsphase ist die Auswahl der Versuchspersonen. Es muss überlegt werden, über welche Kenntnisse und Erfahrungen die Personen verfügen müssen/dürfen und mit wie vielen Versuchspersonen das Experiment durchgeführt werden soll. Darüber hinaus müssen vorhandene Ressourcen, wie beispielsweise Räumlichkeiten und Arbeitsplätze berücksichtigt werden. Auch die Anzahl der Experimentatoren, sowie die Zeit, die diese in das Experiment investieren wollen und können, müssen bedacht werden.

Für die meisten Experimente ist eine Einteilung der Versuchspersonen in eine Experiment- und eine Kontrollgruppe oder auch in mehrere verschiedene Gruppen erforderlich. Hierbei ist zu überlegen, ob dem eigentlichen Mittel der Randomisierung zum Einteilen der Gruppen vielleicht eine Blockung vorzuziehen ist. Ist dies der Fall, können z.B. Fragebögen oder kurze Tests helfen, die Blockung der Teilnehmer vorzunehmen

Schulungen sind ein weiteres Thema über das man sich in der Planungsphase Gedanken machen muss. Eventuell müssen die Teilnehmer den Umgang mit einer bestimmten Technik oder einem Werkzeug lernen. In diesem Fall müssen entsprechende Schulungsmaterialien entwickelt werden.

Zusätzlich ist es in dieser Phase erforderlich, die so genannten Störvariablen zu identifizieren und Maßnahmen festzulegen, wie diese konstant gehalten werden können. Hierzu zählen z.B. die Gestaltung des Arbeitsplatzes der Versuchspersonen oder die Auswahl der Hilfsmittel, die zur Lösung der Experimentaufgabe benutzt werden dürfen.

Ein weiteres Entscheidungskriterium, welches sich sowohl aus der Experimentfrage als auch aus den zu ihr entwickelten Hypothesen ergibt, ist die Wahl von Datenerhebungsmethoden. Hier bieten sich z.B. Logfiles zum Auswerten von Quellcode, Fragebögen zur Erfahrungssammlung oder das Beobachten der Versuchspersonen in Hinblick auf ihr Verhalten an.

Des Weiteren muss überlegt werden, mit welchen Methoden die gesammelten Daten ausgewertet werden. Eine Möglichkeit, die Auswertungsverfahren im Hinblick auf die zu beweisenden Hypothesen auswählen, bietet die Goal-Question-Metric-Methode.

In der Durchführungsphase wird eine Reihe von Dokumenten benötigt werden (zur Aufgabenstellung, Schulungen, Datenerfassung,…), welche bereits in der Planungsphase entworfen und gesammelt werden sollten.

Im Folgenden sind die wesentlichen Arbeitsschritte und Entscheidungskriterien der Planungsphase noch einmal zusammengefasst:

- Hypothesen aufstellen
- Aufgabenstellung und zugehöriges Arbeitsmaterial entwickeln
- Störvariablen identifizieren
- Versuchspersonen auswählen
- Datenerhebungsmethoden und Auswertungsverfahren festlegen
- benötigte Dokumente entwickeln/sammeln

2.5.3 Durchführung

Die Durchführungsphase sollte, wenn möglich, mit einem oder mehreren Pilottests beginnen. Dafür werden Personen benötigt, die später nicht am eigentlichen Experiment als Versuchspersonen teilnehmen. Derartige Pilottests können helfen, Mängel am Versuchsaufbau, an Formulierungen in Aufgabenstellungen oder Fragebögen oder andere Probleme aufzudecken und diese vor dem eigentlichen Experimentdurchlauf zu beseitigen.

Die eigentliche Experimentdurchführung beginnt mit der Vorbereitung der Versuchspersonen. Die in der Planungsphase entwickelten Schulungsmaterialien kommen hierbei zum Einsatz,

Anschließend wird die Aufgabenstellung ausgegeben. Die Experimentatoren haben in dieser Phase für einen reibungslosen Ablauf zu sorgen, und mögliche Faktoren, die die innere Gültigkeit des Experiments beeinträchtigen könnten auszuschalten.

Ein weiterer Schwerpunkt liegt in der Datenerhebung.

Die Dokumentation der Arbeitsschritte und Experimentkriterien ist in allen Phasen des Experiments von großer Bedeutung. Vor allem in der Durchführungsphase sollten alle Besonderheiten, die eventuell das Ergebnis beeinflussen könnten, sowie Abweichungen vom Experimentplan schriftlich festgehalten werden.

Die wesentlichen Kriterien und Arbeitsschritte der Durchführungsphase noch einmal zusammengefasst:

- Pilottests (falls möglich)
- Teilnehmer vorbereitet
- Ablauf überwachen
- Daten sammeln

2.5.4 Interpretation

Nach Beendigung der Durchführungsphase müssen die gesammelten Daten zunächst auf Vollständigkeit und Konsistenz überprüft werden.

Anschließend folgt die Analysephase. Hierbei ist es wichtig, die Auswertung der Daten auf die am Anfang des Experiments aufgestellten Hypothesen zu beziehen. Aus den Ergebnissen der Auswertungen werden Schlussfolgerungen gezogen, welche praktische Bedeutung die gewonnen Erkenntnisse für das Software Engineering haben

Der letzte Schritt der Interpretationsphase ist die Präsentation der Ergebnisse sowie die Veröffentlichung des Experimentaufbaus.

Bei der Darstellung des Experimentaufbaus sollten möglichst alle relevanten Faktoren Berücksichtigung finden. Darüber hinaus sollte man auf alle Probleme hinweisen, die im Laufe des Experiments aufgetreten sind und wie man versucht hat, diese zu lösen. Eine solch detaillierte Darstellung erleichtert das Design von weiteren Experimenten.

Die Veröffentlichung der Ergebnisse umfasst zum einen die bloße Wiedergabe der Rohdaten und zum anderen die Auswertungen und Schlussfolgerungen, die anhand dieser Daten vorgenommen wurden.

Die wesentlichen Kriterien und Arbeitsschritte der Interpretationsphase sind also:

- Daten auf Vollständigkeit/Konsistenz prüfen
- Daten auswerten (Methoden, Statistiken)
- Experimentaufbau, Rohdaten, Ergebnisse veröffentlichen

2.6 Ein Beispiel für ein Experiment im Software Engineering

Am Institut für Software Engineering der Universität Hannover wurde im Wintersemester 2004/05 in Form einer Laborübung ein Experiment (Test-First Experiment) mit Studenten durchgeführt [4].

Dieses Experiment erfüllt im Wesentlichen den in Kapitel 2.5 erläuterten Experimentprozess und wird deshalb als Beispiel für ein Experiment im Software Engineering vorgestellt.

Die Experiment-Frage lautet:

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

Zu Beginn der Planungsphase wurden mehrere Hypothesen aufgestellt von denen letztlich die folgenden vier im Experiment berücksichtigt und ausgewertet wurden.

- Beim Test-First-Vorgehen werden mehr Testfälle erzeugt als beim klassischen Testen.
- Die Testabdeckung bei Test-First ist höher (≥ 90) als beim klassischen Testen (≥ 70).
- Die Entwicklung mit Test-First ist langsamer als mit klassischem Testen
- Die Akzeptanz verläuft in einem klaren Kurvenverlauf

Um den Unterschied zwischen klassischem Testen und der Test-First Methode herauszustellen, wurden die Versuchspersonen in zwei Gruppen eingeteilt. Beide Gruppen mussten die gleiche Aufgabestellung bearbeiten, wobei die Testgruppe streng nach den Prinzipien von Test-First zu entwickeln hatte, während die Kontrollgruppe die Testfälle auf klassische Art und Weise entwickeln musste.

Die zu bearbeitende Aufgabe war die Implementierung einer Bibliothek zur Verwaltung und Analyse einer semantischen Beschreibungssprache in Java. Es wurde sich dafür entschieden neben Test-First noch weitere XP-Techniken einzusetzen.

Die Anforderungen wurden auf Storycards notiert und ein On-Site-Customer konnte jeder Zeit befragt werden. Außerdem wurde in Form von Pairprogramming gearbeitet.

Zur Identifikation der benötigten Daten, die zum Beweisen der Hypothesen benötigt wurden, sowie zum Finden geeigneter Metriken und Auswertungsverfahren wurde die Goal-Question-Metric-Methode verwendet [6].

Es wurde festgelegt, sämtliche Arbeitsschritte der Versuchspersonen mit Hilfe von Logfiles aufzuzeichnen. Darüber hinaus wurden die Zeiten gemessen, die zum Erfüllen der Anforderungen einer Storycard benötigt wurden. Zusätzlich wurden Fragebögen entwickelt, mit denen die Versuchspersonen gezielt nach ihrem Empfinden und ihren Einschätzungen zur jeweiligen Entwicklungstechnik befragt wurden.

Als Versuchspersonen wurden 19 Masterstudenten der Informatik ausgewählt. Man hätte das Experiment auch mit noch mehr Versuchspersonen durchführen können (an der Laborübung hatten noch mehrere Studenten Interesse), entschied sich aber dafür, die möglichen Versuchspersonen mit Hilfe von Fragebögen auf ihre Eignung zur Teilnahme an diesem Experiment zu überprüfen. Die 19 Personen, die nach der Auswertung der Fragebögen übrig geblieben waren, wurden per Randomisierung in eine Kontroll- und eine Testgruppe eingeteilt. Innerhalb dieser Gruppe wurde zur Bildung der Paare für das Pairprogramming erneut randomisiert.

In der Planungsphase wurden einige Störvariablen für das Experiment identifiziert und mit den folgenden Werten belegt:

- Die Entwicklungsumgebung:
  - Eclipse
- Hilfsmittel zur Lösung der Aufgabe:
  - die Computer wurden vom Netz getrennt
  - alle Materialien wurden nur während der Dauer der Laborübung ausgegeben
  - den Versuchspersonen wurde die Kommunikation untersagt
  - es war verboten, außerhalb der Laborübung zu arbeiten

Während der Planungsphase wurden außerdem Schulungsmaterialien erstellt, mit deren Hilfe die Versuchspersonen auf die zu lösende Aufgabe vorbereitet wurden.

Es wurde eine gemeinsame Schulung für alle Teilnehmer in die Entwicklungsumgebung eclipse entwickelt und durchgeführt. Außerdem gab es eine gemeinsame Einführung in das Thema der Aufgabenstellung sowie zu allgemeinen Grundlagen des Testens.

Zusätzlich wurden spezielle Vorbereitung für die Test First Gruppe so wie für die Gruppe, die mit klassischem Testen arbeitete, entwickelt und durchgeführt.

Bei der Durchführung des Experiments kam es zu einer Reihe von kleineren Problemen (z.B. Absturz eines Rechners, ungleiche Pärchenstärke), von denen aber keine die innere Gültigkeit des Experiments im größeren Maße beeinflusst.

Die Auswertung der Daten erfolgte in zwei Phasen. Unmittelbar nach dem Experiment erfolgte eine erste Auswertung der Hypothesen anhand eines Teils der gesammelten Daten. Dieses Vorgehen war notwendig, da man die Ergebnisse des Experimentes noch während des laufenden Semesters mit den Studenten besprechen wollte und eine komplette Auswertung der Daten in diesem Zeitrahmen jedoch nicht durchführbar war.

Die ausführliche Auswertung aller Daten ist bisher noch nicht abgeschlossen es lässt sich aber festhalten, dass die vier aufgestellten Hypothesen bewiesen werden konnten.

Zu dem Test-First-Experiment wurde bisher ein Paper veröffentlicht, welches den Experimentaufbau und die Schwierigkeiten beim Design und bei der Durchführung beschreibt.

2.7 Der Komplexitätsbegriff

Allgemein versteht man unter Komplexität die Eigenschaft eines Systems oder Modells, die die Beschreibungen seines Gesamtverhaltens in einer beliebigen Sprache erschwert.

In der Informatik beschreibt die Komplexität in erster Linie die „Kompliziertheit“ von Daten, Algorithmen, Problemen, und Produkten. Je nach Fachgebiet gibt es ganz unterschiedliche Definitionen des Komplexitätsbegriffs.

In der theoretischen Informatik findet man beispielsweise den Begriff der *Komplexität von Algorithmen*. Er beschreibt den maximalen Ressourcenbedarf eines Algorithmus. Hierbei wird meistens noch zwischen *Zeit-* und *Speicherkomplexität* unterschieden.

Im Software Engineering hingegen existieren die Begriffe der *Produktkomplexität* und *Prozesskomplexität* [10].

Unter *Produktkomplexität* versteht man hierbei die Kompliziertheit eines Stück Softwares hinsichtlich bestimmter Faktoren wie Lesbarkeit, Wartbarkeit oder Zuverlässigkeit.

Die *Prozesskomplexität* dagegen beurteilt den Entwicklungsprozess, z.B. in Bezug auf Zeit- und Kostenaufwand.

2.8 Metriken

Zum Messen von Komplexität dienen Metriken. Der Begriff der Metrik stammt aus der Mathematik, er drückt den Abstand zwischen zwei Dingen (axiomatisch definiert) aus [11].

Für Softwaremetriken gilt nach IEEE Standard 1061 die folgende Definition:

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

Softwaremetriken werden analog zum Komplexitätsbegriff in *Prozess-* und *Produktmetriken* unterteilt.

Prozessmetriken dienen dazu, die Komplexität des Entwicklungsprozesses zu beurteilen. Sie messen z.B. den erwarteten Zeit und Kostenaufwand oder beurteilen die Effizienz bestimmter Techniken und Werkzeuge.

Produktmetriken werden weiter in *Größen-* und *Strukturmetriken* unterteilt. *Größenmetriken* bieten Maße zur Bestimmung des Umfangs von Quellcode. Bekannte Größenmetriken sind die Lines-of-Code-Metrik und die Metriken von Halstead [10].

Strukturmetriken analysieren, wie der Name schon sagt, die Struktur des Codes. Die wohl bekannteste Metrik aus diesem Bereich ist die McCabe-Metrik aber auch sämtliche objektorientierte Metriken gehören zu den *Strukturmetriken*.

Metriken werden anhand folgender Gütekriterien beurteilt:

- **Objektivität** (keine subjektiven Einflüsse des Messenden auf die Messung möglich)
- **Zuverlässigkeit** (eine Wiederholung unter den selben Messbedingungen liefert die selben Ergebnisse)
- **Nützlichkeit** (praktische Bedürfnisse, die erfüllt werden)
- **Normierung** (existiert eine Vergleichbarkeitsskala, auf welcher die Messergebnisse eindeutig abgebildet werden)
- **Ökonomie** (Kosten für die Durchführung der Messung)
- **Validität** (Messergebnisse erlauben einen eindeutigen und unmittelbaren Rückschluss auf die Ausprägung der Kenngrößen)

2.9 Skalen

Die Aussagekraft einer Metrik hängt von der Skala ab, auf der sie dargestellt wird.

Eine Skala ist eine regelmäßig eingeteilte Anzeigefläche, die dazu dient, einen Wert anzuzeigen.

Man unterscheidet verschiedene Typen von Skalen.

Eine *Nominalskala* besteht aus einer Menge von Werten, die nur durch einen eindeutigen Namen gekennzeichnet sind. Ein Messwert kann dabei lediglich auf Gleichheit bzw. Ungleichheit mit den Werten der Skala geprüft und diesen zugeordnet werden. Eine Vergleichsmöglichkeit zwischen den einzelnen Werten besteht nicht.

Eine *Ordinalskala* definiert zusätzlich eine Ordnung der Werte. Messwerte können so mit Hilfe von „ist größer/kleiner als“ Relationen miteinander verglichen werden.

Eine *Intervallskala* erweitert die Aussagekraft einer *Ordinalskala* durch die zusätzliche Möglichkeit der Abstandsbestimmung zwischen zwei Messwerten. Hierzu wird willkürlich ein Nullpunkt festgelegt.

Eine *Absolutskala* besitzt das höchste Skalenniveau. Sie ist eine Skala mit echtem Nullpunkt, auf der sich sowohl absolute als auch relative Häufigkeiten darstellen lassen.

Das Ziel beim Festlegen von Skalen zur Darstellung der gemessenen Werte sollte deshalb nach Möglichkeit die Entwicklung einer Absolutskala sein.

2.10 Die Goal-Question-Metric-Methode

Die grundlegenden Fragen, die bei der Entwicklung von Metriken Beachtung finden müssen, sind im Wesentlichen:

Was soll gemessen werden (Eigenschaften, Merkmale von Produkten, Methoden)?

Wie misst man diese Eigenschaften?

Wie werden die Messwerte dargestellt (Skala)?

Ein häufig gebrauchter Ansatz zur Definition der Messziele ist die Verwendung besonders gut messbarer Eigenschaften. Erst im zweiten Schritt wird überlegt, welche Interpretationsmöglichkeiten die gemessen Werte liefern. Dieser Ansatz führt oft zu einer relativ geringen Aussagekraft der letztlich definierten Metriken, er birgt sogar die Gefahr, Eigenschaften zu messen, die keinerlei Relevanz haben.

Ein anderes, zielorientiertes Vorgehen zur Identifikation von Metriken bietet die Goal-Question-Metric-Methode [6]. Sie stützt sich auf die Devise, nicht dass zu messen, was leicht zu messen ist, sondern die Messziele aus den Projektzielen heraus abzuleiten.

Der Ablauf dieser Methode ist im Wesentlichen durch drei Schritte definiert:

1. Ziele definieren

Zunächst wird festgelegt, was man mit Hilfe der zu entwickelnden Metriken überhaupt messen möchte. Dabei können ausgehend von einem übergeordnetem Ziel Teilziele formuliert werden, welche alle zur Erfüllung des Hauptziels erforderlich sind.

Um die Definition dieser Ziele zu erleichtern wurde ein Template entwickelt, das so genannte Goal-Question-Metric-Goal-Template, welches in Kapitel 2.5.1 bereits vorgestellt wurde.

2. Fragen formulieren

Aus jedem Teilziel werden Fragen abgeleitet, deren Beantwortung notwendig ist, um das Erreichen der Zielsetzung zu überprüfen.

3. Metriken definieren

Für jede Frage wird geprüft, welche Daten zum Beantworten erforderlich sind und wie die Erhebung der Daten erfolgen kann.

3 Anforderungsanalyse

3.1 Vorgehen bei der Anforderungserhebung und Dokumentation

In den folgenden Kapiteln werden die Anforderungen an das zu entwickelnden Werkzeug vorgestellt.

Die Idee für dieses Werkzeug entstand im Wintersemester 2004/05 während der Durchführung des Test-First Experiments. Die Anforderungen kommen also in erster Linie von den Mitarbeitern des Software Engineering Instituts der Universität Hannover, die an diesem Experiment als Experimentatoren beteiligt waren oder bei der Planung und Auswertung mitgewirkt haben (auch nur Beobachten der Vorgänge).

In Gesprächen sowie anhand der während des Experiments entstandenen Dokumente wurde in den ersten Wochen dieser Arbeit versucht, die grundlegenden Funktionalitätsanforderungen an das Werkzeug herauszuarbeiten. Im Laufe der Entwicklungsphase wurden diese konkretisiert und mit Hilfe der erarbeiteten Kenntnisse über den Ablauf von Experimenten um eigene Ideen ergänzt.

In Kapitel 3.2 werden technische, funktionale und Qualitäts-Anforderungen sowie Anforderungen an die Oberfläche in kurzen Texten beschrieben. Dabei werden die in [12] empfohlenen Kriterien zur Anforderungsdokumentation berücksichtigt

Grundlegende funktionale Anforderungen werden mit Hilfe von Use Cases noch detaillierter erläutert (Kapitel 3.2.4).

3.2 Anforderungen

3.2.1 Allgemeine, technische Anforderungen

Es soll eine Java Applikation erstellt werden. Mit Hilfe des Programms können Benutzer Experimente planen, verwalten und analysieren.

Das Programm soll durch PlugIns erweiterbar sein. Hierfür ist eine einheitliche PlugIn-Schnittstelle zu erstellen. Ein PlugIn, welches das Hauptprogramm um Metriken zur Komplexitätsmessung erweitert wird innerhalb dieser Arbeit implementiert.

Die Applikation verfügt über eine graphische Oberfläche, auf der sämtliche Benutzeraktionen stattfinden (Anforderungen an die Oberfläche: Kapitel 3.2.6). Das Hauptelement der Oberfläche bildet ein gerichteter Graph, der Experimentgraph.

Die Knoten dieses Graphen stellen Arbeitsschritte in einem Experiment dar (s. Kapitel 2.5).

Einen Anhaltspunkt dafür, in welchem Grad diese Arbeitsschritte unterteilt werden sollen, bietet der folgende Auszug von Schritten, der bei der Analyse des Test-First Experiments entstand.

- Experimentfrage spezifizieren
- Hypothesen aufstellen
- Versuchspersonenzahl kleiner 10, zwischen 10 und 30, größer als 30 (je ein Knoten)
- Randomisierung
- Blockung
- Schulung mit allen Versuchspersonen
- Spezielle Schulungen
- Datenerhebungsmethoden festlegen: Fragebögen, Logfiles, Quellcode,… (jeweils ein Knoten)
- Analyse

Mit Hilfe des Experimentgraphen sowie der spezifizierten Knoteninhalte ist es möglich, das Test-First Experiment abzubilden.

Der Graph verfügt über einen Start- und einen Endknoten, welche Experimentbeginn und –ende darstellen.

Ein bestimmter Weg durch den Graphen vom Start- zum Endknoten bildet ein spezielles Experiment ab. Zur Veranschaulichung dient die folgende Skizze:

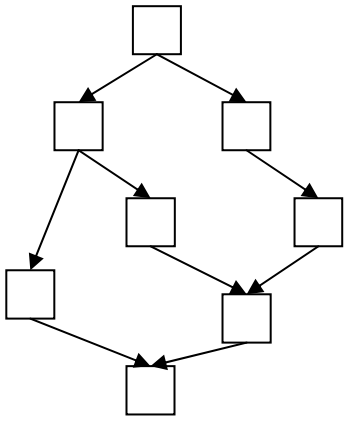

Abb. 3: Aufbau des Experimentgraphen

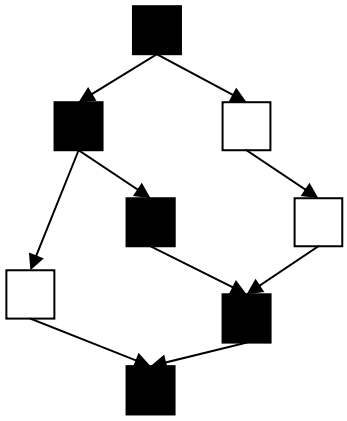

Abb. 4: ein Experiment

Die zunächst gegebene Knotenmenge soll erweiterbar und veränderbar sein.

Der Graph darf keine Zyklen enthalten, man kommt also auf jedem Weg in endlich vielen Schritten vom Start- zum Endknoten.

In dem Graphen wird zur gleichen Zeit immer nur ein Experiment dargestellt.

Jeder Knoten besitzt eine Liste von Templates. Templates sind Dokumentvorlagen, die im Laufe des Experiments ausgefühlt werden müssen. Hierzu zählen z.B. Teilnehmerlisten oder Auswertungsdokumente. Diese Templates müssen benannt und nach Möglichkeit spezifiziert werden. Dabei dienen die Dokumente aus dem Test-First Experiment als Vorlage. Wie bei der Knotenmenge ist auch die Menge der Templates veränderbar. Alle Templates werden in einem Template-Ordner gespeichert.

Jeder Knoten erhält neben der Liste von Templates einen kurzen Beschreibungstext, welcher den Benutzern Informationen über Arbeitsschritte und Auswahlkriterien liefert. Dieser wird den Nutzern in Form eines Tooltipps angezeigt.

3.2.2 Funktionale Anforderungen an das Hauptprogramm

Benutzer können Experimente anlegen (Use Case Nr.1). Für jedes Experiment wird ein neuer Ordner angelegt.

Experimente besitzen je eine Liste für Knoten und Dokumente. Diese sind nach dem Anlegen eines neuen Experiments leer. Ein Benutzer hat die Möglichkeit, sie zu füllen.

Dazu wählt er Knoten im Graphen aus (er markiert diese) (Use Case Nr.2). Es können, ausgehend vom Startknoten, nur solche Knoten gewählt werden, die mindestens einen markierten Vorgängerknoten haben. Des Weiteren dürfen keine Knoten auf gleicher Ebene markierbar sein, da der markierte Weg genau einem Experimentaufbau entspricht.

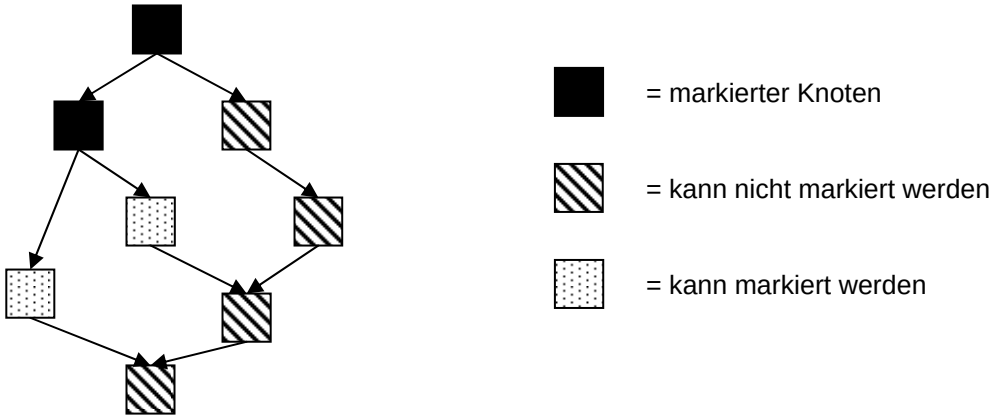

Abb. 5: Markierung der Knoten

Ein Benutzer kann markierte Knoten durch eine erneute Auswahl aus dem Experiment löschen (Use Case Nr.3). Dabei können nur Knoten entfernt werden, die keine Lücke in dem markierten Weg hinterlassen.

Benutzer können Knoten im Graphen aktivieren. Dieses Aktivieren ist unabhängig von dem Markieren der Knoten. Von dem gerade aktiven Knoten werden Detailinformationen angezeigt.

Zu den Detailinformationen zählen:

- Templates
- Dokumente
- Notizen (optional)

Ein Benutzer kann Templates des aktiven Knotens herunterladen, diese also an einer beliebigen Stelle im Dateisystem ablegen(Use Case Nr.4).

Ist ein aktiver Knoten gleichzeitig markiert, so hat der Benutzer die Möglichkeit, Dokumente hochzuladen (Use Case Nr.5).

Ein Benutzer löscht Dokumente, indem er eine entsprechende Löschfunktion für ein Dokument auswählt. Gelöschte Dokumente werden sowohl aus dem Experimentordner als auch aus der Liste der Dokumente entfernt.

Ein Löschen von Templates ist über die Benutzeroberfläche nicht möglich.

In ein Textfeld kann ein Benutzer beliebige Informationen eingeben. Wählt er die Funktion „Notizen übernehmen“ aus, so werden diese Informationen dem aktiven Knoten hinzugefügt.

Benutzer können Experimente speichern und laden.

Zum Speichern eines Experiments muss die Knotenliste dieses Experiments mindestens einen Eintrag enthalten. Gespeichert werden alle markierten Knoten mit zugehörigen Dokumenten und Notizen.

Will ein Benutzer ein Experiment laden, so kann er aus einer Liste aller gespeicherten Experimente eins auswählen und dieses öffnen (Use Case Nr.6).

Fährt der Benutzer mit der Maus über einen beliebigen Knoten, so sollen ihm die oben spezifizierten Knoteninformationen angezeigt werden.

Das Ändern des Experimentgraphen erfolgt nicht über die Benutzeroberfläche. Hierzu müssen andere Änderungsmöglichkeiten vorgesehen werden.

Über einen Menüauswahlpunkt haben Benutzter die Möglichkeit PlugIns zu laden (Use Case Nr.7) bzw. wieder zu löschen (Use Case Nr.8).

3.2.3 Technische/funktionale Anforderungen an das Metrik-PlugIn

Das zu entwickelnde PlugIn soll das Programm um Funktionalitäten erweitern, welche die Komplexität von Experimenten abschätzen.

Eine Definition dieser Komplexität sowie von Metriken zur ihrer Berechnung wird in Kapitel 4 vorgestellt. An dieser Stelle werden lediglich die funktionalen Anforderungen an das PlugIn beschrieben.

Das PlugIn soll sämtliche in Kapitel 4 spezifizierte Metriken berechnen. Die Ergebnisse der Berechnung werden auf einer graphischen Oberfläche dargestellt (Anforderungen an die PlugIn-Oberfläche: Kapitel 4.2.7)

Der Benutzer hat die Möglichkeit, sich die Werte für jedes Templates, für jeden Knoten und für das gesamte Experiment anzeigen zu lassen.

Wird im Hauptprogramm ein Knoten markiert oder gelöscht, ein Dokument hochgeladen/gelöscht oder ein neues Experiment geöffnet/angelegt so werden die Metriken, die von den jeweiligen Werten abhängen neu berechnet (Use Case Nr.9).

3.2.4 Use Cases

Die grundlegenden Funktionalitäten des Programms werden in Use Cases genauer spezifiziert.

Use Case Nr.1	Experiment anlegen
Stakeholder und Interesse	Benutzer möchte ein neues Experiment planen
Voraussetzung	Programm wurde erfolgreich gestartet
Erfolgsfall	ein neuer Experiment-Ordner wird angelegt im Graphen wird das neue Experiment dargestellt(hat anfangs keine Einträge)
Auslöser	Benutzer wählt „neues Experiment anlegen“
Beschreibung	1. System fragt nach einem Experimentnamen 2. Benutzter gibt Namen ein 3. System legt neuen Ordner mit diesem Namen im Experiment-Ordner an 4. System löscht alle markierten Einträge im Experimentgraphen 5. System zeigt angelegtes Experiment als aktuelles Experiment an
Erweiterungen	2a Wenn bereits ein Experiment mit dem eingegebenen Namen existiert fragt das System nach einem anderen 2b wenn Benutzer die Eingabe abbricht verändert das System sich nicht 3a wenn Experiment-Ordner nicht vorhanden, meldet das System dies, es wird kein neues Experiment angelegt

Use Case Nr.2	Knoten zum Experiment hinzufügen
Stakeholder und Interesse	Benutzer möchte einen Knoten zum aktuellen Experiment hinzufügen
Voraussetzung	Es ist ein Experiment angelegt oder geladen
Erfolgsfall	Knoten wurde Experiment hinzugefügt und wird im Graphen als markiert gekennzeichnet
Auslöser	Benutzer doppelklickt auf unmarkierten Knoten
Beschreibung	1. System prüft, ob ein Experiment geöffnet ist 2. System prüft, ob Knoten markiert werden kann 3. System fügt Knoten in die Knotenliste des Experiments ein 4. System markiert entsprechenden Knoten im Graphen
Erweiterungen	2a wenn Knoten nicht markiert werden kann, zeigt das System eine entsprechende Meldung, die Schritte 3 und 4 entfallen

Use Case Nr.3	Knoten aus Experiment löschen
Stakeholder und Interesse	Benutzer möchte einen Knoten aus dem aktuellen Experiment entfernen
Voraussetzung	Es existiert ein Experiment dessen Knotenliste mindestens einen Eintrag enthält
Erfolgsfall	Knoten wurde aus Experiment entfernt im Graphen wird die Markierung gelöscht
Auslöser	Benutzer doppelklickt auf markierten Knoten
Beschreibung	1. System prüft, ob Knotenmarkierung gelöscht werden kann 2. System löscht Knoten aus der Knotenliste des Experiments 3. System löscht Notizen und Dokumente von diesem Knoten 4. System löscht Markierung des entsprechenden Knoten im Graphen
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	Template download
Stakeholder und Interesse	Benutzer möchte ein Template im einem Ordner seiner Wahl speichern
Voraussetzung	In der Templateliste ist ein Eintrag vorhanden und selektiert
Erfolgsfall	Eine Kopie des Templates wird im Dateisystem abgelegt
Auslöser	Benutzer wählt „download Template“
Beschreibung	1. System überprüft, ob das gewählte Template im Template-Ordner liegt 2. System zeigt Dateiauswahldialog 3. Benutzer wählt einen Ordner aus und bestätigt 4. System kopiert Template und speichert die Kopie im ausgewählten Ordner 5. System meldet erfolgreichen Download
Erweiterungen	1a wenn Template nicht im Template-Ordner vorhanden ist, zeigt System entsprechende Meldung, die restlichen Schritte entfallen 3a wenn der Benutzer die Auswahl abbricht, entfallen die folgenden Schritte 4a wenn beim Download ein Fehler auftritt, zeigt das System diesen an, Schritt 5 entfällt

Use Case Nr.5	Dokument upload
Stakeholder und Interesse	Benutzer möchte ein eigenes Dokument uploaden
Voraussetzung	es existiert ein aktuelles Experiment mit mindestens einem markierten Knoten der gerade aktivierte Knoten ist markiert
Erfolgsfall	eine Kopie des Dokuments wird im Experimentordner abgelegt das Dokument wird einem Knoten und einem Template zugeordnet
Auslöser	Benutzer wählt „upload Dokument“
Beschreibung	1. System fragt nach Dateinamen 2. Benutzer wählt ein Dokument aus und bestätigt 3. System verlangt Zuordnung des Dokuments zu einem Template 4. Benutzer nimmt Zuordnung vor 5. System fragt nach der benötigten Zeit zum Ausfüllen des Dokuments 6. Benutzer gibt Zeit ein 7. System kopiert Dokument und speichert die Kopie im Experimentordner 8. System speichert Dokumentname mit Template-Zuordnung und entsprechender Zeit an dem aktiven Knoten
Erweiterungen	2a, 3a und 5a wenn der Benutzer die Auswahl abbricht, entfallen die folgenden Schritte 4a Benutzer hat auch die Möglichkeit das Dokument explizit keinem Template zuzuordnen, weiter mit 5

Use Case Nr.6	Experiment öffnen
Stakeholder und Interesse	Benutzer möchte ein vorhandenes Experiment ansehen Benutzer möchte ein Experiment bearbeiten
Voraussetzung	Im Experiment-Ordner befindet sich mindestens ein Experiment
Erfolgsfall	das gewählte Experiment wird mit allen Einträgen dargestellt (Knotenmarkierungen, Dokumente, Notizen)
Auslöser	Benutzer wählt „Experiment öffnen“
Beschreibung	1. System zeigt Auswahldialog mit allen Experimenten, die im Experiment-Ordner gespeichert sind 2. Benutzer wählt ein Experiment aus und bestätigt 3. System liest alle Einträge dieses Experiments aus 4. System löscht alle Einträge des alten Experiments 5. System markiert Experimentknoten 6. zu jedem markierten Knoten werden die Dokument- und Zeitinformationen sowie eventuell vorhandene Notizen übernommen
Erweiterungen	1a wenn der Benutzer die Auswahl abbricht, entfallen die folgenden Schritte 3a tritt ein Fehler auf, so entfallen die folgenden Schritte, es wird eine Fehlermeldung angezeigt

Use Case Nr.7	PlugIn laden
Stakeholder und Interesse	Benutzer möchte das Programm um zusätzlich Funktionalitäten erweitern
Voraussetzung	Im Plugin-Ordner ist mindestens ein PlugIn vorhanden
Erfolgsfall	Die Funktionalität des PlugIns kann im vollen Umfang genutzt werden Auf einer im Hauptprogramm dafür vorgesehenen Fläche wird die graphische Oberfläche des PlugIns angezeigt
Auslöser	Benutzer wählt „PlugIn laden“
Beschreibung	1. System zeigt Liste aller verfügbaren PlugIns 2. Benutzer wählt ein PlugIn aus und bestätigt Eingabe 3. System lädt PlugIn 4. System initialisiert PlugIn mit dem momentan geladenen Experiment 5. System stellt Oberfläche des PlugIns im dafür vorgesehenen Bereich dar
Erweiterungen	2a wenn Benutzer Eingabe abbricht entfallen die folgenden Schritte 3a wenn Probleme auftreten und das PlugIn nicht korrekt geladen werden kann, bricht das System den Vorgang ab und informiert den Benutzer 4a ist kein Experiment geladen, wird Plugin mit einem Dummyexperiment (enthält keine Einträge) initialisiert

Use Case Nr.8	PlugIn deaktivieren
Stakeholder und Interesse	Benutzer benötigt PlugIn-Funktionalitäten nicht mehr
Voraussetzung	Es ist mindestens ein PlugIn im Hauptprogramm geladen
Erfolgsfall	Sämtliche PlugIn-Funktionalitäten sowie die graphische Oberfläche werden im Hauptprogramm gelöscht
Auslöser	Benutzer wählt „PlugIn deaktivieren“
Beschreibung	1. System zeigt Liste aller geladenen PlugIns an 2. Benutzer wählt einen Eintrag aus 3. System deaktiviert Funktionalitäten des PlugIns im Hauptprogramm 4. System gibt die durch das PlugIn belegte Oberfläche wieder frei und löscht die Anzeige
Erweiterungen	1a wenn Benutzer Eingabe abbricht entfallen die folgenden Schritte

Use Case Nr.9	Metrik berechnen
Stakeholder und Interesse	Benutzer möchte Informationen zur Komplexität seines Experiments erhalten
Voraussetzung	das Metrik-Plugin ist geladen ein Experiment ist angelegt/geöffnet
Erfolgsfall	Alle Metriken werden korrekt berechnet und die entsprechenden Werte dem Benutzer angezeigt
Auslöser	Benutzer wählt auf der PlugIn-Oberfläche eine Metrik sowie einen dazugehörigen Eintrag(Knoten, Template) für den die Metrik berechnet werden soll aus Im Hauptprogramm wird das geladene Experiment verändert(Knoten zugefügt/entfernt, Dokument hochgeladen/gelöscht oder anderes Experiment geladen)
Beschreibung	1. Plugin berechnet ausgewählte Metrik des Benutzer bzw. aktualisiert selektierte Metrik 2. die geänderten Werte werden auf der Oberfläche dargestellt
Erweiterungen	

3.2.5 Qualitätsanforderungen

Durch die Wahl von Java als Programmiersprache ist die Portierbarkeit des Programms gegeben.

Darüber hinaus liegt der Fokus der Qualitätsanforderungen auf der flexiblen und leichten Erweiterbarkeit des Experimentgraphen.

Es soll auf „einfache“ Art und Weise möglich sein, die Knoteninhalte zu erweitern oder zu modifizieren und Knoten aus dem Graphen zu entfernen oder einzufügen. Dabei kann davon ausgegangen werden, dass Personen, die dieses Werkzeug benutzten, über Erfahrung im Bereich des Software Engineering verfügen.

Des Weiteren soll das Hinzufügen und Entfernen von neuen Templates auf schnelle und einfache Art und Weise möglich sein.

Für das PlugIn ist die Möglichkeit der Erweiterung um neue Metriken, ohne im großen Maße in den vorhandenen Quelltext eingreifen zu müssen, wünschenswert.

Zum Anpassen des geschätzten Zeitaufwands muss ein leichter und schneller Zugriff auf die vorhandenen Schätzwerte gegeben sein.

3.2.6 Anforderungen an die Benutzeroberfläche des Hauptprogramms

Sämtliche Benutzereingaben finden auf einer graphischen Oberfläche statt.

Es gibt einen Experimentgraphen. Dieser bildet die Hauptkomponente auf der Oberfläche und soll entsprechenden Platz zu gewiesen bekommen.

Daneben gibt es zwei weitere, inhaltlich getrennte Bereiche. In dem einen werden Informationen zu dem gerade aktiven Knoten dargestellt. Hier soll je eine Liste der Templates und Dokumente angezeigt werden. Es muss eine Möglichkeit gegeben sein, über entsprechende Buttons, Dokumente hochzuladen oder zu löschen, und Templates im Dateisystem zu speichern. Ein Eingabefeld, über das Notizen zu den Knoten hinzugefügt werden können soll ebenfalls vorhanden sein.

Der dritte Bereich ist für die PlugIns reserviert. Wünschenswert ist eine dynamische Anpassung der Gestaltung dieses Bereichs in Abhängigkeit der Anzahl der geladenen PlugIns. Zunächst kann aber auch eine Zweiteilung dieses Bereiches entsprechend der zwei PlugIns, die zu programmieren sind, erfolgen.

Bei der Gestaltung der Oberfläche soll in erster Linie auf die Usability (Definition laut ISO 9241) geachtet werden.

3.2.7 Anforderungen an die Benutzeroberfläche des PlugIns

Für das PlugIn steht nur ein kleiner Bereich zur Verfügung. In diesem müssen alle Metriken auf übersichtliche Art und Weise dargestellt werden können.

Über entsprechende Listen hat der Benutzer die Möglichkeit, aus dem zum Experiment gehörenden Knoten und Templates einen Eintrag zu wählen und sich die Werte der Berechnungen für diesen Eintrag anzeigen zu lassen.

4 Entwicklung der Metriken für das PlugIn

4.1 Komplexität von Experimenten

In Kapitel 2.7 werden unterschiedliche Definitionen des Komplexitätsbegriffes vorgestellt. Mit der Komplexität eines Systems verbindet man im intuitiven Sprachgebrauch eine gewisse Schwierigkeit, alle Zusammenhänge dieses Systems zu verstehen. Im Software Engineering wird zwischen Produkt- und Prozesskomplexität unterschieden.

Was aber versteht man unter der Komplexität von Experimenten?

Um Metriken zu entwickeln, die diese Komplexität messen, reicht ein vages und intuitives Verständnis dieses Begriffes nicht aus. Anhand der Definitionen aus dem Gebiet des Software Engineering wird deshalb eine Definition für die Komplexität von Experimenten entwickelt.

Ein großer Teil der Prozessmetriken im Software Engineering bezieht sich auf die Frage nach dem geschätzten Entwicklungsaufwand [13]. Diese Aufwandsschätzung ist mittlerweile in (fast) allen größeren Projekten in den Entwicklungsprozess integriert. Mit ihrer Hilfe ist es möglich, bereits in einer frühen Phase des Projektes einen Überblick über den zu erwartenden Aufwand zu erhalten. Auf diese Weise können Risiken rechtzeitig erkannt werden.

Eine Aufwandschätzung kann in der Planungsphase von Experimenten eine große Hilfe zum Einschätzen des Experimentumfanges liefern. Die Frage die sich stellt, lautete:

Wie kann man den Aufwand eines Experimentes abschätzen?

Im Software Engineering gibt es im Wesentlichen zwei gebräuchliche Metriken zur Aufwandsschätzung.

Das so genannte Function-Point-Verfahren [12], welches 1979 bei IBM entwickelt wurde, orientiert sich an den Anforderungen der zu erstellenden Software. Diese werden in bestimmte Kategorien eingeteilt und nach ihrem Schwierigkeitsgrad klassifiziert. Für jede Kategorie und Klassifizierzng gibt es Gewichtungsfaktoren. Aus allen diesen Angaben sowie Einflussfaktoren, die eine mögliche Änderung dieser Werte auslösen können, wird schließlich ein Function-Point ermittelt, der den Aufwand des Projektes angibt. Anhand von Graphen und Tabellen, in denen die Function-Points von ähnlichen Projekten dem tatsächlich gemessenen Arbeitsaufwand in diesen Projekten gegenübergestellt sind, wird letztlich der zu erwartende Aufwand in Mitarbeitermonaten (MM) abgeschätzt.

Jeder einzelne Schritt in diesem Verfahren erfolgt nach fest definierten Kriterien, welche in dem ca. 100seitigen Function-Point-Counting-Practices-Manual der IFPUG [14] festgehalten sind.

Ein Übertragen dieser Methode auf den Bereich von Experimenten ist nicht ohne weiteres möglich.

Das Erstellen der Kategorien und Klassifizierungsvorschriften erfordert ein großes Maß an Erfahrung mit Experimenten. Man muss sich fragen, ob und anhand welcher Komponenten eines Experimentes eine solche Kategorisierung überhaupt durchführbar ist. Außerdem werden Referenzexperimente benötigt, mit denen entsprechende Function-Point-Tabellen erstellt werden können.

Die Entwicklung einer der Function-Point-Methode entsprechender Metrik für Experimente würde bei Beachtung aller Kriterien den Zeitrahmen dieser Arbeit sprengen. Dieser Ansatz wird deshalb nicht weiter verfolgt.

Ein anderes Verfahren, das bei der Aufwandschätzung im Software Engineering zum Einsatz kommt, ist das COCOMO-Modell nach B. Boehm [12]. Ausgangswert dieser Schätzung sind die geschätzten zu implementierenden Codezeilen, der zu erwartende Aufwand wird in Personenmonaten angegeben.

Dieses Verfahren orientiert sich also an einer gemeinsamen Eigenschaft von Softwareprodukten, nämlich der Länge ihres Codes. Eine Anwendung dieser Methode bei Experimenten erübrigt sich, da sie speziell auf Software zugeschnitten ist.

Sowohl das Function-Point-Verfahren als auch das COCOMO-Modell schätzen den Aufwand der Softwareentwicklung anhand von zwei Kriterien ab, die allen Entwicklungsprozessen gemeinsamen sind, deren Ausprägungen aber stark variieren können.

Gibt es Komponenten, die in allen Experimenten vorkommen?

Auch in Experimenten lassen sich Komponenten identifizieren, die in allen Experimenten vertreten sind. Dies sind z.B. Hypothesen, Versuchspersonen, Templates und Dokumente.

Da Hypothesen im Allgemeinen noch nicht genügend Aussagekraft über den Aufbau und Ablauf eines Experiments bieten, liefern sie kein hinreichendes Kriterium zur Aufwandsschätzung.

Versuchspersonen beeinflussen zwar den Aufwand eines Experimentes, sie lassen sich aber sehr schwer Kategorisieren oder Klassifizieren, eine objektive Beurteilung ist kaum möglich.

Aus diesen Gründen dienen im Folgenden die Dokumente und Templates eines Experiments als Grundlage zur Aufwandsschätzung.

Der Aufwand von Experimenten definiert sich über Templates und Dokumente.

In Kapitel 4.2 werden mit Hilfe der GQM-Methode Metriken entwickelt, mit deren Hilfe sich der so definierte Aufwand von Experimenten abschätzen lässt.

Einen weiteren Schwerpunkt der Komplexitätsabschätzung im Software Engineering bilden die Produktmetriken. Sie beurteilen erstellte Software anhand des Quellcodes z.B. im Hinblick auf Lesbarkeit oder Wartbarkeit.

Produktmetriken im Software Engineering messen die Ausprägung bestimmter gemeinsamer Komponenten von Quellcode. Bekannte Metriken auf diesem Gebiet sind z.B. die Lines-of-Code Methode, die Halsteadmetriken, die Metriken zur Bestimmung der cyclomatischen Komplexität nach McCabe und viele objektorientierte Metriken.

Auch die Beurteilung von durchgeführten Experimenten ist sinnvoll, um z.B. Experimente miteinander zu vergleichen. Die Frage, die es bei der Entwicklung von Produktmetriken für Experimente zu beantworten gilt, lautet:

Welche Komponenten charakterisieren Experimente?

Die Entwicklung von Metriken zur Bestimmung der Produktkomplexität von Experimenten wird in Kapitel 4.3 behandelt.

4.2 Metriken zur Aufwandsschätzung von Experimenten

Mit Hilfe des GQM-Verfahren wurden acht Metriken entwickelt, welche auf der Grundlage von Templates und Dokumenten eine Hilfestellung beim Abschätzen des Aufwands von Experimenten bieten.

Diese Metriken bauen aufeinander auf, sie unterscheiden sich daher in erster Linie in der Anzahl der Dokumente und Templates die sie für ihre Messungen berücksichtigen.

Die Metriken berechnen den zu erwartenden bzw. tatsächlichen Aufwand in Personenstunden (PersStd).

Ziel:	**Aufwandsschätzung für ein Template**
Fragen:	Wie groß ist die voraussichtlich benötigte Zeit zum Ausfüllen eines Templates? Wie häufig muss dieses Template ausgefüllt werden? Wie groß ist die tatsächlich zum Ausfüllen benötigte Zeit? Wie ist das Verhältnis von benötigter zu geschätzter Zeit?
Metriken:	**M1**: **geschätzter Aufwand pro Template** Berechnet die Zeit, die zum Ausfüllen aller Dokumente eines Templates voraussichtlich benötigt wird. $A_{geTemp}(temp) = dok_{be}(temp) * t_{ge}(temp)$ A_{geTemp}: geschätzter Aufwand dok_{be}: Anzahl benötigter Dokumente t_{ge}: geschätzte Zeit für das Ausfüllen eines Dokuments

	M2: durchschnittliche Ausfüllzeit eines Dokumentes Berechnet die durchschnittliche Dauer, die für das Ausfüllen eines Dokumentes benötigt wurde. $$A_{average} = \frac{\sum_{dok \in \{Dok\}}^{\#Dok} dok * t_{be}(dok)}{\#Dok}$$ $A_{average}$: durchschnittliche Ausfüllzeit t_{be}: benötigte Zeit zum Ausfüllen eines Dokument *#Dok:* Anzahl Dokumente zu diesem Template **M3: noch verfügbare Zeit für fehlende Dokumente** Berechnet anhand des geschätzten Aufwands für alle benötigten Dokumente und den tatsächlich benötigten Zeiten für bereits vorhandene Dokumente die noch verfügbare Zeit für fehlende Dokumente. Angegeben wird die Abweichung dieser noch zur Verfügung stehenden Zeit zum Ausfüllen eines Dokumentes von der geschätzten Zeit. Ein positiver Wert bedeutet, dass zusätzliche Zeit vorhanden ist, ein negativer Wert gibt an, dass das Ausfüllen der vorhandenen Dokumente bereits mehr Zeit als erwartet in Anspruch genommen hat. $$t_{abw}(dok) = \frac{A_{ge}(temp) - \sum_{dok \in \{Dok\}}^{\#Dok} t_{be}(dok)}{dok_{be} - dok_{vor}} - t_{ge}(temp)$$ t_{abw}*:* Abweichung von der geschätzten Zeit dok_{vor}*:* vorhandene Dokumente

Hilfsmittel:

Die Anzahl benötigter Dokumente zu einem Template wird vom Autor des Templates vorgegeben, sie kann allerdings von den zwei Faktoren ***Versuchspersonenzahl*** und ***Häufigkeit der Datenerhebung*** abhängen.

Folgende drei Fragen muss der Autor eines Templates beantworten:

- Wie häufig wird das Dokument unabhängig von anderen Faktoren benötigt? (***benAnz***)
- Hängt die benötigte Anzahl von der Versuchspersonenzahl ab? Wenn ja - wie oft pro Versuchsperson wird es benötigt? (***benAnzProVers***)

- Hängt die benötigte Anzahl von der zu erhebenden/erhobenen Datenmenge ab? Wenn ja - wie oft pro Datenerhebung wird es benötigt? (***benAnzProDat***)

Es werden also drei Werte zu jedem Template angegeben.

Darüber hinaus müssen die Anzahl der Versuchspersonen im Experiment sowie die Häufigkeit der Datenererbung bekannt sein.

Die benötigte Anzahl Dokumente zu einem Template berechnet sich auf der Grundlage der oben spezifizierten Attribute und Werte dann wie folgt:

$$dok_{be}(temp) = benAnz + benAnz\,\mathrm{Pr}\,oVers * V + benAnz\,\mathrm{Pr}\,oDat * Dat$$

V: Anzahl Versuchspersonen

Dat: Anzahl der Datenerhebung

Jedes Template muss eine Angabe über den geschätzten Zeitaufwand zum Ausfüllen eines entsprechenden Dokuments enthalten.

Beim Speichern von Dokumenten muss eine Zuordnung der Dokumente zu einem Template erfolgen. Die tatsächlich benötigte Zeit zur Bearbeitung des Dokumentes muss angegeben und gespeichert werden.

Ziel:	**Aufwandsschätzung für einen Knoten**
Fragen:	Wie viele Templates besitzt der Knoten? In welchem Maße trägt der Aufwand eines einzelnen Templates zum Gesamtaufwand des Knotens bei? Wie viele Dokumente wurden noch nicht ausgefüllt? Wie viel Zeit steht zum Ausfüllen dieser Dokumente noch zur Verfügung? Wie viel Zeit wird voraussichtlich für das Ausfüllen der fehlenden Dokumente noch benötigt?
Metriken:	**M4: geschätzter Aufwand pro Knoten** Berechnet für einen Knoten den geschätzten Aufwand. Dafür wird die Summe über den Aufwand aller Templates im Knoten gebildet. $A_{geKnoten}(k) = \sum_{temp \in \{Temp\}}^{\#Temp} A_{geTemp}(temp)$ $A_{geKnoten}$: geschätzter Aufwand pro Knoten

	M5: verbleibende Zeit für fehlende Dokumente im Knoten Berechnet die noch verbleibende Zeit im Knoten unter Berücksichtigung aller vorhandenen Dokumente in diesem Knoten. $t_{frei} = A_{ge}(k) - A_{be}(k)$ $A_{beKnoten}(k) = \sum_{temp \in \{Temp\}}^{\#Temp} \sum_{dok \in \{Dok\}}^{\#Dok} t_{be}(dok)$ t_{frei}: noch übrige Zeit zum Ausfüllen von fehlenden Dokumenten $A_{beKnoten}$: benötigter Aufwand im Knoten

Hilfsmittel:

Die Metriken zur Berechnung des Knotenaufwands benutzen die Werte der Templatemetriken.

Es werden alle Dokumente bzw. Templates eines Knotens gleich gewichtet. Eine differenzierte Gewichtung der Templates kann aufgrund fehlender Erfahrung über die Bedeutung der Dokumente für das Experiment nicht vorgenommen werden.

Ziel:	**Aufwandsschätzung für ein Experiment**
Fragen:	Wie viele Templates enthält das Experiment? Wie viel Zeit wurde für das Experiment eingeplant? Wie ist das Verhältnis von benötigter Zeit zu eingeplanter Zeit? Wie ist der Aufwand auf die einzelnen Knoten verteilt?
Metriken:	**M6: geschätzter Aufwand des Experiments** Berechnet den Aufwand des gesamten Experiments anhand der einzelnen Knotenaufwände. $A_{geExp}(\exp) = \sum_{k \in \{Knoten\}}^{\#Knoten} A_{geKnoten}(k)$ A_{ge}: geschätzter Aufwand des Experiments

	M7: geschätzter Aufwand im Vergleich zum tatsächlichen Aufwand Vergleicht den geschätzten Aufwand des Experiments mit dem (bisher) benötigten Aufwand. Der benötigte Aufwand berechnet sich als Summe aller benötigten Zeiten für im Experiment befindliche Dokumente. $A_{tatExp} = A_{geExp}(\exp) - \sum_{\in\{Knoten\}}^{\#Knoten} A_{beKnoten}(k)$ A_{taExpt}: Personenstunden, die vom geschätzten benötigten Aufwand zum Ausfüllen aller im Experiment benötigten Dokumente noch übrig sind Eine negative Zahl bedeutet, dass die geschätzte Zeit schon überschritten wurde **M8: geschätzter Aufwand im Vergleich zum eingeplantem Aufwand** Vergleicht die eingeplante Zeit für das Experiment mit dem geschätzten Aufwand. $t_{diff} = A_{eingeplant}(\exp) - A_{ge}(\exp)$ t_{diff} Abweichung der eingeplanten Zeit vom geschätzten Aufwand $A_{eingeplant}$: eingeplante Zeit für das Experiment

Hilfsmittel:

Es werden die Werte der Knotenmetriken benötigt.

Außerdem ist eine Angabe zur eingeplanten Zeit für das Experiment erforderlich.

4.3 Metriken zur Komplexitätsbestimmung von Experimenten

Ebenfalls mit Hilfe der GQM-Methode wurden vier Metriken definiert, welche dazu dienen, die Produktkomplexität von Experimenten zu vergleichen.

Hierbei wurden die in Kapitel 4.1 ermittelten Komponenten, die zur Charakterisierung von Experimenten dienen können, herangezogen.

Ziel:	**Produktkomplexität eines Experiments bestimmen**
Fragen:	Wie viele Templates/Dokumente enthält das Experiment? Aus wie vielen unterschiedlichen Arbeitsschritten besteht das Experiment? Wie viele Hypothesen sind auszuwerten/ wurden ausgewertet?
Metriken:	**M9: Anzahl Templates im Experiment** Es werden alle Templates im Experiment gezählt Dargestellt werden die Werte auf einer Intervallskala. Das Minimum ist die kleinste Anzahl an Templates, die in allen darstellbaren Experimenten möglich ist. Das Maximum entspricht der größtmöglichsten Anzahl an Templates **M10: Anzahl Dokumente im Experiment** Die Anzahl der vorhandenen Dokumente wird auf einer Absolutskala dargestellt. Diese ist nach oben unbeschränkt **M11**: **Anzahl der Knoten im Experiment** Es werden alle Knoten im Experiment gezählt. **M12: Anzahl Hypothesen im Experiment** Über eine Abfrage in einem entsprechenden Knoten wird die Hypothesenanzahl ermittelt.

Hilfsmittel:

Der Knoten, in dem die Hypothesenanzahl eingegeben wird, muss zu jedem Experiment gehören.

5 Entwurf

5.1 Der Experimentgraph

Der Entwurf des Experimentgraphen orientiert sich streng an den Anforderungen aus Kapitel 3.2. Er wird anhand des Aufbaus des Test-First Experiments entwickelt.

Ausgehend von einem auf der Grundlage dieses Experiments spezifizierten Pfades wird der Graph um weitere Knoten, die Alternativen zu den bereits entwickelten bieten, erweitert.

Anschließend wird anhand des theoretischen Aufbaus von Experimenten (s. Kapitel 2.5) geprüft, ob weitere Pfade und Verzweigungen möglich sind.

Der auf diese Weise entwickelte Graph ist im Anhang dargestellt.

5.2 Templates

Templates und Dokumente bilden die Grundlage für die Berechnung des Aufwands eines Experiments. Aufgrund der knapp bemessenden Zeit sowie dem Mangel an auszuwertenden Experimenten können diese Templates inhaltlich nicht spezifiziert werden.

Es wird sich deshalb darauf beschränkt, die Templates mit Namen zu benennen.

Die folgende Liste enthält die im Experimentgraphen benannten Templates, die geschätzten Ausfüllzeiten für entsprechende Dokumente sowie die zunächst angenommene benötigte Häufigkeit(allgemein, pro Datenerhebung, pro Versuchsperson):

- Experimentfrage-Template (Zeit=1 PerStd, 1, 0, 0)
- Hypothesen-Template (Zeit=5 PerStd, 1, 0, 0)
- Teilnehmerliste (Zeit=3 PerStd, 1, 0, 0)
- Aufgabestellung für Experimentatoren (Zeit=5 PerStd, 1, 0, 0)
- Aufgabenstellung für Teilnehmer (Zeit=5 PerStd, 1, 0, 0)
- Schulungsmaterialien-Template für alle Versuchspersonen (Zeit=3 PerStd, 1, 0, 0)
- Spezielle Schulungsmaterialien-Template (Zeit=3 PerStd, 2, 0, 0)
- Datenerhebungsbogen-Template (Zeit=4 PerStd, 1, 1, 0)
- Auswertungs-Template (Zeit=6 PerStd, 1, 0, 0)

Die Spezifizierung der zum Ausfüllen dieser Templates benötigten Zeiten erfolgt auf der Grundlage der Erfahrungen im Test-First Experiment. Da die Zeiten hier nicht explizit gemessen wurden, müssen erst einmal geschätzte Werte eingetragen werden. Auch die benötigte Häufigkeit wird zunächst geschätzt. So wird z.B. angenommen, dass im Durchschnitt drei unterschiedliche Datenerhebungsmethoden und Auswertungsverfahren verwendet werden, die entsprechenden Templates deshalb dreimal benötigt werden.

Eine Entwicklung neuer Templates kann nach und nach erfolgen. Aufgrund der Entwurfentscheidungen zur Datenspeicherung (s. Kapitel 5.3) können diese einfach in das Programm integriert werden.

5.3 Datenspeicherung

Die Informationen des Experimentgraphen sowie der einzelnen Experimente werden in XML-Dateien gespeichert.

Eine andere Speichermöglichkeit wäre der Einsatz einer Datenbank gewesen. Es wurde XML gewählt, da die zu speichernde Datenmenge relativ klein ist und die Konfiguration und Benutzung einer Datenbank im Verhältnis zu dieser Datenmenge zu aufwendig ist.

XML ist lizenzfrei, plattform- und herstellerunabhängig und bietet so zusammen mit der Programmiersprache Java ein großes Maß an Flexibilität und Portabilität [15].

Mit Hilfe von XML-Schematas oder Document-Type-Definitions (DTDs) lässt sich die erlaubte Struktur von XML-Dateien festlegen [15].

XML-Schematas sind mächtiger als DTDs. Sie erlauben z.B. das Festsetzen eigener Datentypen, während man bei DTDs nur auf drei vordefinierte Datentypen zurückgreifen kann.

Für das Werkzeug wurde je Schema zur Speicherung der Daten des Experimentgraphen und der Daten von Experimenten definiert.

Java stellt verschiedene Parser zur Verfügung, um XML-Daten aus XML-Dateien auszulesen. Ein XML-Parser kann ein Schema zusammen mit einer XML-Datei verwenden, um deren syntaktische Korrektheit zu überprüfen [16].

Es existieren zwei unterschiedliche Schnittstellen zum Parsen von XML-Dateien. Die so genannten SAX-Parser unterstützen lediglich das Einlesen von XML-Dokumenten. Sie erzeugen bei jedem gefundenen Tag ein Ereignis, die Datei wird also sequentiell abgearbeitet.

DOM-Parser hingegen laden ein komplettes XML-Dokument in den Hauptspeicher und erzeugen dort eine Baumstruktur. Auf diese kann nicht nur beim direkten Auslesen der Informationen, sondern auch später noch zugegriffen werden. Zusätzlich ist das Manipulieren der Daten möglich, der veränderte Baum kann wieder abgespeichert werden.

Zum Parsen der Experimentgraphdatei und Experimentdateien wird ein DOM-Parser verwendet. Dieser ist zwar speicherintensiver als ein SAX-Parser, das Erstellen des Experimentgraphen wird dadurch für die Anwendung jedoch einfacher. Ein weiterer Vorteil des DOM-Parsers ist die Funktionalität, aus den Daten der Anwendung heraus einen Parsebaum erzeugen zu können und diesen mit Hilfe eines Serialisierers in eine XML-Datei zu schreiben. Auf diese Weise ist ein relativ einfaches Speichern der Experimente möglich.

Da XML-Dateien reine Textdateien sind, ist das Ändern und das Erweitern der gespeicherten Daten um neue Datensätze verhältnismäßig leicht. Auch eine Änderung der Datenstruktur ist mit grundlegenden Kenntnissen von XML schnell möglich.

5.3.1 Daten des Experimentgraphen

Die XML-Datei, welche die Daten des Experimentgraphen speichert, wird bei jedem Programmstart ausgelesen. Aus den in ihr enthaltenen Informationen wird sowohl die Struktur des Graphen als auch sein Inhalt erzeugt. Da ein Ändern dieser Informationen über die Benutzerschnittstelle nicht möglich ist, müssen diese Informationen beim Beenden des Programms nicht abgespeichert werden.

Die folgenden Daten werden gespeichert:

Knoten mit folgenden Inhalten:

- kurzer Beschreibungstext
- Vorgängerknoten
- Templates
- Koordinaten

Zu den Templates werden folgende Informationen benötigt:

- wie häufig muss das Template unabhängig von anderen Faktoren ausgefüllt werden
- geschätzte Ausfüllzeit
- wie häufig pro Person muss das Dokument ausgefüllt werden
- Wie häufig pro Datenerhebung muss das Dokument ausgefüllt werden

Da Kanten im Graphen über keine zusätzlichen Informationen verfügen werden sie nicht als eigene Elemente abgespeichert.

Als Wurzelelement wird ein Element mit Namen ***graph*** definiert.

Dieses muss mindesten zwei ***node*** Elemente (Start- und Endknoten) als Unterelemente enthalten. Die maximale Anzahl von Knoten ist nicht festgelegt.

Jeder Knoten erhält eine eindeutige ***ID*** in Form eines Attributs. Über diese ID können die Vorgängerknoten referenziert werden. Auf diese Weise ist das Festsetzen aller Kanten mit Hilfe der von Referenzangaben möglich.

Jeder Knoten erhält außerdem die zwei Attribute ***xwert*** und ***ywert***, welche Koordinatenangaben enthalten. Diese geben die Positionen an, an denen die entsprechenden Knoten auf der graphischen Oberfläche dargestellt werden.

Da kein Layoutalgorithmus zur Anordnung der Knoten zu finden war, fiel die Wahl auf dieses explizierte Festlegen der Positionen der Knoten.

Das Element ***name***, über das jeder Knoten verfügt, enthält den Titel des Knotens.

Jeder Knoten muss als Unterelement genau einmal das Element ***description*** enthalten. In diesem wird die Beschreibung des Auswahlkriteriums/Arbeitsschritts in einem kurzen Text festgehalten.

Ein Knoten kann beliebig viele Unterelmente vom Typ ***template*** haben. Dieses Element verfügt über die Attribute

- ***expectedTime*** (geschätzte Ausfüllzeit in Personenstunden)
- ***numberOfDataCollection*** (wie oft wird das Dokument pro Datenerhebung benötigt)
- ***numberOfPerson*** (wie oft wird das Dokument pro Versuchsperson benötigt)

Der Wert des Template-Elements enthält den Namen des Templates.

```
<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="node" type="IDREF"
                    use="required" />
            </complexType>
            <complexType name="template_type" type="string">
                <attribute name="expTime" type="integer"
                use="required"/>
                <attribute name="number"
                    type="integer"
                    use="required" />
                <attribute name="numberOfDataCollection"
                    type="integer"
                    use="required" />
                <attribute name="numberOfPerson" type="integer"
                    use="required" />
            </complexType>
        </xs:element>
</xs:schema>
```

Abb. 6: Schema für den Experimentgraphen

5.3.2 Daten der Experimente

Zum Speichern der Experimentinformationen wird ein eigenes Schema verwendet. Die bereits im Experimentgraphen enthaltenen Knoteninformationen müssen nicht mit abgespeichert werden. Es genügt, lediglich die Referenz auf den Knoten zu speichern.

Darüber hinaus verfügt ein Experiment über eigene Dokumente. Diese sind den Templates des Experimentgraphen zugeordnet, sie müssen Angaben zur benötigten Ausfüllzeit enthalten.

Außerdem werden zur Berechnung der Metriken Angaben zur Versuchspersonenzahl und Datenerhebungshäufigkeit benötigt.

Das XML-Schema für Experimentdateien hat das Wurzelelement ***experiment.***

Dieses verfügt je genau einmal über die Unterelemente ***name***, ***numberOfSubjects***, ***numberOfDataCollection*** und ***scheduledTime*** welche den Name, die Anzahl der Versuchspersonen, die Häufigkeit der Datenerhebung und die für das gesamte Experiment eingeplante Zeit enthalten. So lange zu den letzten drei Attributen noch keine Angaben gemacht wurden, werden ihre Werte mit 0 angenommen.

Ein Experiment muss mindestens über ein ***node*** Element verfügen. ***Node*** Elemente besitzen ein Attribut ***ID***, welches die ID des entsprechenden Knotens des Experimentgraphen enthält. Darüber hinaus können ***node*** Elemente ***document*** Elemente als Unterelemente besitzen.

Ein ***document*** Element besitzt die Attribute ***template*** und ***attendedTime***. ***Template*** enthält den Namen des zugehörigen Templates und ***attendedTime*** die zum Ausfüllen des Dokuments benötigte Zeit.

```
<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>
```

Abb. 7: Schema für Experimente

5.4 Visualisierung des Graphen

Zur Darstellung der Graphinformationen wird eine vorhandene Graphenbibliothek verwendet. Die Wahl fiel dabei auf eine Kombination der opensource Javabibliotheken JGraph (Version 1.3) und JGraphT (Version 0.5.3).

JGraph ist zu basiert auf Java und Swing. Es ermöglicht die Darstellung von unterschiedlichen Graphen. Aufgrund der Vielfältigkeit von JGraph ist die Verwendung trotz einer umfassenden API-Dokumentation recht schwierig. Deshalb wird zusätzlich die JGraphT- Bibliothek verwendet.

JGraphT ermöglicht ebenso wie JGraph die Konfiguration unterschiedlicher Graphentypen. Für den Experimentgraphen wird ein gerichteter Graph verwendet.

Bei einer Kombination der beiden Bibliotheken werden die jeweiligen Vorteile ausgenutzt. JGraphT erlaubt eine einfache Verwaltung der Konteninformation(Datenmodell), während JGraph auf die Visualisierung von Graphen optimiert ist.

5.5 Graphical User Interface

Die Gestaltung des Graphical User Interfaces erfolgt mit Swing. Swing ist eine Bibliothek für die Gestaltung graphischer Anwenderschnittstellen, die fester Bestandteil des Java Development Kit ist. Da Swing zu 100% in Java implementiert ist, erlaubt es ein einheitliches Look&Feel auf allen Plattformen. Des Weiteren erfüllen die einzelnen Komponenten von Swing die gängigen Regeln der Usability von Benutzeroberflächen.

5.5.1 Graphical User Interface des Hauptprogramms

Die Oberfläche besteht aus einem Hauptfenster das in drei Bereiche unterteilt ist.

Am oberen Fensterrahmen befindet sich eine Menüleiste. Über diese Menüleiste ist das Laden und Speichern sowie das Anlegen neuer Experimente möglich. Außerdem gibt es einen Menüpunkt, über den die einzelnen PlugIns geladen oder beendet werden können.

Im oberen linken Bereich, der ungefähr ¾ der Fensterhöhe und Fensterbreite einnimmt, wird der Experimentgraph dargestellt.

Darunter liegt der Bereich zur Anzeige der Knoteninformationen. Hier werden Detailinformationen des gerade im Graphen aktivierten Knotens angezeigt.

Dazu ist dieser Bereich in drei Abschnitte eingeteilt. In ersten und zweiten Abschnitt wird je eine Liste mit den Namen der Templates bzw. der Dokumente angezeigt. Über entsprechende Buttons in diesen Abschnitten ist ein Download der Templates sowie ein Upload und Löschen von Dokumenten möglich. Der dritte Abschnitt enthält ein Eingabefeld. Hier werden zum einen bereits vorhandene Notizen des selektierten Knotens angezeigt zum anderen hat der Benutzer die Möglichkeit neuen Text einzugeben. Über einen Button kann er den Text Knoten hinzufügen.

Der rechte Bereich des Fensters steht für die PlugIns zur Verfügung. Ist kein PlugIn geladen, so wird der Bereich, welcher den Experimentgraphen enthält auf die komplette Fensterbreite ausgedehnt.

Die momentane Gestaltung dieses Bereiches ermöglicht die Anzeige von zwei PlugIn-Oberflächen. Hierfür wurde der zur Verfügung stehende Raum in eine untere und obere Hälfte unterteilt. Das erste PlugIn, das geladen wird, wird immer im oberen Teil, das zweite grundsätzlich im unteren dargestellt. Sollte nur ein PlugIn geladen werden, bleibt die untere Hälfte leer.

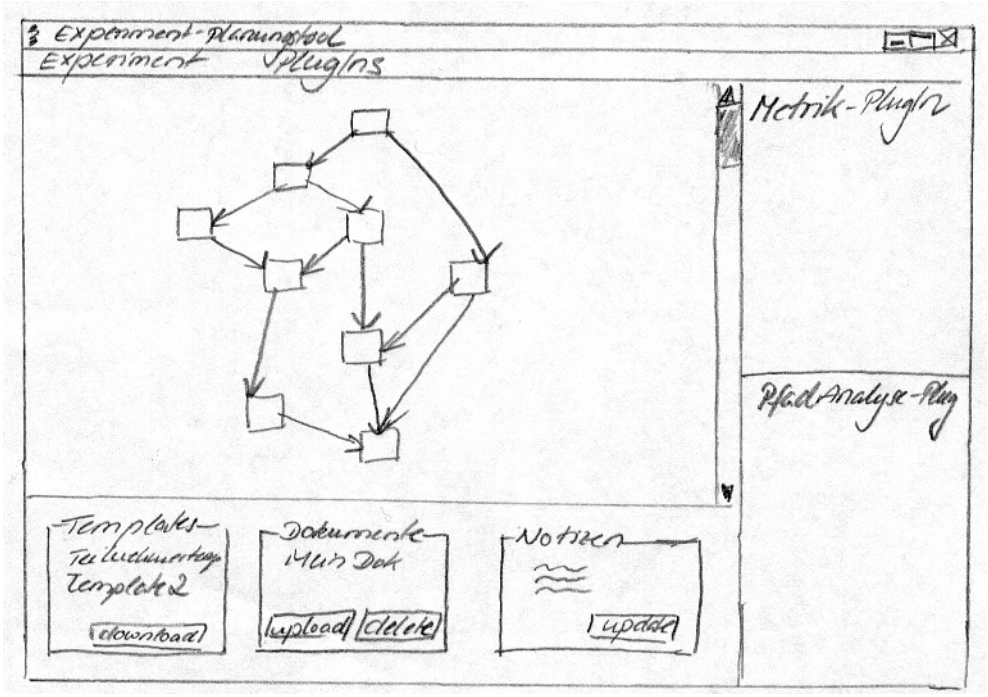

Abb. 8: Entwurf der Oberfläche

5.5.2 Graphical User Interface des PlugIns

Die PlugIn-Schnittstelle schreibt vor, dass die Hauptkomponente der Oberfläche ein JPanel-Objekt ist.

Dieses JPanel-Objekt wir mit Hilfe eines JTabbedPane-Objekts gestaltet. Ein JTabbedPane benutzt das bekannte Karteireiterprinzip.

Der Vorteil dieser Gestaltungsvariante ist, dass relativ einfach neue Reiter hinzugefügt werden können, so dass z.B. die Anzeige von weiteren Metriken ohne eine komplette Neugestaltung der Oberfläche ermöglicht wird. Außerdem kann auf verhältnismäßig geringem Raum eine große Menge von Daten dargestellt werden.

Es gibt je einen Reiter für Templatemetriken, Knotenmetriken, Experimentmetriken und Produktmetriken.

Jede dieser Teiloberflächen wird entsprechend der anzuzeigenden Informationen der einzelnen Metriken gestaltet.

In jeweils einer Liste werden Templates und Knoten des aktuellen Experiments angezeigt. Der Benutzer hat die Möglichkeit, jeweils einen Listeneintrag zu selektieren. Auf der Oberfläche werden dann die Werte des selektierten Eintrags für die entsprechenden Metriken dargestellt.

Eine Konfiguration der Metriken über die Oberfläche ist nicht möglich.

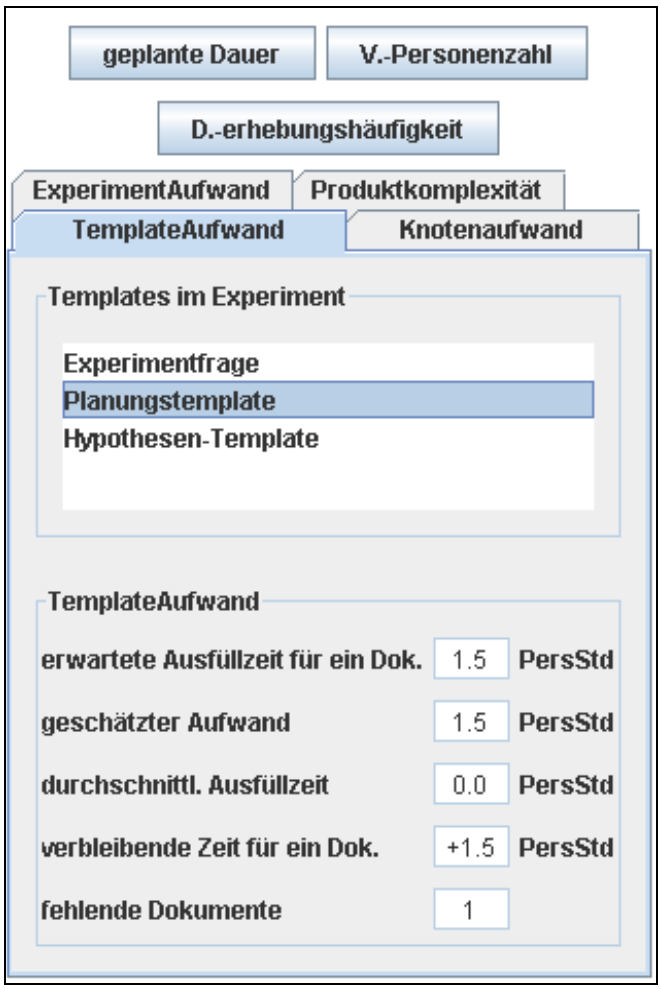

Abb. 9: Oberfläche des PlugIns

5.6 Das Model–View–Controller-Pattern

Um die Flexibilität des Programms sowie die Wartbarkeit zu erhöhen erfolgt eine strikte Trennung von Daten, Präsentation und Interaktion.

Zu diesem Zweck wird das Model-View-Controller-Pattern(MVC) [17] eingesetzt. Das MVC-Pattern ermöglicht den Austausch sowie das Anbinden verschiedener Darstellungsformen an einen Datensatz.

Da dieses Design-Pattern mittlerweile zu den bekanntesten gehört und in sehr vielen Programmen zum Einsatz kommt, wird auf eine genauere Erklärung des Prinzips verzichtet. Die folgende Abbildung soll noch einmal den strukturellen Aufbau sowie den Informationsaustausch zwischen den einzelnen Komponenten verdeutlichen.

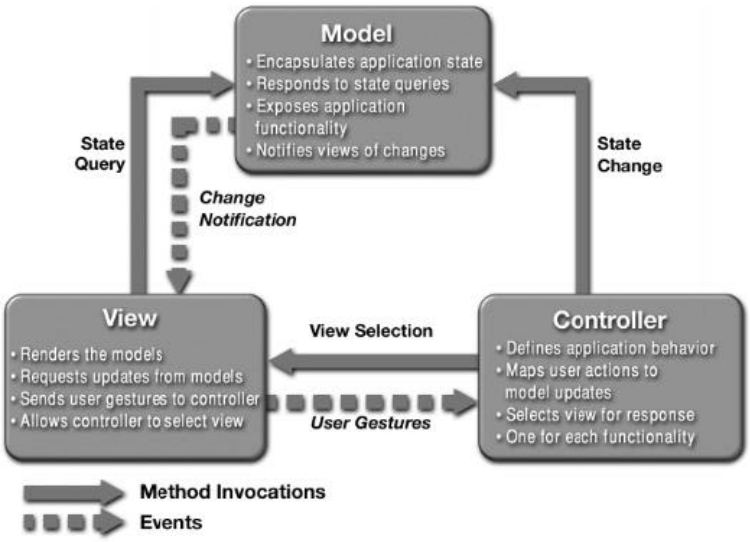

Abb. 10: Model-View-Controller-Pattern [18]

5.7 Packages und Ordner des Hauptprogramms

Das Hauptprogramm ist in sechs Packages unterteilt.

Darüber hinaus gibt es die drei Ordner „Experimente“ und „Templates“ und „PlugIns“.

Die Funktion der einzelnen Packages und Ordner sowie ihre Beziehungen untereinander werden im Folgenden erläutert.

5.7.1 Package interfaces

Die Implementierung des gesamten Programms orientiert sich an dem Grundsatz

> *„Program to an interface, not to an implementation”* [17]

Die Nutzung von Interfaces in Java führt zu einer erhöhten Abstraktion und Modularisierung des Codes. Dadurch, dass im Programm nur Interfaces zum Zugriff auf Funktionalitäten verwendet werden, wird die Abhängigkeit zur konkreten Implementierung auf ein Minimum reduziert. Konkrete Klassen können ohne größere Eingriffe in den restlichen Quellcode ausgetauscht werden, es muss lediglich dasselbe Interface implementiert werden

Das *interface Package* enthält alle im Hauptprogramm verwendeten Interfaces sowie die Schnittstellen für die Implementierung der PlugIns.

Im Folgenden werden diese Interfaces kurz vorgestellt.

Für den Datenzugriff existieren die folgenden Interfaces:

IGraphAccess	auslesen der Graphinformationen und erstellen des Graphenschemas
IExperimentAccess	auslesen der Experimentinformationen und laden des Experiments
IDocumentAccess	speichern und laden sowie kopieren von beliebigen Dateien

Interfaces, welche Controllerfunktionen zur Verfügung stellen:

IGraphListener	erbt von Mouselistener Klassen, die auf das Selektieren von Graphelementen (Konten) reagieren, müssen dieses Interface implementieren
IButtonListener	erbt von ActionListener Klassen die auf Benutzerinteraktionen auf Buttons reagieren müssen dieses Interface implementieren
IMenuListener	erbt von Actionlistener Klassen, die auf Menüereignisse reagieren, müssen dieses Interface implementieren

Interfaces für die graphische Oberfläche:

IMainFrame	Interface für das Hauptfenster
IGraphView	Klasse, die für Visualisierung des Graphen zuständig ist muss dieses Interface implementieren ermöglicht Änderung der Knotenfarbe

Interfaces zur Datenhaltung:

IExperimentGraph	ermöglicht das Setzten und Abfragen vom aktuellen Experiment und aktiviertem Knoten Schnittstelle, an der sich die Darstellungsklassen anmelden können Positionierung der Knoten
IExperiment	Methoden zum Hinzufügen und Entfernen von Knoten Schnittstelle zur Anmeldung von PlugIns Methoden, um angemeldete PlugIns über Änderungen am Experiment zu informieren

INode	Klassen, welche den Inhalt von Knoten repräsentieren, müssen dieses Interface implementieren Methoden zur Zustandsabfrage (Name, Tooltip) und Zustandsänderung (Templates, Dokumente, Markierung, Notizen)
IDocument	ermöglicht das Setzen und Abfragen von Zeiten und Templates eines Dokuments
ITemplate	die Klasse, welche zur Bildung von Template-Objekten zuständig ist, muss dieses Interface implementieren erlaubt die Abfrage der Metainformationen eines Templates (Zeit, Personenabhängig, Datenerhebungsabhängig)

PlugIn-Schnittstelle:

IPlugIn	die Hauptklasse von jedem PlugIn muss dieses Interface implementieren ermöglicht dynamisches Laden der entsprechenden Klasse stellt Methode zur Verfügung, um das PlugIn mit dem Experimentgraphen, einem Experiment sowie der verfügbaren Oberfläche zu initialisieren
IGraphChangedListener	die Klasse des PlugIns, welche dieses Interface implementiert, kann sich beim Experiment als Observer anmelden und wird dann über Änderungen im Hauptprogramm informiert

5.7.2 Package dataAccess

In diesem Packages befinden sich die konkreten Klassen für das Auslesen der Graph- und Experiment-Informationen, sowie zum Up- und Download von Dokumenten.

GraphAccess	übernimmt das Auslesen der XML-Datei, welche die Graphinformationen enthält. Sie erstellt aus diesen Informationen den Graphen.
ExperimentAccess	ist für das auslesen und speichern der Experimente in Form von XML-Dateien zuständig.
DocumentAccess	ermöglicht das Kopieren von Dokumenten sowie das Speichern und Laden dieser Dokumente aus oder in das Dateisystem

5.7.3 Package data

Dieses Package enthält sämtliche Klassen, die zur Datenhaltung benötigt werden.

Experimentgraph	wurde als Singleton realisiert; zum einen gibt es nur einen Experimentgraphen, was die Verwendung dieses Design-Patterns an sich schon rechtfertig, zum anderen bietet diese Entwurfentscheidung noch einen weiteren Vorteil: durch die statische Instanzvariable ist ein Zugriff auf den Graphen aus allen anderen Klassen heraus möglich, ohne dass diese eine entsprechende Referenz halten müssen; der Experimentgraph dient deshalb als Zugriffsklasse auf das gerade geladene Experiment; zur selben Zeit kann immer nur ein Experiment geladen sein, welches in der entsprechenden Instanzvariable des Graphen gespeichert wird.
Experiment	enthält im wesentlichen Variablen zum Speichern der Knoten sowie entsprechende Zugriffsmethoden darüber können experimentspezifische Informationen wie Versuchspersonenanzahl, Datenerhebungshäufigkeit und Experimentname in Objekten von diesem Typ gespeichert werden
ExperimetalNode	hält Referenzen auf Templates und Dokumente des gerade geladenen Experiments und ermöglicht ein Arbeiten mit diesen.
Edge	repräsentiert die Kanten
Template	speichert die Metainformationen der Templates und ermöglicht die Abfrage von diesen
MyDocument	jedes Dokument besitzt eine Referenz auf ein Template, dem es zugeordnet ist (oder explizit keine Zuordnung) außerdem werden an Document-Objekten die benötigten Ausfüllzeiten gespeichert

Dem Model im Sinne des MVC-Patterns entspricht die Klasse Experimentgraph. Bei ihr müssen sich die graphischen Klassen, welche die Detailinformationen der einzelnen Knoten darstellen, anmelden.

5.7.4 Package handler

In diesem Package befinden sich die Klassen, welche im Sinne des MVC-Patterns Controllerfunktionen übernehmen.

MenuListener	überwacht die Auswahl der einzelnen Menüpunkte
ButtonController	reagiert auf sämtliche Eingaben, die der Benutzer an Buttons auf der Oberfläche vornimmt
GraphListener	stellt Methoden zur Verfügung, welche die Selektion von Knoten im Graphen behandeln; dabei wird zwischen einem einfachen Klick und einem Doppelklick unterschieden.

5.7.5 Package gui

Im *gui Package* befinden sich alle Klassen zur Erzeugung von Oberflächenobjekten. Außerdem enthält das Package sämtliche Dialogklassen.

Das Hauptfenster wird dabei aus mehreren Einzelkomponenten zusammengesetzt. Auf dieses Weise wird das Umgestalten der Oberfläche erleichtert.

MainFrame	das Hauptfenster
EMenuBar	die Menüleiste des Hauptfensters mit sämtlichen Einträgen
EInfoPanel	der untere Teil im Hauptfenster, in dem die Knoteninformationen angezeigt werden
GraphScrollPane	der Hauptteil des Fensters, in dem der Graph angezeigt wird
PlugInPanel	der Teil der Oberfläche, der für die PlugIns zur Verfügung steht
TempDocDialog	Dialog, über den der Benutzter die Zuordnung von Dokumenten zu Templates vornimmt
TimeDialog	Dialog, in dem die Ausfüllzeiten zu den Dokumenten angegeben werden

5.7.6 Package main

In diesem Packages befindet sich lediglich die Klasse ***main***. Sie besitzt die *main()*-Methode und dient zum Instanziieren der zum Starten des Programms benötigten Objekte.

5.7.7 Ordner Experimente

Im ***Experiment-Ordner*** werden die Schemadateien sowie die XML-Datei gespeichert, welche die Graph-Informationen enthält.

Darüber hinaus wird in diesem Ordner für jedes zu speichernde Experiment ein neuer Ordner angelegt, welcher zum einen die entsprechende XML-Datei des Experiments und zum anderen sämtliche experimentspezifischen Dokumente enthält.

Jeder dieser Ordner wird nach dem Experiment benannt.

5.7.8 Ordner Templates

Im ***Template-Ordner*** liegen alle Templates, die im Experimentgraphen enthalten sind. Ein Download von Template ist nur möglich, wenn diese sich im ***Template-Ordner*** befinden.

5.7.9 Ordner PlugIns

Der ***PlugIn-Ordner*** enthält die einzelnen PlugIns, die zusätzlich zum Hauptprogramm geladen werden können. Damit die PlugIns geladen werden können müssen diese zum einen die PlugIn-Schnittstelle implementieren, zum anderen in diesem Ordner liegen.

5.8 Das Observer-Pattern im Metric-PlugIn

Die PlugIn-Schnittstelle wurde mit Hilfe des Observer-Patterns [17] gestaltet.

Jedes PlugIn benötigt eine Klasse, die als Observer fungiert. Diese muss das Interface ***IGraphChangedListener*** implementieren und sich beim Hauptprogramm – genauer beim geladenen Experiment (dem Observable) - anmelden.

Im Metric-PlugIn übernimmt die Klasse ***GraphObserver*** diese Funktion.

Um die Erweiterbarkeit des PlugIns um neue Metriken möglichst einfach zu halten, wurde im PlugIn ebenfalls das Observer-Pattern verwendet. Das *Observable* ist in diesem Fall der ***GraphObserver***. Alle Metrikklassen, welche über Änderungen am Experiment informiert werden wollen, müssen das Interface ***IMetric*** implementieren und sich beim ***GraphObserver*** anmelden.

IMetric stellt Methoden zur Verfügung, die über das Hinzufügen oder Löschen eines Knotens oder Dokuments sowie einen Wechsel des geladenen Experiments informieren.

5.9 Packages des Metric-PlugIns

Das PlugIn ist ebenfalls in mehrere Packages unterteilt. Diese und ihre Funktionen werden im Folgenden kurz erläutert.

5.9.1 Package interfaces

Dieses Package beinhaltet sämtliche im PlugIn benötigten Interfaces. Dieses sind zum einen die Interfaces des Hauptprogramms, welche den Zugriff auf die Experimentgraph- und Experimentdaten erlauben und zum anderen eigene Interfaces des PlugIns.

Eigene Interfaces des PlugIns:

IMetric	muss von jeder Metrikklasse implementiert werden stellt sicher, dass die entsprechende Klasse über alle Änderungen im Hauptprogramm informiert werden kann
IMetricView	ermöglicht die Darstellung auf der PlugIn-Oberfläche

Jede Metrikklasse besitzt ein eigenes Interface, um den Programmierstil „*Program to an Interface*“ einzuhalten. Auf diese Wiese können die Berechnungsvorschriften für die einzelnen Metriken auch ohne weitere Eingriffe in den restlichen Quellcode vorgenommen werden. Dieses sind die Interfaces ***ITemplateMetric***, ***INodeMetric***, ***IExperimentMetric*** und ***IProductMetric***.

5.9.2 Package metrics

Das metrics Package enthält die Klassen, welche die Algorithmen zur Berechnung der einzelnen Metriken enthalten. Momentan sind dieses vier Klassen:

TemplateMetric	enthält die Algorithmen zur Berechnung des Templateaufwands
NodeMetric	enthält die Algorithmen zur Berechnung des Knotenaufwands
ExperimentMetric	enthält die Algorithmen zur Berechnung des Experimentaufwands
Productmetric	enthält die Algorithmen zur Berechnung der Produktkomplexität

5.9.3 Package gui

Jede Metrikklasse verfügt über eine eigene Oberfläche. Diese Klassen sind nach den Metriken, für die sie die Darstellung liefern, benannt (z.B. ***TemplateView***).

Damit die Darstellung der einzelnen Metriken auf der PlugIn-Oberfläche erfolgen kann, müssen diese das Interface ***IMetricView*** implementieren.

IMetricView stellt die beiden Methoden *getPanel()* und *getTitle()* zur Verfügung, welche das JPanel-Objekt, das in die Oberfläche eingefügt, sowie den Titel des entsprechenden Reiters liefern.

Außerdem gibt es die Klasse ***MainView*** welche das JPanel-Objekt instanziiert, das im Hauptprogramm auf der Oberfläche eingefügt wird.

Auf dieser Oberfläche existieren neben der Anzeige für die Metriken noch drei Buttons. Über diese können Eingabedialoge aufgerufen werden, welche die Anzahl der Versuchspersonen, die Häufigkeit der Datenerhebung und die eingeplante Zeit abfragen.

5.9.4 Package observer

In diesem Package befindet sich der ***GraphObserver,*** der Änderungen im Hauptprogramm mitgeteilt bekommt.

Des Weiteren gibt es Observerklassen, die Interaktionen des Benutzers auf der PlugIn-Oberfläche überwachen. Die Klassen ***TemplateListObserver*** und ***NodeListObserver***, reagieren auf das Selektieren von Listeneinträgen auf den entsprechenden Metrikoberflächen und leiten diese an die Metrikklassen weiter.

Außerdem gibt es eine Klasse ***ButtonListener***, welche auf Benutzerinteraktionen auf den oben spezifizierten Buttons reagiert.

6 Zusammenfassung und Ausblick

6.1 Zusammenfassung

Das Ziel dieser Arbeit war die Entwicklung eines Werkzeuges, mit dessen Hilfe sich Experimente im Software Engineering planen und analysieren lassen und das gewisse Metriken zur Abschätzung der Komplexität von diesen Experimenten zur Verfügung stellt.

Um ein solches Werkzeug zu implementieren wurden theoretische Grundlagen über den Aufbau und den Nutzen von Experimenten erarbeitet. Das Experiment, welches das Institut für Software Engineering der Universität Hannover im Wintersemester 2004/05 durchführte, wurde als Basis für den Aufbau des Programms herangezogen.

Das entwickelte Werkzeug enthält einen Experimentgraphen, mit dem sich verschiedene Experimente planen lassen. Dieser Graph beinhaltet Arbeitsschritte und Kriterien, die eine entscheidende Rolle beim Experimentdesign darstellen.

Darüber hinaus wurden ebenfalls auf der Grundlage des Test-First Experiments einige Templates benannt, welche Bestandteil des Graphen sind. Jedes dieser Templates besitzt ein Attribut, das die voraussichtlich zum Ausfüllen benötigte Zeit enthält, sowie weitere Attribute, aus denen sich die benötigte Anzahl Dokumente berechnet.

Das Werkzeug bietet die Möglichkeit, diese Templates herunter zu laden und entsprechende Dokumente zum Experiment hinzuzufügen. Bei diesem Upload von Dokumenten muss die benötigte Zeit zum Ausfüllen angegeben werden.

Mit Hilfe dieser Zeitangabe, der Versuchspersonenzahl, der Häufigkeit der Datenerhebung sowie der eingeplanten Zeit für das Template bzw. das gesamte Experiment werden einige Metriken berechnet, welche Werte liefern, mit denen der Aufwand abgeschätzt werden kann.

Das Test-First Experiment lässt sich mit dem spezifizierten Experimentgraphen abbilden. Eine Berechnung der Metriken für dieses Experiment war aufgrund fehlender Zeitangaben nicht möglich. Zur Validierung der Metriken wird in Kapitel 6.2 Stellung genommen.

6.2 Implementierte Metriken

Es wurden acht Metriken implementiert, welche auf der Grundlage der Templates und Knoten im Experiment den Aufwand abschätzen.

Basis dieser Berechnung sind dabei geschätzte Angaben des Autors eines Templates zur Ausfülldauer entsprechender Dokumente sowie die Angaben der Experimentatoren über die tatsächlich hierfür benötigte Zeit.

Folgende Kritikpunkte lassen sich an den implementierten Metriken ausmachen:

- Es ist praktisch nicht möglich, den benötigten Zeitaufwand zum Ausfüllen eines Templates objektiv einzuschätzen. Dieser kann unter Umständen von Erfahrungen der Experimentatoren oder auch von anderen experimentspezifischen Faktoren abhängen
- Die benötigte Zeit zum Ausfüllen eines Templates wird sich verringern, je häufiger man dieses ausfüllt. Derartige „Lerneffekte“ werden von den Metriken nicht berücksichtigt.
- Der Aufwand eines Experiments lässt sich nicht ausschließlich über die Templates definieren. Ein Experiment enthält noch viele andere Kriterien und Arbeiten, die einen Beitrag zum Zeitaufwand liefern.

Abgesehen von diesen Kritikpunkten denke ich, dass die implementierten Metriken einen ersten guten Ansatzpunkt bieten, um einen Überblick über den Aufwand eines Experiments zu erhalten.

Eine Anwendung der Metriken auf bereits durchgeführte Experimente wurde innerhalb dieser Arbeit nicht vorgenommen. Zur weiteren Spezifizierung der Metriken sowie zur Validierung ist es erforderlich, aus mehreren Experimenten Templates und benötigte Zeiten herauszuziehen und so die Schätzwerte anzupassen. Dies ist ein Schwerpunkt für weitere Arbeiten.

Neben den Metriken zur Aufwandsschätzung wurden drei weitere Metriken implementiert (M12: Anzahl der Hypothesen wurde nicht implementiert), welche die Produktkomplexität eines Experiments abschätzen.

Diese Metriken liefern Angaben zur Template- und Dokumentenzahl sowie zur Anzahl der Knoten im Experiment. Ihr Nutzen liegt in erster Linie darin, Experimente anhand dieser Faktoren zu vergleichen und ihren Schwierigkeitsgrad abzuschätzen.

Ob diese Faktoren tatsächlich eine geeignete Grundlage zum Vergleich bzw. zu Bestimmung der Komplexität liefern, lässt sich erst mit Hilfe von mehreren zur Analyse zur Verfügung stehenden Experimenten klären.

6.3 Ausblick

Der in dieser Arbeit erstellte Experimentgraph, auf dessen Grundlage Experimente geplant werden, liefert nur einen ersten Anhaltspunkt, wie Experimente strukturiert sind.

Je größer der Erfahrungsschatz im Umgang mit Experimenten im Software Engineering wird, umso detaillierter kann dieser Graph erweitert werden. Eine solche Erweiterung kann verschiedene Punkte enthalten:

- existierende Knoten können um Erfahrungen und Hinweise ergänzt werden
- neue Knoten (=Arbeitsschritte) können hinzugefügt werden
- Templates können genauer spezifiziert bzw. neu entwickelt werden

Ein anderer Schwerpunkt für weitere Arbeiten liegt in der Validierung der hier implementierten Metriken sowie in der Entwicklung weiterer Metriken zur Komplexitätsbestimmung.

Die Validierung umfasst die Anwendung der Metriken auf verschiedene Experimente und das Überprüfen ihrer Aussagekraft. Darüber hinaus sind die Schätzwerte zum Ausfüllen der einzelnen Templates anzupassen und gegebenenfalls genauer zu differenzieren. Dabei können z.B. Lerneffekte berücksichtigt werden oder es sind Faktoren zu identifizieren, welche die Ausfüllzeit beeinflussen.

Es wäre auch denkbar, die Experimentdaten in einer Datenbank zu speichern und diese mit einer Experience Base zu verknüpfen. Auf diese Weise können Erfahrungen in der noch relativ neuen Technik der kontrollierten Experimente im Software Engineering weitergegeben und direkt mit neuen Planungen verbunden werden.

Abbildungsverzeichnis

Literaturverzeichnis

[1] V. R. Basili, F. Shull, and F. Lanubile, "Building Knowledge through Families of Experiments," *IEEE Transactions on Software Engineering*, vol. Vol. 25, 1999.

[2] F. Houdek, "Empirisch basierte Qualitätsverbesserung. Systematischer Einsatz externer Experimente im Software Engineering," in *Institut für Informatik*: Universtät Ulm, 1999.

[3] W. F. Tichy, P. Lukowicz, L. Prechelt, and E. A. Heinz, "Experimental evaluation in computer science: A quantitative study," *Journal of Systems and Software*, pp. 9-18, 1995.

[4] T. Flohr and T. Schneider, "An XP Experiment with Students - Setup and Problems," presented at Profes2005, Oulu, Finland, 2005.

[5] I. Sommerville, *Software Engineering*, vol. 1, 5 ed. Harlow, England: Addison-Wesley Longman Limited, 1995.

[6] V. Basili, G. Caldiera, and H. Rombach, "Goal question metric paradigm," in *Encyclopedia of Software Engineering*, vol. 1, J. J. Marciniak, Ed. New York: John Wiley & Sons, 1994, pp. 528-532.

[7] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation In Software Engineering: An Introduction*. USA, 2000.

[8] L. Prechelt, *Kontrollierte Experimente in der Softwaretechnik: Potenzial und Methodik*. Berlin, Heidelberg, New York: Springer-Verlag, 2001.

[9] S. L. Pfleeger, "Experimental Design and Analysis in Software Engineering," 1995.

[10] G. E. Thaller, *Software-Metriken: einsetzen-bewerten-messen*: Verlag Technik Berlin, 2000.

[11] *Wikipedia*,http://de.wikipedia.org.

[12] I. Sommerville, *Software Engineering*: Addison-Wesley, 2001.

[13] C. Ebert and R. Dumke, *Software-Metriken in der Praxis: Einführung und Anwendung von Software-Metriken in der industriellen Praxis*. Berlin, Heidelberg: Springer-Verlag, 1996.

[14] I. F. P. U. Group, "Function Point Counting Practices Manual IFPUG," 1999.

[15] *XML*,http://www.w3.org/XML.

[16] *Java und XML - Grundlagen*: RRZN / Universität Hannover.

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

[18] *Model View Controller*,http://java.sun.com/blueprints/patterns/MVC-detailed.html.

[19] M. M. Müller and O. Hagner, "Experiment about Test-first programming," *IEEE Proceedings Software*, vol. 149, 2002.

Anhang

Anhang A) Experimentgraph