Masterarbeit

an der

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Fachbereich 5
Elektrotechnik, Informationstechnik und CMD

in der Einrichtung Simulations- und Softwaretechnik

am Deutschen Zentrum für Luft und Raumfahrt e.V.

zum Thema

Konzeption der Generalisierbarkeit und Implementierung einer automatisierten Erkennung von Usability-Schwächen am Beispiel einer Java SWT Anwendung

Geschrieben von

Simon Roder

Matr.Nr.: 393279

19. August 2014

Referent: Prof. Dr.-Ing. Thomas Ritz

Korreferent: Oliver Seebach M.Sc.

Danksagung

Diese Arbeit wurde in der Einrichtung *Software und Simulationstechnik* des *Deutschen Zentrums für Luft und Raumfahrt e.V.* ermöglicht und so möchte ich meinen herzlichsten Dank dafür an meine Betreuer Herrn Oliver Seebach und Frau Doreen Seider sowie die Kollegen der Einrichtung aussprechen. Vielen Dank für das freundschaftliche Ambiente, die stetige Unterstützung mit Rat und Tat und die netten Runden am Kicker neben der Masterarbeit.

Ebenso möchte ich mich dafür bedanken, dass alle nötigen Mittel für die Arbeit zur Verfügung gestellt wurden und es somit weder an Hard- und Software noch an Informationsquellen fehlte.

Ein besonderer Dank gilt auch dem Betreuer der Fachhochschule Aachen Prof. Dr.-Ing. Thomas Ritz, welcher mir ebenfalls eine Möglichkeit bot Unterstützung zu erhalten und mir Anregungen zu Verbesserungen meiner Arbeit unterbreitete.

Auch der Fachhochschule selber gilt mein Dank für die kostenfreien Möglichkeiten über die Hochschulbibliothek und verschiedenen Onlinebibliotheken schnell und an ausreichend Informationen zu gelangen.

Gleichermaßen möchte ich den Probanden danken, welche sich die Zeit nahmen mir bei der Nutzerstudie zur Verfügung zu stehen. Mit der Zeit und den gelösten Aufgaben konnte ich genügend Daten sammeln um die Algorithmen zu testen und dessen Ergebnisse auf Validität hin zu untersuchen.

Zudem möchte ich mich noch bei meinen Freunden und Verwandten für die vielen motivierenden Worte und die gesamte persönliche Unterstützung bedanken.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Im Verlauf dieser Arbeit werden immer wieder Bezeichnungen durch eine hellblauer Schriftfarbe kenntlich gemacht. Diese können für eine kurze Erläuterung im Glossar auf Seite 58 nachgeschlagen werden. In der elektronischen Form können diese wie auch sonstige Referenzen und andere Stellen in blauer Schrift im Literaturverzeichnis auf Seite 61 nachgeschlagen oder per Maus angewählt werden, um zu diesen Stellen zu gelangen.

1.1 Aufgabenstellung

Bei der Masterarbeit zu dem Thema „Konzeption der Generalisierbarkeit und Implementierung einer automatisierten Erkennung von Usability-Schwächen am Beispiel einer Java SWT Anwendung“ geht es darum, ein Konzept zu entwickeln potenzielle Usability-Schwächen am Beispiel der Software Remote Component Environment (RCE) des Deutschen Zentrums für Luft und Raumfahrt e.V. (DLR) automatisiert zu erkennen und aufzuzeigen. Anhand von Log-Daten und auf Basis der Master-Arbeit von Herrn Oliver Seebach

> Conception and Implementation of a Usability Problem Analysis Tool and its Comparison with another Usability Technique on the Example of a Workflow-driven Integration Environment

wird dies geschehen. Das Konzept soll ebenso prototypisch mittels aspektorientierter Programmierung (siehe Kapitel 2.2) implementiert werden und für weitere Softwareprojekte mit Java Standard Widget Toolkit (SWT) als Bibliothek zugänglich sein.

Bei der zu Grunde liegenden Arbeit wurden wegen eines anderen Fokus und Zeitgrenzen Befehle im Quellcode von RCE an den passenden Stellen zur Erstellung von Log-Dateien eingefügt. Diese werden durch fest implementierte Patterns ausgewertet und auf Usability-Schwächen analysiert.

Die Generalisierung und somit die Wiederverwendbarkeit als Bibliothek, die Anpassbarkeit und Erweiterbarkeit durch Konfigurationsdateien um neue oder speziellere Pattern steht hier im Vordergrund. Bei der Verwendung der Bibliothek sollen die Programmierer einer zu testenden Software möglichst wenig beeinflusst werden und so sollen diese die minimalen Unterschiede zur bisherigen Programmierung kaum bemerken. Die Bibliothek soll also nahezu nahtlos an

SWT anschließen, über Annotationen einzubinden sein oder anderweitig transparent Log-Daten zu den vorhandenen GUI-Komponenten und deren Nutzung erzeugen.

Für die Patterns soll eine Schnittstelle erstellt werden über die weitere Analysen implementiert werden können und so weitere Usability-Probleme aufgedeckt werden können.

Eine beispielhafte Benutzerstudie soll in kleinem Umfang von wenigen Probanden (< 10) durchgeführt werden, um eine Tendenz der neuen Methode darzustellen. Ebenso soll vorerst nur das Pattern der maximal wiederholten Befehlen Maximum Repetition Pattern (MRP) bei der Benutzerinteraktion erkannt werden und als potenzielle Usability-Schwäche gemeldet werden. Beide Einflussgrößen können bei gutem Fortschritt noch erweitert werden.

1.2 Motivation

Bei der heutigen Fülle an Angeboten freier und kostenpflichtiger Software müssen Unternehmen ihre Anwendungen positiv von der Konkurrenz abheben. Eine Möglichkeit dazu ist es die Usability der Software als Einsatz- und Kaufkriterium zu verstärken und somit dem Nutzer ein effizientes und effektives Arbeiten zu ermöglichen.

In den meisten Fällen werden dazu verschiedene Usability Expertentests eingesetzt, welche jedoch sehr zeit- und somit kostenintensiv in der Durchführung und der Auswertung sind. Zudem werden die Ergebnisse durch die Beobachtung der Probanden und die Aktionen bei z.B. der Think-Aloud Methode unbeabsichtigt beeinflusst.

Um diese Probleme zu umgehen, Kosten einzusparen und möglichst kontinuierliche Rückmeldung zu der Usability zu erhalten bietet es sich an während der Nutzung Log-Daten bei den Clients zu erstellen und diese automatisiert zu analysieren. Dadurch kann die Benutzeroberfläche immer weiter verbessert und an die Bedürfnisse der Nutzer angepasst werden.

Zudem lassen sich ausführliche Logs in einer weiterführenden Iteration erneut mit einem anderen Fokus analysieren, womit die Intensität der Analyse flexibel bleibt und beliebig viel Aufwand dort hinein gesteckt werden kann.

1.3 Herangehensweise

Im Vergleich zu den verwandten Arbeiten muss die Herangehensweise allgemeiner gefasst werden, um die Generalisierbarkeit zu erlangen und so eine allgemeingültige Java-Bibliothek für Standard Widget Toolkit (SWT) zu erhalten. Dazu werden die Komponenten von SWT genauer auf Gemeinsamkeiten, Unterschiede und potenziell beteiligte Objekte an problembehafteten Nutzerinteraktionen untersucht. Zudem werden auch zusammengesetzte, auf SWT basierte Objekte beispielsweise aus JFace oder aus Eigenkreationen genauer betrachtet. Daraus werden mögliche Ansatzpunkte extrahiert, um Nutzerinteraktionen und Zustandswechsel der Software in Log-Daten abzubilden.

Für eine spätere Analyse werden auch die Patterns zur Erkennung von Usability-Problemen auf ihre Generalisierbarkeit hin untersucht, sodass diese in Konfigurationsdateien je nach Fokus der Usability-Untersuchungen vorliegen können und anpassbar sind.

Anhand von einer oder mehreren Benutzerstudien könnten sich das Konzept und die Implementierung im Vergleich mit den bereits bewährten aber sehr aufwendigen Techniken behaupten. Damit wird der potenzielle Gewinn eines Einsatzes der Bibliothek für die Usability quantifiziert.

1.4 Mögliche Ergebnisse und deren Bedeutung

Sollte sich in der Implementierungsphase zeigen, dass das Konzept nicht so allgemein fassbar ist muss das Log-Verhalten weiterhin anwendungsspezifisch an verschiedenen Stellen des Quellcodes integriert oder angepasst werden und somit würde nur ein kleiner Teil der gewünschten Dynamik, Zeit- und Kostenersparnis erreicht werden. Ist das Konzept fassbar, so kann dies dazu führen, dass viele Softwareprodukte des DLR in den Genuss von einfachen und günstigen Usability Analysen kommen und so die Qualität der Software ansteigen könnte.

Im Vergleich zu den Benutzerstudien ließe sich bei der Erkennung ähnlicher Usability-Probleme schließen, dass die automatisierten Logs und dessen Auswertungen praxistauglich sind und somit würde es sich lohnen das Konzept weiter zu verfolgen und von der prototypischen zu einer endgültigen Implementierung zu überführen. Sollten wider Erwarten viele irreale oder kaum reale Probleme erkannt werden, so sind entweder die Pattern nicht ausreichend formuliert oder das Konzept nicht soweit generalisierbar. Ob dann die Kostenersparnis bei den Tests gegenüber dem Zeitaufwand für einen höheren Analyseaufwand noch Gewinn bringt, bleibt danach noch offen.

Generell kann vorweg genommen werden, dass die Ergebnisse einer Log-Analyse nicht so exakt sind, als die eines Expertentests. Es ist allerdings hinreichend, um Trends und schwerwiegende Probleme kostengünstig aufzuzeigen.

2 Hintergrundinformationen

In diesem Kapitel werden Grundlagen für die Lösung der in der Einleitung genannten Zielsetzung aufgezeigt. Die folgenden Unterkapitel 2.1 und 2.2 geben genauere Informationen zu den genutzten Mitteln und den zu erkennenden Usability-Schwächen. Diese Informationen werden vornehmlich aus [11], [21] und [5] entnommen und an einigen Stellen durch eigene Gedanken ergänzt. Daraufhin werden in dem Unterkapitel 2.3 die nötigen Grundlagen zu dem von der zugrunde liegende Software genutzten Plattform geschaffen und für die Umsetzung dieser Arbeit motiviert. Dieses Kapitel basiert dabei auf [29]. Im folgenden Unterkapitel 2.4 wird die beispielhaft zu erweiternde Software beschrieben. Als letztes Unterkapitel 2.5 wird das Deutsche Zentrum für Luft und Raumfahrt vorgestellt und die Einrichtung Simulations- und Softwaretechnik mit ihren Aufgaben genauer erläutert.

2.1 Usability

Usability ist nach [11] Seite 19 die Güte der Gebrauchstauglichkeit eines Produktes und lässt sich auf die Güte der Effektivität, Effizienz und Zufriedenstellung des Users mit der Nutzung des Produktes aufteilen.

Mit der folgenden Abbildung 1 nach DIN EN ISO 9241-11 von 1999 lässt sich die Vorstellung der Gebrauchstauglichkeit genauer erfassen. Die Gebrauchstauglichkeit muss immer in Bezug zu dem Nutzer und dem Ziel gesehen werden, bei dem die Arbeitsaufgabe, die Arbeitsmittel und die Arbeitsumgebung als Nutzungskontext mit Einfluss auf die Usability hat.

Dieser Nutzungskontext muss für diese Arbeit auf die softwareseitige Umgebung beschränkt bleiben, da dieser im Vorhinein bekannt und konstant ist und zu viel Dynamik nicht mehr automatisiert bei der Auswertung berücksichtigt werden kann. Zwar könnten für die nachträgliche manuelle Betrachtung über Fragebögen, Nutzerkommentare oder Ähnlichem die Situation und Umgebung des Probanden genauer erfasst werden, jedoch ist dies für die automatisierte Analyse schwierig in von Computern verwertbare Faktoren zu überführen.

2.1.1 Usability Konzepte und Methoden

In diesem Kapitel werden grundlegende Konzepte und drei ausgewählte Methoden beschrieben die Usability eines Produktes zu testen beziehungsweise

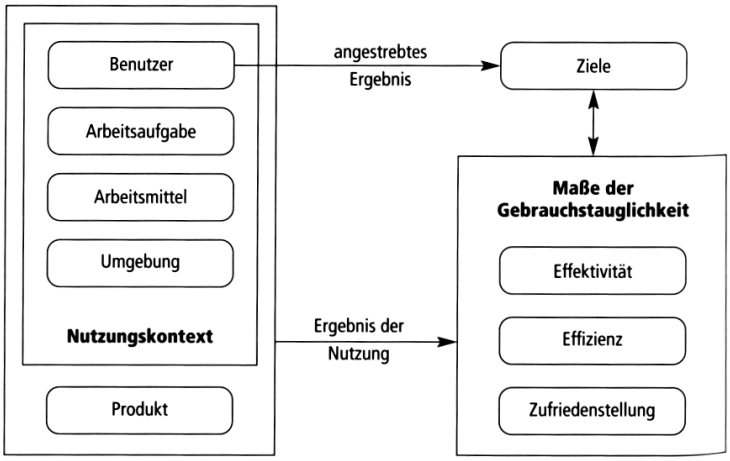

Abbildung 1: Anwendungsrahmen für Gebrauchstauglichkeit (Quelle: [11] Seite 38)

Usability-Probleme aufzudecken. Dazu zählen zum einen die Möglichkeiten und Nachteile einer *Remote-Analyse*, sowie das Konzept der *synchronen* und der *asynchronen* Usability-Analyse.

Die *Remote-Analyse* bezeichnet die Durchführung einer Usability-Analyse an räumlich getrennten Orten. Dies könnte im einfachsten Fall ein Raum neben dem Testlabor sein oder auch über das Internet auf verschiedenen Kontinenten (Abbildung 2).

Abbildung 2: Eine Remote-Analyse aus Indien eines deutschen Nutzers

Als ein großer Vorteil wird häufig die Möglichkeit von geringeren Kosten genannt. So kann die Beobachtung per Video-, Audiostream, Telefon, Online-Meeting oder auch per Fernwartung in ein Land mit geringeren Lohnkosten

ausgelagert werden oder anders betrachtet können viele über den Globus verteilte Probanden beobachtet werden. Als weiterer Vorteil können hierüber viele Analysen parallel verlaufen, ohne zuvor einen Testarbeitsplatz zu schaffen, da die Probanden von einer eigenen, gewohnten Umgebung aus die Aufgaben lösen. Diese Umgebung bietet zusätzlich auch einen negativen Aspekt, so kann der Nutzer mit einem für die Anwendung ungeeigneten Gerät arbeiten, was zu einer Fehlinterpretation der Ergebnisse führen könnte. Auf der Anderen Seite bringt diese Art des Testens eine möglicherweise sinnvolle Bandbreite an Testgeräten ein, wodurch Inkompatibilitäten schnell entdeckt und ebenso schnell behoben werden können.

Bei der *synchronen Analyse* kann auf das Verhalten jedes einzelnen Probanden eingegangen werden, Hilfestellung gegeben werden und der Fokus der Beobachtung angepasst werden (siehe Abbildung 3 links). Nachteilig sind jedoch die Tatsachen, dass eine beobachtende Person und besonders das Eingreifen in die Handlung des Probanden das Ergebnis verfälschen, da möglicherweise intuitive Handlungsweisen übergangen werden beziehungsweise mit der zusätzlichen Hilfestellung Nutzerfehler nicht auftreten.

Dies Beeinflussung wird bei der *asynchronen Usability-Analyse* deutlich gemildert, da dort gegebenenfalls nur noch das Gefühl beobachtet zu werden den Probanden beeinflussen kann (siehe Abbildung 3 rechts). Je passiver diese Beobachtungen, desto weniger treten diese Phänomene auf. Zusätzlich sind hier Zeitzonen nicht mehr von Bedeutung, so kann eine Remote-Analyse an einem anderen Ort der Welt zu den dort üblichen Arbeitszeiten geschehen.

Abbildung 3: Links eine synchrone Beobachtung und rechts eine asynchrone Beobachtung

Die in der Theorie beste Art ohne Beeinflussung aus zukommen, wäre eine unwissentliche Beobachtung des Probanden[22], was allerdings als Überwachung interpretiert werden kann und somit weder ethisch noch gesetzlich vertretbar ist. Dadurch dass die Nutzer der Log-Erzeugung bewusst zustimmen müssen, wird nach [25] die Bereitschaft zu der Analyse beizutragen gemindert. Zudem fehlen in diesem Fall auch viele Hintergrundinformationen zu dem Probanden und der aktuellen Situation und Gefühlslage dieser. Ohne Interaktion mit den

Probanden sind diese Faktoren nur schwierig zu erfassen und so bleiben die Ergebnisse nur wenig aussagekräftig. Durch die geringere Interaktion kann es dazu kommen, dass Nutzer frühzeitig aufgeben und nur wenige statt möglicherweise vieler Usability-Probleme erkennbar werden. Dadurch sinkt die Effizienz der Tests und diese müssen unter Umständen mehrfach wiederholt werden.

Die Usability Methode, welche hier angestrebt wird ist eine örtlich und auch zeitlich getrennte und somit kann die Methode des Log-Schreibens als asynchrone Remote-Analyse bezeichnet werden, welche bezogen auf die Usability viel Interpretationsspielraum bietet. Dennoch können mit dieser Methode parallel und kontinuierlich viele Verhaltensweisen beobachtet werden und schwer wiegende Probleme schnell erkannt werden.

Zu den weiteren alternativen Techniken zählen zum Beispiel die Methoden *Think-Aloud*, *Heuristic Evaluation* und *Cognative Walkthrough* mit ihren individuellen Eigenschaften, Vor- und Nachteilen.

Bei der *Think-Aloud-Methode* wird durch das unübliche Aussprechen aller Gedanken die Effizienz der Nutzer deutlich gesenkt, womit diese anfälliger für Frustration werden. Dennoch werden hier mit einem recht großen Zeit- und Arbeitsaufwand viele der Usability Probleme entdeckt.

In der *Heuristic Evaluation* prüfen mehrere professionelle Usability-Tester eine Software der Reihe nach auf alle Aspekte der Usability, wie zum Beispiel der Konsistenz, Klarheit und Effizienz. Dabei werden jedoch häufiger irreale Probleme entdeckt, welche in der Praxis nicht aufgetaucht wären.

Bei dem *Cognative Walkthrough* hingegen prüfen professionelle Usability-Tester nicht anhand der Usability-Aspekte, sondern nach Aufgaben. Bei gegebenen Aufgaben werden nur diese auf die oben genannten Aspekte geprüft und somit werden die in Aufgaben nicht vorgesehenen Möglichkeiten und Variationen nicht getestet. Zwar werden hier auch zu viele irreale Probleme (False-Positives) gefunden, doch weniger als bei der *Heuristic Evaluation*.

2.1.2 Usability Probleme

Nach [21] gibt es fünf verschiedene Gründe warum Produkte nur schlecht gebrauchstauglich sind. Diese Gründe werden in diesem Kapitel durchlaufen und geprüft, wie sie sich in dem späteren Produkt beziehungsweise dem Verhalten der Nutzer wiederspiegeln könnten.

Als erstes Problem könnte der *Fokus der Entwickler* auf dem Gerät oder System liegen: Auch wenn das Gerät noch so hochwertig, in der Laborumgebung

besonders gebrauchstauglich ist und Umwelteinflüsse nicht in der Entwicklung beachtet werden, so verliert dieses in der Praxis seine gute Usability.

Wenn etwas beispielsweise darauf ausgelegt ist mit zwei Händen bedient zu werden und diese in der Situation der Nutzung nicht zur Verfügung stehen, da zusätzlich etwas festgehalten werden muss, ist das Produkt nicht mehr praktikabel.

Solche und viele weitere äußere Umwelteinflüsse müssen mit beachtet werden, damit diese nicht zu einer schlechten Usability führen.

Ein weiteres Problem kann eine *variable Zielgruppe* sein: Werden Änderungen dieser in der Entwicklung nicht weiter berücksichtigt, so kann eine spätere Einführung zu Usability-Problemen führen.

Ein Beispiel für solch eine Situation gab es bei dem Spiel „Duke Nukem Forever“. Bei einer Entwicklungszeit von 13 Jahren und mit den in der Zwischenzeit erschienenen Titeln stiegen die Erwartungen immer weiter. Obwohl immer neuere Technologien genutzt wurden, um den zeitlichen Standards zu entsprechen, wurde im Endeffekt doch nur mittelmäßiger Erfolg durch ein mittelmäßiges Spiel erreicht[2], da vermutlich nicht die aktuellen Standards und Interessen der damaligen Fans erreicht wurden.

Die Entwicklung von besonders gebrauchstauglichen Produkten ist eine Herausforderung und somit auch selbst eines der Probleme: Es existieren *keine Pläne* und es wird auch keine geben nach denen solch ein Produkt erstellen werden kann. Jedes Produkt muss individuell an die Aufgabe und an die äußeren Einflussfaktoren angepasst werden, sodass für den Moment, die Situation, die Nutzer etc. das Produkt ein hohes Maß an Usability aufweist.

Das vierte Problem bezieht sich auf die Entwickler und an der Entwicklung beteiligte Personen selbst. Diese Personen arbeiten nicht immer gut miteinander zusammen, sodass es durch die sehr spezialisierte Ausrichtung der einzelnen Personen zu *Verständigungsschwierigkeiten* kommen kann.

Als Beispiel ist das bekannte Schaukel-Projekt aus dem Projektmanagement passend (Abbildung 4). Darin hat der Kunde eine Vorstellung, welche von jedem der Teammitglieder anders verstanden wurde und im Endeffekt wurde ein kaum nutzbares Seil am Baum mit den Kosten einer Achterbahn umgesetzt.

Zusätzlich kann es auch dazu kommen, dass durch die Aufgabenteilung Ressourcen verschwendet werden. Wenn beispielsweise eine Person etwas besonders gut kann oder daran viel Spaß hat, legt diese die Qualität und somit den zeitlichen Aufwand dessen oft größer als notwendig aus, sodass für andere für die Zielgruppe wichtige Qualitäten kaum noch Kapazitäten bleiben.

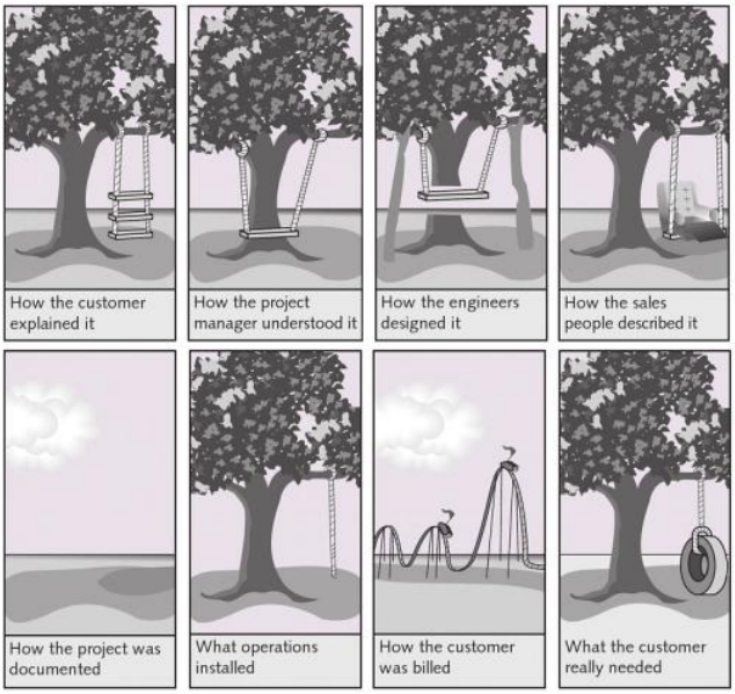

Abbildung 4: Verständigungsschwierigkeiten in einem Projekt (Quelle: [23] Seite 95)

Aus den Problemen der Zeichnung stammt auch das letzte Problem. Design und Implementierung stimmen nicht immer überein. Somit wurde statt des Designs der Ingenieure „How the engineers designed it“ nur ein Seil am Baum implementiert „What operations installed“. Mit dem Problem ist hier jedoch nicht ein Verständigungsproblem untereinander gemeint, sondern ein Problem mangels *Fähigkeiten, Budget und zeitlichen Möglichkeiten* des Teams beziehungsweise einzelner Teammitglieder.

Bei solchen Projekten ist ein gutes Ergebnis unter den genannten Einschränkungen oft nicht möglich. Beispiele hierfür sind die immer öfter verfügbaren Beta-Versionen von Software. Dieser Versionsstand verweist zwar auf eine teilweise funktionale Software, jedoch können grobe Fehler oder spätere grobe Änderungen und somit Inkompatibilitäten nicht ausgeschlossen werden. Die Software kann daher noch nicht als endgültig fertig angesehen werden und ist gerade für Firmen kaum einsatzfähig.

Diese Probleme wirken sich alle auf die Usability und somit das Nutzerverhalten aus, doch dies wird sich nicht so exakt unterteilt darstellen. Alle Nutzeraktionen können einen Anteil der Probleme wiederspiegeln. Der Nutzer wird versuchen je nach Situation und Problem entweder die Hilfe zu den benötigten Funktionen

zu durchsuchen, durch ein nicht zielgerichtetes Ausprobieren das Problem zu umgehen oder zu lösen und im schlechtesten Fall aufgeben. Auf diese Verhaltensweisen hin wird sich auch die Prüfung der Benutzeraktionen beschränken und somit zählen Hilfeaufrufe, wiederholte ähnliche Aktionen und Schließen der Software ohne zu speichern zu möglichen Mustern für eine schlechte Usability.

2.2 Aspektorientierte Programmierung (AOP)

Dieses Kapitel mit seinen Unterkapiteln beschreibt die allgemeine aspektorientierte Programmierung, sowie den genaueren Fokus auf die in Java verbreitete Programmiersprache AspectJ und den zugehörigen Werkzeugen aus dem AspectJ Development Toolkit (AJDT).

2.2.1 Das Programmierparadigma der AOP

Die aspektorientierte Programmierung ist eine Weiterführung der objektorientierten Programmierung. Beide Programmierparadigmen schaffen eine höhere Abstraktionsstufe in der Programmierung, sodass neben Befehlen und Variablen mit gekapselten Elementen gearbeitet werden kann und somit die Wiederverwendbarkeit von Quellcode gefördert wird. Hier werden allerdings nicht nur neue Eigenschaften und Funktionen zusammengefasst, sondern auch voneinander getrennt. Damit ist gewährleistet, dass gemeinsame Eigenschaften verschiedener Objekte und Klassen nicht mehrfach implementiert werden müssen, sondern nur einmal erstellt und allen passenden Objekten und Klassen zugewiesen werden können. Bei der Objektorientierung würde solch eine Änderung einen Eingriff in bereits existierende Klassen beziehungsweise eine Umstrukturierung der Vererbungshierarchie bedeuten.

An die natürliche Sprache angelehnt - können Nomen als Objekte, Verben als Funktionen und Adjektive als Aspekte implementiert werden.

> Beispiel:
> Die aktualisierbare *Dropdownliste* kann *geöffnet* werden!
> Das aktualisierbare *Label* kann sich *rot färben*!

Hierbei ist die *Dropdownliste*/*Label* das Objekt mit den Funktionen *open*/*colorizeRed* und *refresh*. Die letztere Funktion *refresh* lässt sich auch als gemeinsame Eigenschaft über ein Aspekt implementieren.

Die Vorgehensweise bei der objektorientierten Programmierung, wäre es das allgemeine aktualisierbare Objekt mit der Funktion *refresh* zu erstellen und durch Vererbung um die jeweilig nötige Funktion *open/colorizeRed* zu dem jeweilig speziellen Objekt zu erweitern.

Bei der aspektorientierten Programmierung würde hingegen das Objekt *Dropdownliste/Label* mit seiner Funktion *open/colorizeRed* erstellt, aber nicht weiter verändern oder vererbt. Es kann nun der Aspekt *aktualisierbar* als eigenständiger Quellcode erstellt werden. Im Hintergrund werden automatisiert überall dort Sprungmarken, Verweise oder anderer Quellcode eingefügt, wo die zugehörigen Bedingungen zutreffen. Diese Art verteilter gleicher Eigenschaften nennt sich Cross-Cutting Concern (C-CC). Eigenschaften dieser Art lassen sich besonders gut mit wenigen Aspekten implementieren, wodurch Fehler vermieden werden können.

Zu den typischen C-CC zählen vor allem die Protokollierung von Benutzeraktionen und Eingaben, Sicherheitsaspekte der Kommunikationsschnittstellen (Integrität und Vertraulichkeit) und die Persistenz von Daten.

Zur vereinfachten Kommunikation über Aspekte und AOP gibt es verschiedene Begriffe, welche in diesem Zusammenhang und teilweise auch als Schlüsselworte verwendet werden. Dazu zählen Pointcuts, Advices, Aspects und Join Points, welche im Folgenden genauer erklärt werden.

Pointcuts sind bei der AOP besonders wichtige Grundelemente mit deren Hilfe auf verschiedene Positionen im Quellcode verwiesen werden kann. Damit wird definiert an welchen Stellen des Quellcodes eingesprungen oder eingegriffen werden soll, um die neuen Eigenschaften und Funktionen einzubringen.

Advices definieren genauer, ob etwas vor, nach oder um einen Pointcut herum geschehen soll. Zudem kann hiermit auch auf Inhalte von Exceptions oder Rückgaben zugegriffen oder auch abgefangen werden.

Innerhalb des Advices werden das neue Verhalten und Ergänzungen definiert, was normalem Java-Code entspricht. Dieser hat zusätzlich Zugriff auf Eigenschaften des ausgelösten Pointcuts und gegebenenfalls je nach Deklaration auch auf die Übergaben und Rückgaben davon. Darunter sind beispielsweise Referenzen auf den Klassennamen, den ausgelösten Pointcut, geworfene Exceptions oder Rückgaben von Funktionen.

Aspects fassen alles zu einem Paket zusammen und umfassen somit alle zu ergänzenden Eigenschaften und Funktionen, welche in Pointcuts, Advices oder eigenständig erstellt werden.

Alles worauf Pointcuts potenziell verweisen können nennt sich *Join Points*, was als Einstiegspunkt übersetzt werden kann. Dies sind eindeutig identifizierbare Stellen im Programmcode:

- Aufrufe von über Wildcards identifizierbaren Funktionen
- Die Rückkehr aus solchen Funktionen
- Die Stelle an der eine Exception geworfen wurde
- Die Erstellung eines Objektes

2.2.2 AspectJ und AJDT

AspectJ sowie das AspectJ Development Toolkit (AJDT) sind speziell für Java entwickelte Werkzeuge und eine Programmiersprache für die AOP. Analog dazu gibt es noch weitere Werkzeugsammlungen und Sprachversionen für viele andere Programmiersprachen, wie zum Beispiel AspectC++ (C++), AspectJS (JavaScript), Aspyct (Python), phpaspect (PHP5) und viele mehr. Obwohl es neben AspectJ, AspectWerkz, Spring u.A. für die Aspekt Orientierte Programmierung (AOP) in Java verschiedene Möglichkeiten existieren, scheint nach der aktuellen Anzahl der Suchergebnisse bei Google die Verbreitung von Spring und AspectJ die größte zu sein. Auf Grund von bereits vorhandenen Grundlagen zu AspectJ beim DLR fiel die Wahl darauf.

AspectJ greift bei der AOP auf verschiedenen Verfahren zurück. Zum einen kann Load-Time Weaving (LTW) oder Compile-Time Weaving (CTW) verwendet werden. Ersteres heißt „Lade-Zeit Verweben“ und bedeutet es wird erst nach der eigentlichen Kompilierung während der Ausführung des Programms, dieses analysiert und passend dazu die neuen Funktionen und Eigenschaften mit ausgeführt. Bei Compile-Time Weaving dem „Kompilier-Zeit Verweben“ wird schon vor der Kompilierung eine statische Quellcode-Analyse durchgeführt und daraufhin der Quellcode entsprechend angepasst und so kompiliert. Dadurch ist die zweite Lösung von der Ausführungsgeschwindigkeit schneller und effizienter jedoch auch weniger flexibel. Bei beiden Verfahren muss für den AspectJ-Compiler ein Teil des Quellcodes bekannt sein, was grundsätzlich bei den Standard Compileroptionen in Java gegeben ist. In Java bleiben gewisse Klasseneigenschaften nach der Kompilierung erhalten und somit muss dort nichts besonders angepasst werden.

Um unter AJDT ein aspektorientiertes Projekt zu erzeugen kann entweder auf ein vorhandenes Java-Projekt zurückgegriffen werden und dieses durch die

AJDT konvertieren lassen oder es wird direkt ein solches Projekt erstellt. Grundsätzlich lässt sich in diesem Projekt unverändert in Java programmieren, zusätzlich können dann auch Aspekte erzeugen werden, welche als eigenständige Dateien gehandhabt werden. Darin können Pointcuts und die zugehörigen Advices erzeugt werden. Das Besondere an solch einem Projekt ist die Einbindung der in Java implementierten Runtime Bibliothek von AspectJ und die Nutzung dessen Compilers ajc.

2.3 Eclipse Rich Client Platform (RCP)

Dieses Kapitel basiert auf den Informationen aus [29] und [7]. Es zeigt die für diese Arbeit relevante Grundlagen sowie Komponenten zu Eclipse RCP und der Programmierung mit diesem Hilfsmittel.

Die Rich Client Platform (RCP) von Eclipse[6] ist eine weit verbreitete Basis, für eigenständige Entwicklungsumgebungen. Diese bietet alle Features von Eclipse als Integrated Developement Environment (IDE), welche je nach Wünschen verwendet, angepasst oder durch Plug-Ins und Funktionen erweitert werden kann.

Programme wie beispielsweise die bekannteren Entwicklungsumgebungen für Adobe Flex/Air (Flash Builder)[15], verschiedene Mikrocontroller (ONE Workbench von Beck)[9], Android (Android Developement Toolkit (ADT)[13]) und viele mehr nutzen die Eclipse RCP in der Praxis.

Grundsätzlich ist Eclipse RCP wie in der Abbildung 5 exemplarisch dargestellt aufgebaut. Die wichtigsten Komponenten für diese Arbeit sind dabei aus der in der Grafik untersten, dem Nutzer im Programm sichtbaren Schicht bestehend aus dem UI-Core mit JFace und SWT und zusätzlich gegebenenfalls noch das Framework Equinox (OSGi-Implementierung) als Komponente für die über Bundles dynamische erweiterbare und modifizierbare Anwendung. Diese Komponenten werden in dem folgenden Kapiteln 2.3.2 und 2.3.1 genauer beschrieben und erläutert.

2.3.1 SWT und JFace

Die Benutzeroberfläche (UI-Core) von Eclipse RCP basierten Anwendungen ist hauptsächlich mit dem Standard Widget Toolkit (SWT) und JFace implementiert. Diese beiden Java-Bibliotheken sind plattformunabhängig und bieten für jedes Betriebssystem eine entsprechend angepasste Optik, um dem Aussehen und Verhalten des jeweiligen Betriebssystems gerecht zu werden. Dabei greift

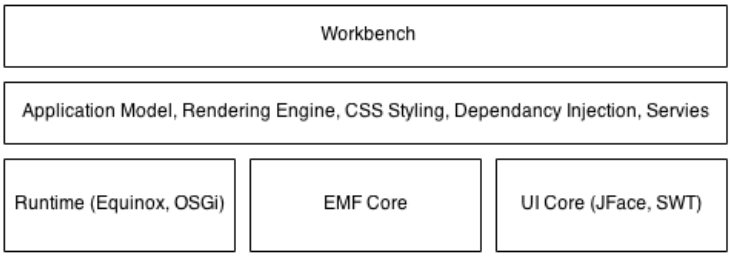

Abbildung 5: Grundsätzlicher Aufbau einer Eclipse RCP Anwendung (Vergleiche [30])

SWT auf grundlegende Komponenten des Betriebssystems zurück, sofern diese verfügbar sind. Ansonsten bildet SWT die Komponenten und deren Verhalten über Grafiken und einer eigenen Logikebene nach.

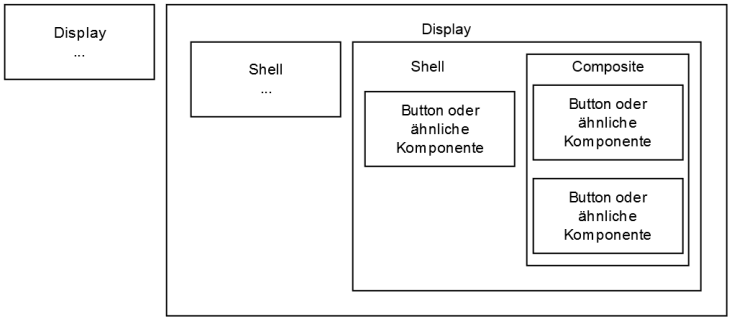

Abbildung 6: Aufbau einer grafischen Benutzeroberfläche mit SWT Komponenten

Wie in Abbildung 6 zu sehen ist werden die Elemente der GUI in einander geschachtelt, sodass ein Baum-Struktur beziehungsweise ein Wald, also eine Ansammlung von untereinander unabhängigen Bäumen, entsteht. Darin sind die einzelnen Elemente mit Referenzen zueinander versehen und können so durchlaufen werden.

JFace bietet keine direkte Benutzeroberfläche oder Elemente für diese, sondern baut auf den Komponenten von SWT auf. Damit können Assistenten, Dialoge oder andere zusammengesetzte, komplexere Benutzeroberflächen-Elemente direkt erstellt und über eine vereinfachte APIs angesprochen werden. So können auf einfache Weise beispielsweise Daten über einen Benutzerdialog gesammelt werden ohne die Dialoge im Einzelnen erstellen zu müssen.

2.3.2 OSGi und Equinox

OSGi bedeutet „Open Services Gateway initiative“ und ist eine Hardware unspezifische, modulare Plattform welche im OSGi-Framework von Eclipse genutzt wird. Dabei wird die Implementierung Equinox seit der Eclipse-Version 3.0 verwendet.

Gerade wegen dieser modularen Plattform wird Eclipse als über Plug-Ins erweiterbare Entwicklungsumgebung genutzt und so gibt es für die verschiedenen Anwendungsmöglichkeiten einer Entwicklungsumgebung eine Vielzahl von Plug-Ins. Die Entwicklungsumgebung an sich ist eine Erweiterung der Eclipse RCP, welche in einem vorigen Kapitel bereits erläutert wurde, um Funktionen und Konfigurationen bezüglich der entsprechenden Programmiersprache und des Kompilierers.

2.4 Remote Component Environment (RCE)

RCE nannte sich früher *Reconfigurable Computing Environment* und ist von dem DLR eine Software zur Entwicklung und zum Ausführen eines Workflows, einem so genannten Workflow-Management-System. Ein Workflow oder auch Arbeitsablauf definiert eine Abfolge von verschiedenen Aktionen, welche in diesem Zusammenhang auf ein Softwaresystem bezogen sind. Mit diesem Workflow-Management-System können grafisch Elemente des Workflows per Drag and Drop zusammengestellt und ohne notwendige Programmierkenntnisse genutzt werden. Auf der Internetpräsens[26] wird RCE als *Distributed, Workflow-driven Integration Environment* bezeichnet, womit eine weitere Eigenschaft der verteilten Ausführung „remote “ von RCE beschrieben wird. Mehrere Instanzen der Anwendung können auf einem Computer beziehungsweise über das Netzwerk untereinander kommunizieren.

RCE gibt es jetzt seit circa 6 Jahren als Open Source Software unter der Eclipse Public License (EPL). RCE gibt es aktuell in der Version 5.1 in drei Versionen:

- Standard
- Transport
- CPACS

Diese unterscheiden sich in einigen wenigen Features. Auf der Standard-Edition bauen die beiden anderen auf und die CPACS Edition bietet eine Unterstützung

für das Datenformat CPACS, welches von anderen Programmen des DLR verwendet wird. Die Transportedition hat diese Unterstützung nicht, jedoch bietet diese eine Excel- und eine SQL-Komponente zu der Standard-Edition. Diese Arbeit beschränkt sich auf die Standard-Edition von RCE, wobei die Aufgabe wegen derselben Grundlagen ohne Probleme auch auf diese beiden Versionen übertragbar wäre.

Als Workflow-Management-System baut RCE auf einer eigenen, grafischen, flexiblen Notation auf. Im Gegensatz zu der BPMN beziehungsweise der BPML basiert diese Sprache nicht auf dem wirtschaftlichen Anwendungsbereich, sondern ist flexibel für verschiedene wissenschaftliche Anwendungsbereiche verwendbar. Der Fokus bei RCE liegt in der Simulation von Berechnungsprozessen. Ein Workflow für Simulations- oder auch Berechnungsprozesse, wird iterativ durchlaufen, in verschiedenen kleinen Teilprogrammen berechnet und an die nächsten Teilprogramme weitergereicht bis alle Teilprogramme beendet wurden. Verschiedene externe Werkzeuge lassen sich damit ansteuern und intern bieten einem die verschiedenen Workflow-Elemente eine Möglichkeit für iterative Näherungen und Berechnungen. Erweitert werden diese Fähigkeiten durch die Möglichkeit mit Python, Jython sowie der Tool-Integration externe Simulationen, Berechnungen und Programme anzustoßen und deren Daten zu weiter zu verarbeiten.

Die Grundelemente, die Modellierung und auch die Ausführung der Werkzeuge beider Einsatzgebiete der Arbeitsabläufe haben so manche Parallelen. Einerseits wird versucht eine vereinfachte Handhabung der Programmierung durch grafische Elemente zu erreichen. Daraus ergeben sich verschiedene Elemente mit einem ähnlichen Verhalten. So gibt es beidseitig Funktionsblöcke, welche kleine Aufgaben erfüllen, Verzweigungen, Schleifen und Datenweitergaben.

Wie in Abbildung 7 zu erkennbar ist, basiert RCE auf der Version 3 der Rich Client Platform (RCP) von Eclipse.

In der dargestellten Ansicht in Abbildung 7 ist auf der linken Seite die Übersicht aller Projekte, welche bei RCE einer Sammlung von Workflows entspricht, zu sehen. Rot umrahmt mit der Nummer (1) ist einer der Workflows geöffnet, welcher über seine Elemente und deren Verbindungen eine Reihe von Aktionen und Berechnungen tätigen kann. Über die Palette (2) lassen sich Werkzeuge sowie neue Elemente dem Workflow hinzufügen. Nachdem dann die Struktur oder Teile dessen existieren können dessen Eigenschaften im unteren Bereich (2) angepasst werden. Dazu gehören zunächst In- und Outputs, welche zur Verbindung der Komponenten erstellt werden müssen. Nachdem die Inputs existieren erhält das Element noch einige Eigenschaften, wie zum Beispiel bei der Script-Komponente den auszuführenden Code per Python oder Jython. Wie

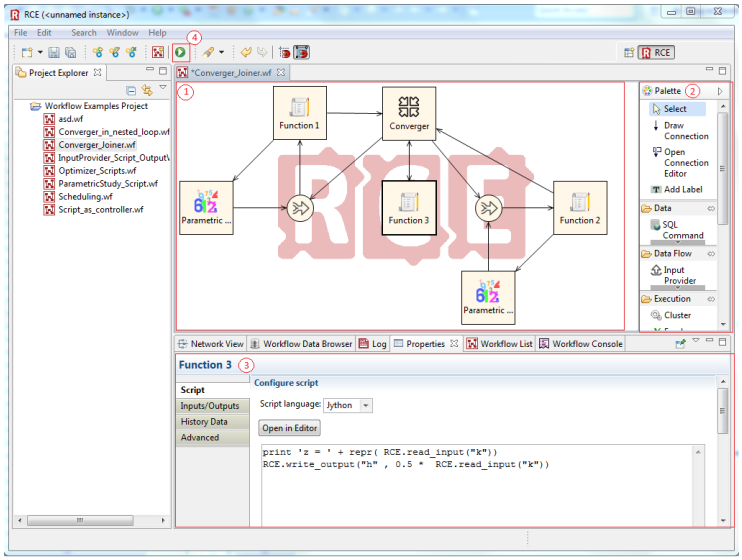

Abbildung 7: Die grafische Oberfläche von RCE

möglicherweise aus Eclipse bekannt kann das Programm oder hier der Workflow mit dem Symbol (4) ausgeführt werden.

Andere Workflow-Management-Systeme, wie das Activiti BPMN 2.0 Eclipse Plug-In, benötigen eine zusätzliche Umgebung für die Ausführung des eigentlichen Workflows und müssen noch vor der Nutzung aufwändig konfiguriert werden. RCE wird als vollständiges Paket ausgeliefert und muss nur entpackt werden.

2.5 Deutsches Zentrum für Luft- und Raumfahrt (DLR)

Das DLR [e.V.] als Forschungszentrum der Bundesrepublik Deutschland forscht nicht wie der Name vermuten lässt alleine im Gebiet Luft und Raumfahrt. Die Forschung betrifft zudem noch die Bereiche Energie, Verkehr und Sicherheit und ist damit international tätig. Mit den Forschungen zielt es auf die Interessen der Republik und entwickelt somit an Technologien für den Erhalt der Umwelt, den schonenden Umgang mit dieser in der Energiegewinnung sowie dem Verkehrswesen.

Für das gesamte Raumfahrtprogramm Deutschlands ist das DLR zuständig und erforsch so das Universum, unser Sonnensystem und die Erde selbst.

Bei der Grundlagenforschung und der Entwicklung von Zukunftstechnologien und Produkten betreibt das DLR Forschungsanlagen und bietet sich als Dienst-

leister für Wirtschaftspartner an. Weiterhin werden an den 16 Standorten regional und bundesweit Nachwuchskräfte gefördert sowie eine in der Politik beratende Funktion wahrgenommen.

Das DLR beschäftigt in seinen 32 Instituten, Test- und Betriebseinrichtungen ungefähr 7.700 Mitarbeiterinnen und Mitarbeiter an 16 Standorten in Deutschland. Zu diesen Standorten gehören Köln (Sitz des Vorstands), Augsburg, Berlin, Bonn, Braunschweig, Bremen, Göttingen, Hamburg, Jülich, Lampoldshausen, Neustrelitz, Oberpfaffenhofen, Stade, Stuttgart, Trauen und Weilheim.

International existieren noch Büros in Brüssel, Paris, Tokio und Washington D.C. zum weltweiten Austausch mit anderen Forschungseinrichtungen sowie der lokalen Betreuung von Kunden.

Von dem Budget des DLR werden circa 50 Prozent durch erworbene Drittmittel finanziert. Im Raumfahrtbereich gehen von dem um 50 Prozent höheren Budget etwa 70 Prozent in die Finanzierung der Europäischen Weltraumorganisation ESA und ungefähr 15 Prozent jeweils in das deutsche Raumfahrtprogramm beziehungsweise in die Weltraumforschung des DLR selbst. Weitere Fördermittel stehen dem Projektträger im DLR und dem für Luftfahrt zur Verfügung.

Die Mitarbeiterinnen und Mitarbeiter können sich kontinuierlich fortbilden und mittels Gleitzeit, Teilzeitbeschäftigungen und anderen Fördermaßnahmen soll eine Chancengleichheit für Jedermann geboten werden (vgl. [16]).

Die Einrichtung Simulations- und Softwaretechnik des DLR befindet sich verteilt an unterschiedlichen Standorten in Deutschland. Der Großteil der Mitarbeiter agiert von Köln und Braunschweig aus, wobei noch weitere aus Berlin und dem Homeoffice mitarbeiten.

Die Einrichtung befasst sich mit der Forschung im Bereich der Softwareentwicklung für verteilte und mobile Systeme, Software für eingebettete Systeme, Visualisierung und High Performance Computing. Bei der Entwicklung für High Performance Computing geht es vor allem um die massive Parallelisierung von Algorithmen und der Optimierung dieser zur Verwendung in großen Clustern oder Rechenzentren.

Aus diesen Aufgaben heraus ergibt sich die Einteilung in die Abteilung „Verteilte Systeme und Komponentensoftware“ und „Software für Raumfahrtsysteme und interaktive Visualisierung“, wobei diese Arbeit im Auftrag der Abteilung „Verteilte Systeme und Komponentensoftware“ begonnen wurde (vgl. [12]).

Zu Programmen der Einrichtung gehören die wenigen Folgenden sowie noch einige weitere (vgl. [17]):

- Remote Component Environment (RCE), siehe voriges Kapitel.
- RepoGuard, einer Software zur Überwachung der Commits in ein Repository anhand von gesetzten Regeln.
- TiGLViewer, eine Desktop- und Android-Anwendung zur Darstellung von 3D Daten, welche mit der TiGL Bibliothek erstellt wurden.

3 Verwandte Arbeiten

Eine aus anderen Arbeiten bekannte Alternative zu manuellen Tests ist das passive Mitschreiben der Aktivitäten eines Users und der nachträglichen Auswertung. Die Probanden werden dadurch nicht so stark beeinflusst, wie bei der Think-Aloud Methode. Dennoch bleibt die manuelle Analyse der Log-Daten wegen der möglicherweise enormen Datenmenge aus den Log-Daten sehr zeit- und kostenintensiv, sofern dies überhaupt manuell möglich ist. Durch automatisierte Abstraktion der Log-Daten auf ein höheres Niveau kann dies vereinfacht und somit auch günstiger gestaltet werden. Dies wird, wie in verwandten Forschungsarbeiten [18], [25], [27], [8] und [24] beschrieben, bei einfacheren Nutzerschnittstellen (z.B. Kommandozeilenbasiert) durch Muster-Erkennung in den Befehlsketten oder bei komplexeren Programmen durch speziell an diese angepasste Log-Erstellung und Muster-Erkennungen mit vorgegebenen Aufgaben gelöst.

Eine Arbeit von 2006 [3] befasst sich mit den Details der Zuverlässigkeit und Sicherheit von implementierten Aspekten mittels AOP bei nebenläufigen Programmen. Eine ebenso aus demselben Sammelwerk stammende Arbeit [10] befasst sich zwar mit einem zu dieser Arbeit sehr ähnlichen Thema, jedoch ohne die Anwendung auf ein grafikbasiertes Programm, sondern wie oben genannte Werke auf ein kommandozeilenbasiertes Programm. Zudem wird lediglich die Mitschrift der Aktionen automatisiert und dynamisch implementiert. Hingegen die Analyse der Daten bleibt eine manuelle Aufgabe für Spezialisten.

Frau Vasundhara beschäftigt sich in [28] vom Dezember 2013 mit der allgemeinen Wiederverwendbarkeit von Aspekten und der effiziente Verbesserung von Software durch den Einsatz von Aspektorientierung. Somit bringt sie die Konzepte der Objektorientierung dem Kapseln von Eigenschaften zu einer Einheit und der Aspektorientierung dem Trennen von gemeinsamen Eigenschaften in eigenständige Elemente in Einklang, um eine effiziente Softwareentwicklung zu ermöglichen.

Obwohl das Konzept der Aspektorientierung schon seit mehr als 10 Jahren existiert, beschreibt Esubalew Alemneh 2014 in [1] die Technologie als „neue “ Technologie in den „Kinderschuhen“. Im Gegensatz zu der Objektorientierung hat sich die Aspektorientierung bisher nicht weit in der Industrie durchgesetzt und zeigt sich daher noch als neu beziehungsweise wenig bekannt.

In allen Arbeiten bezüglich der Aspektorientierung zeigt sich die Erkenntnis von Esubalew Alemneh, so dass diese hauptsächlich Grundlagen oder einfache Anwendungen der Technologie betrachten und nicht wie es bei Objektorientierung der Fall ist diese als simples Hilfsmittel betrachten. Es wird ähnlich wie bei der OOP noch Zeit benötigen, sodass ein Umdenken möglich ist und der Bedarf dieser Technologie vollständig geweckt wird.

Der Artikel [4] behandelt eine Automatisierungsmöglichkeit von GUI-Tests, jedoch vollständig automatisiert über Bilderkennungs-Algorithmen. Diese sind vorwiegend dazu da um die Funktion der Software und die Erreichbarkeit von Funktionen zu belegen. Diese Art zu testen ist jedoch ungeeignet um Usability-Schwächen zu erkennen.

Eine frühe Arbeit von 1996 [22] befasst sich bereits mit der automatisierten Aufzeichnung von Benutzeraktionen und der Reproduktion dessen Aktionen um Benutzerstudien durchzuführen. Dabei wird insbesondere die Beeinträchtigung der Probanden durch die Beobachtung bei anderen Observationsmethoden erwähnt.

4 Konzeption

In diesem Kapitel wird ein Konzept erstellt, in wie weit sich eine automatisierte Erkennung von Usability Schwächen generalisieren lässt. Dazu wird im ersten Kapitel 4.1 das eigentliche Nutzerverhalten analysiert und dieses Verhalten auf die Usability bezogen. Im zweiten Kapitel 4.2 wird das Log-Verhalten analysiert, welche der Benutzerinteraktionen von Relevanz und wie diese abzubilden sind. Zu guter Letzt wird die Analyse der erhaltenen Log-Daten im dritten Kapitel 4.3 genauer betrachtet und somit ein Verfahren ausgewählt, welches aus den Log-Daten nutzbare Daten extrahiert.

4.1 Von Nutzerverhalten zu verwertbaren Daten

Zu der Benutzerinteraktion zählen sowohl Ein- und Ausgaben des Computers, dessen zeitliches Verhalten sowie auch jegliche Aktionen des Nutzers vor dem Computer, die von diesem nicht entgegengenommen werden oder werden kann.

Die Benutzereingaben bei Computerprogrammen beschränken sich weitestgehend auf Tastatur- und Maus-Eingaben und aktuell auch auf Touch-Eingaben, welche wiederum auf Maus und Tastatureingaben zurückführbar sind, da diese im Normalfall für die gleichen Aktionen nur als Alternative gelten und nicht gleichzeitig kombiniert werden. Somit kann eine Berührung einer Schaltfläche mit einem Klick, eine „Wisch“-Geste also dem waagerechten oder senkrechten Streichen über das Display mit Scrollen und die Vergrößerungsgeste, dem berühren mit zwei sich auf einander oder voneinander wegbewegenden Fingern, mit einem aktivieren eines Bereiches gefolgt von einem Vergrößerungs-Shortcut gleichgesetzt werden. Da jedoch die Touch-Interaktion in der Anwendung RCE nicht im Fokus liegt, wird diese auch für die weitere Arbeit außen vor gelassen.

Als Benutzerausgaben stehen üblicherweise optische, wie auch akustische Signale zur Verfügung. Weitere Sinneseindrücke sind zwar technisch möglich, doch weder die Regel noch finden sie in RCE Anwendung. In der Arbeit werden nur die Hauptausgaben über den Monitor als optisches Medium betrachtet, da in dieser Anwendung akustische Signale vom System selbst kommen und nur der Verdeutlichung von optischen Signalen dienen wie zum Beispiel bei Warnmeldungen.

Für die Analyse des zeitlichen Verhaltens kann jeder Aktion ein Zeitstempel zugeordnet werden, sodass die Aktionen in eine Reihenfolge gebracht und ebenso die Dauer, wie auch Pausen erfasst werden können. Um eine Reihenfolge zu gewährleisten müssen Start-, End- oder mittleren Zeiten von Aktionen priorisiert

werden, da sich überschneidende Aktionen schwierig auszuwerten sind. In Abbildung 8 wird die Eingabe des Wortes „Hello“ dargestellt, während der Fokus von einer Warnmeldung in Anspruch genommen wird. Auf Grund der gegebenen nebenläufig agierenden Systemen und multiplen Ein- und Ausgabemöglichkeiten ist die Serialisierung nicht trivial im Vergleich zu früher üblichen Konsolenprogrammen. Bezüglich der Wertung von Aktionsdauern wird es komplex die Nutzerwirkung mit dem Verhalten zu verbinden, da jeder Nutzer auf ein Zeitverhalten hin sehr unterschiedlich reagieren kann. Manch ein Nutzer erwartet eine Rückmeldung oder Reaktion des Systems innerhalb von Millisekunden, bei manchen ändert sich dieses Verhalten über die Zeit, manche sind dauerhaft sehr geduldig und andere Nutzer wiederum stört die zu schnelle Reaktion des Systems die eigene Ruhe.

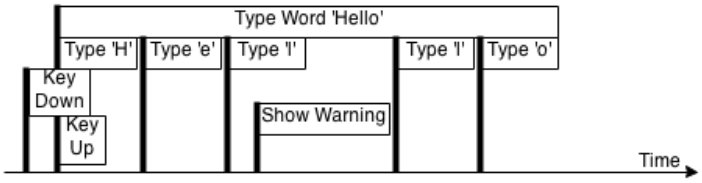

Abbildung 8: Eine Aktionsfolge bei einer nebenläufigen GUI

Bei dieser Varianz an Nutzern gibt es verschiedene Möglichkeiten der Analyse solcher Daten. Es kann zusätzlich der Gemütszustand des Probanden per Video, bestimmten Interaktionspattern (mehrfaches anklicken von Knöpfen) oder der Interaktionsgeschwindigkeit beobachtet werden, weiter gibt es auch Möglichkeiten den Probanden durch medizinische Beobachtung genauer einzuschätzen. Auf Grund des Ziels einer Beobachtung von Nutzern in der Anwendungspraxis, bleibt lediglich die Beobachtung von Interaktionspattern und der Interaktionsgeschwindigkeit. Letztere sind als absolute Werte wie oben beschrieben nicht tauglich, jedoch kann eine steigende Geschwindigkeit auf verschiedene noch genauer zu untersuchende Ereignisse hindeuten.

Bei der Usability Methode Think-Aloud werden insbesondere die sonstigen Aktionen, welche nicht als Eingaben dem Computer zur Verfügung stehen, genauer betrachtet, um auf Usability Schwächen zu schließen. Zu der computergestützten visuellen Erfassung von Benutzeraktionen, der Mimik, Gesten und Gemütern gibt es auch bereits Forschungsansätze wie z.B. [20]. Auf dies soll hier zur Reduzierung des Umfangs und des Aufwands höchstens über Interaktionspattern geschlossen werden.

4.2 Log-Verhalten bei Benutzerinteraktion

Das hier betrachtete Verhalten beschränkt sich, wie oben beschrieben, auf die Computer Ein- und Ausgaben sowie dessen zeitliche Abfolge.

Allgemein wird die Mitschrift von Log-Daten angewendet, um Interaktionen und Zustände reproduzieren zu können. Dazu würde es genügen die Maus- und Tastatureingaben sowie den aktuellen Daten des Computers abzubilden. Da ein Computer ein deterministischer, endlicher Automat ist genügen die Eingaben, wobei der Ausgangszustand durch die Daten des Computers und die Eingaben die Zustandsübergänge zu eindeutigen Zwischen- und Endzuständen zu führen. Einerseits bedeutet dies eine sehr große Menge an sensiblen Daten des Nutzers, welche gespeichert werden müsste, und andererseits auch eine große, unübersichtliche Menge an Daten zur Analyse. Daher wird das Log-Verhalten auf bereits leicht abstrahierte Interaktionen beschränkt. Damit muss bei einer Benutzereingabe nicht mehr beachtet werden, welchen Weg der Mauszeiger zu dem Interaktionselement genommen hat, sondern lediglich dass ein spezielles Interaktionselement zu einem gewissen Zeitpunkt per Maus genutzt wurde.

Auch bei den Interaktionselementen kann und sollte auf nur wenige Grundelemente abstrahiert werden, da diese im Zusammenschluss wenig aussagekräftig sind. Das mehrfache Interagieren mit einem Formular sagt nicht aus, ob der Nutzer den gewollten Weg der Eingabe genutzt hat oder mit dieser nicht zurechtkam. Zudem muss bei Grundelementen das Log-Verhalten nur einmalig implementiert werden, sodass eine Wiederverwendbarkeit und Fehlerreduktion möglich ist.

Zu den Grundelementen bei grafischen Oberflächen zählen Knöpfe, Eingabefelder, Label, Bilder, Elementgruppen und Fenster. Werden nun dessen Interaktionen mit diesen Elementen als Daten mit einem passenden Zeitstempel mitgeschrieben, so lässt sich aus diesen deutlich geringeren Datenmengen ebenso das Nutzerverhalten ableiten und es müssen nur geringe Datenmengen und sensible Benutzerdaten analysiert werden, welche zudem noch durch Zufallstexte desensibilisiert und anonymisiert werden können.

Die Rückgaben müssen nicht zwangsweise mitgeschrieben werden, da diese durch Wiederholen der Aktionen reproduziert werden können. Dennoch bietet es sich an dieser Stelle an während der Log-Erzeugung auch Rückgaben mitzuschreiben, sodass diese nicht erneut erzeugt werden müssen. Zusätzlich bietet dies eine Redundanz, um zuvor als unwichtig deklarierte, nicht mitgeschriebene Daten reproduzieren zu können oder auch eine Plausibilitätsprüfungen der Daten durchführen zu können. Gerade bei bereits abstrahierten, gefilterten und somit nicht vollständigen Daten sind Redundanzen sinnvoll.

Die erhaltenen Aktions-Daten sind zwar bereits zu Log-Daten abstrahiert und gefiltert, doch potenziell ist es auch möglich diese schon beim Client/Probanden zu analysieren oder daran weitere Voranalysen zu treffen. Damit werden die Daten noch abstrakter, jedoch möglicherweise nicht mehr reproduzierbar. Bei der Mitschrift aller Aktionen neben den Mausbewegungen sind die Datenmengen in RCE ausreichend gering, sodass die Zusatzlast des Client-Rechners nicht nötig ist. Bei den Mausbewegungen hingegen fallen so viele Daten an, dass eine Voranalyse auf Bewegungs-Pattern oder Gesten sinnvoll erscheint. Dennoch werden zu Gunsten des Umfangs lediglich die Mitschrift von Mausbewegungen in RCE vorbereitet und nicht weiter implementiert. Als recht gut und schnell zu implementierende Voranalyse sollen Doppelklicks, Texteingaben als zusammenhängend und Größe- beziehungsweise Positionsänderungen des Fensters als kompakte Ereignisse erfasst werden.

Zusätzlich zu der Abstraktion soll auch per Kompression und gegebenenfalls Verschlüsselung den Problemen der großen Datenmengen und dem Offenlegen sensibler Daten vorgebeugt werden. Gerade bei für Menschen lesbaren Log-Daten ist die Verringerung der Datenmengen sehr effizient umsetzbar und sinnvoll.

4.3 Datenanalyse aus den Log-Daten

Die Log-Daten enthalten nach den oben beschriebenen Abstraktionen Eingaben, Interaktionen und Rückgaben aus der Anwendung. Diese können mittels verschiedener Pattern auf die Usability hin analysiert werden. Da bei der Usability zwischen Ease of Learning (intuitiver Handhabbarkeit) und Ease of Use (effizienter Nutzbarkeit) unterschieden wird und sich diese Ziele grundsätzlich unterscheiden sollte es einen Fokus geben. In der speziellen Anwendung RCE sollte für die grundlegende Nutzung und somit zu einem vereinfachten Einstieg in die Software *Ease of Learning* und für fortgeschrittene Aufgaben zur Verhinderung von ermüdenden und umständlichen Aktionsfolgen *Ease of Use* angestrebt werden. Für diese Arbeit liegt der Fokus dabei den Einstieg für neue Nutzer zu betrachten und somit bei *Ease of Learning* in den weniger fortgeschrittenen Nutzerinteraktionen.

Es lässt sich über eine wiederholte, gleiche Abfolge von Nutzeraktionen ohne erkennbare Wirkung ein sich nur schlecht selbst erklärendes System ableiten und somit das Suchverhalten des Nutzers beschreiben. Unscharf bleibt hierbei jedoch, wann eine Wiederholung zu erkennen ist. Eine Nutzeraktion entspricht nicht immer exakt einem Log-Eintrag, da neben Aktionen des Nutzers auch Rückgaben geloggt werden und die Einschätzung einer Aktion sehr situations-

abhängig ist. Wird ein Klick auf dasselbe Objekt, ohne Zustandswechsel, direkt wiederholt ohne ein verändertes Ergebnis, so scheint dies bereits ein Usability Problem zu sein. Wird in einem Tutorial oder einer geführten Eingabe häufiger der „Weiter “-Knopf verwendet, so ist dies noch kein Problem. Es ist auch nicht festlegbar, was genau eine Nutzeraktion abgrenzt. Es könnte das Einfügen eines konfigurierten Elements in den Arbeitsbereich sein oder jeder Konfigurationsschritt einzeln. Genauso ließe sich auch das öffnen verschiedener Menüs ohne Aktion als ein Suchverhalten nach einer spezifischen Funktion und somit einem Usability Problem identifizieren. Jedoch können die Aktionen wiederum durch andere Aktionen unterbrochen werden, mal per Tastatur und mal per Maus und somit beliebig komplex werden.

Auf Basis dieses beschriebenen Problems muss eine passende Filter- und Abstraktionsmethode gewählt werden. Doch auch mit den besten Filtern und Abstraktionen wird nie die richtige oder beste Methode genutzt, sondern nur eine für diese Anwendung tendenziell gute Methode.

Das „Maximum Repeating Pattern MRP“ von [25], welches eine Wiederholung von Aktionen maximaler Länge erkennt ist bezüglich der Komplexität von wiederholten Aktionen bereits robuster als eine einfache Wiederholungserkennung. Es deutet auf eine schlechte Usability des Interfaces hin, bei dem ein häufig verwendetes Verhalten zudem noch besonders aufwändig ist. Möglicherweise genügen bei dem Problem bereits einige kurze Hilfetexte oder aber es sollte die nötige Befehlsfolge verkürzt beziehungsweise überarbeitet werden.

Der Intuition nach sollte nicht bloß ein Verhaltensmuster, welches sich wiederholt und die maximale Länge aufweist, auf ein Problem der Usability hindeuten, sondern auch die häufigste Wiederholung eines Musters kurzer Länge. Dabei darf das Muster nur so kurz sein, dass ein Verhaltensmuster erkennbar bleibt. Das bedeutet es muss mindestens so lang sein, wie eine in der zu prüfenden Interaktion üblichen Befehlskette. Solch eine Auffälligkeit deutet möglicherweise auf eine umständliche, ermüdende Vorgehensweise in einer Aufgabe hin. Das kann einerseits an der Aufgabe liegen, welche für das Programm ungünstig gewählt ist, an einer umständlichen Handhabe oder auch an einem schwierig auffindbaren einfachen Lösungsweg.

Als ein weiteres Pattern lässt sich prüfen, ob alle Benutzeroberflächenelemente deckend verwendet werden und somit zumindest einmalig Verwendung finden. Da jedoch viele Interaktionselemente erst im Fehler- oder Ausnahmefall zu betätigen sind, ist dieses Pattern nur selten ohne eine hinreichend großen Probandenzahl und eine passende Vorhersage der Elementnutzungen erfolgversprechend. Dies in Verbindung mit einer Nutzungsstatistik aller Elemente der Benutzero-

berfläche kann für eine Erweiterung viel versprechende Ergebnisse liefern und die weitere Verbesserung dienlich sein.

Interaktionselemente können zu guter Letzt über ein Punktesystem gewertet werden, welches beispielsweise einem „Abbrechen“-Knopf mehr Fehlerpotenzial zuordnet als einem „OK“-Knopf, da ersterer auf einen Benutzerfehler deutet, welcher durch ein Usability Problem ausgelöst werden konnte. Einem „Hilfe“-Knopf kann ein mittleres Maß an Überarbeitungspotential zugeordnet werden, da bei dessen Betätigung der Nutzer scheinbar Probleme mit der Bedienung hat. Er findet beispielsweise kein ausreichend aussagekräftiges Interaktionselement oder hat sonstige Verständnisprobleme.

Die Software RCE wird im Folgenden um die Möglichkeit einer Mitschrift von Benutzeroberflächen-Interaktionen mittels aspektorientierter Programmierung erweitert und eine weitere Software erstellt, um die daraus entstandenen Daten nach dem benannten MRP Verfahren und einer einfachen Erkennung von Wiederholungen fester Länge zu analysieren.

5 Implementierung

Dieses Kapitel befasst sich mit der Implementierung der in der Konzeption ausgearbeiteten Inhalte.

5.1 AspectJ Integration

RCE als RCP-basierter Anwendung besteht aus einem Kern und vielen nicht optionalen Bundles zur Erweiterung des Funktionsumfangs und Kapselung von Funktionen und Daten. Aus diesem Grund werden die neuen Implementierungen ebenso als Bundle deklariert. Lediglich die Schaltfläche zum Aktivieren der Log-Erzeugung auf der Werkzeugleiste von RCE wird in der XML-Datei des Haupt-Bundles ergänzt. Diese enthält eine Referenz auf die Aktivierungsfunktion der Log-Erzeugung. Von dieser Funktion lässt sich allerdings nicht das Einweben abhängig machen, sondern das Aufzeichnen der gesammelten Ereignisse.

Bei der Integration von Aspekten über ein Bundle muss dieses in Eclipse zu einem AspectJ-Projekt umgewandelt werden, sodass Eclipse den AspectJ-Compiler anstelle des Java-Compilers verwendet. Bei der Verwendung von bundleübergreifenden Aspekten und Load-Time Weaving (LTW) in einem Projekt mit OSGi als Softwareplattform und Equinox als Framework müssen zusätzlich noch vier Bundles OSGi bekannt gemacht werden. Zu diesen Bundles gehören zuerst die Aspectj Runtime, der Aspectj Weaver, der Equinox Weaving Hook und das Equinox Weaving Aspectj. Dabei muss Letzterer bereits gestartet sein bevor die Aspekte einzuweben sind und bevor die betroffenen Bundles gestartet wurden. Mit dem folgenden Argument für die Java VM

−Dosgi.framework.extensions=org.eclipse.equinox.weaving.hook

und der Ergänzung der genannten Bundles zu einem gemeinsames Target sind alle Schritte für Aspekte innerhalb eines Bundles erledigt.

Für die bundleübergreifenden Aspekte gibt es zwei Möglichkeiten. Zum einen kann jedes Bundle die Aspekte für sich aktivieren oder das Aspektbundle wird global aktiviert. Bei der zweiten, hier gewählten Möglichkeit kann zusätzlich jedes Bundle individuell das Einweben von Aspekten für seine Klassen und Funktionen verbieten. Die Wahl der zweiten Möglichkeit ist sinnvoll, da die Bundles mit den anzupassenden GUI-Elementen zahlreich sind, als Bibliotheken nicht immer angepasst werden können und so alle neu erstellten Bundles auch berücksichtigt werden.

Die Umsetzung dieser Lösung bedarf eines Exports der Aspekte, der Ergänzung der Datei MANIFEST.MF um die Zeile

Eclipse-SupplementBundle:

gefolgt von den betroffenen Bundles, welche auch Wildcards enthalten können, sowie einer Datei aop.xml mit dem folgenden Inhalt (siehe Listing 1).

Listing 1: aop.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<aspectj>
  <weaver options="-Xset:weaveJavaxPackages=true">
  </weaver>
    <aspects>
        <aspect name="packet.aspectname"/>
    </aspects>
</aspectj>
```

In dieser Datei werden in dem Tag <aspects> alle Aspekte als jeweils einzelner Tag aufgeführt. Der Tag <weaver> ist optional und enthält hier eine Option, welche es erlaubt auch in Java eigene Bundles (org.java.*) Aspekte einzuweben.

Nach dieser Vorbereitung und dem Erstellen eines ersten Aspekts sind erste Warnungen des Weavers zur Laufzeit möglich (siehe Listing 2). Diese zeigen an, dass der Weaver an eine Stelle gekommen ist, welche für ihn nicht einsehbar ist und somit potenzielle *Join Points* ignoriert werden. Diese Warnungen lassen sich beheben, wenn die fehlenden Klassen in der MANIFEST.MF als Import ergänzt werden.

Listing 2: Weaver Warnung

```
[[..]] warning can't find type [..] whilst determining
    signatures of call or execution join point for void [..], 
    this may cause a pointcut to fail to match at this join 
    point
when weaving type [..]
when weaving classes
when weaving
[Xlint:cantFindTypeAffectingJPMatch]
```

5.2 Log-Erzeugung

Bei dem Erstellen von Log-Daten gibt es verschiedene Herangehensweisen. In der gängigeren Methode werden die Elemente und Programmstellen mit einem

Aufruf an eine externe Log-Funktion ergänzt, da diese Ergänzungen nur stetig mehr werden und so gezielt eingesetzt werden können. Damit entstehen nur die gewünschten Daten und Log-Einträge. Dennoch ist diese Methode nicht besonders flexibel, da sie in jedem neu erstellten Programmteil erneut integriert werden muss. Für die aspektorientierte Programmierung ist dieser Ansatz sogar sehr aufwändig umzusetzen, da die Pointcuts eindeutig nur auf eine Position zutreffen dürfen. Dementsprechend viele Pointcuts müssen ertstellt werden.

Eine andere Methode ist es ausschließlich bereits Log erzeugende Elemente zu verwenden und im späteren Verlauf aus diesen vielen Daten die richtigen Daten zu filtern. Dabei muss zwar bei der Implementierung auf die Verwendung entsprechender Elemente geachtet werden und möglicherweise müssen diese auch zuvor erstellt werden, doch diese können danach immer wieder verwendet werden. Für diesen Ansatz ist die Aspektorientierung besonders eignet, da wenige Pointcuts allgemein gehalten auf viele Join Points zutreffen dürfen. Dabei müssen keine Log erzeugende Elemente verwendet werden, sondern damit werden die Elemente zu Log erzeugende Elementen angepasst. Der Aspekt *log-erzeugend* muss somit auf alle betroffenen Elemente zutreffend definiert sein.

Mit der Möglichkeit Funktionen, Klassen, Rück- und Übergaben per Wildcards anzusprechen kann der Aufwand der Einflechtungen gering gehalten werden. Alle betroffenen Elemente sind GUI-Elemente und diese haben eine große Gemeinsamkeit. Üblicherweise werden bei der Benutzung der Programmoberfläche Events (Objekte von Aktionen) geworfen und diese auch von einem Event-Listener bearbeitet. Dies bedeutet es werden Funktionen von Objekten aufgerufen, welche speziell für die Behandlung der aufgetretenen Aktionen erstellt wurden. Solche Funktionen besitzen in Java SWT üblicherweise einen Datentyp, welcher von EventObject erbt.

Es besteht einmal die Möglichkeit alle grafischen Objekte, welche Events werfen um einen Event-Listener zu erweitern und diese dann zu verwenden. Eine andere Möglichkeit greift dort ein wo bereits ein Listener zugewiesen wurde, dieser Events erhält und diese von dem Aspect weiter verwendet werden können. Bei der ersten Lösung werden mehr Klassen betroffen sein als bei der zweiten Lösung, wobei die Analyse bereits statisch im Vorfeld getätigt werden kann und damit dauerhaft weniger Aufwand bedeutet. Bei der zweiten Lösung werden erst zur Laufzeit Events abgefangen, was zu einer höheren Systemlast führt. Dennoch werden bei der zweiten Lösung nur die auch sonst beachteten also wichtigen Events behandelt. Dabei werden zusätzlich nur wenige gleichwertige Events abgefangen, sodass die Filter weniger umfangreich sein müssen.

Anhand dieser Argumente wird die zweite Lösung gewählt und ein Pointcut für die Ausführung von Funktionen mit der Übergabe eines EventObjects oder einer

davon erbenden Klasse erstellt (siehe Listing 3 Zeile 1). Der Aufruf noAspect() ist die Integration eines weiteren Pointcuts in diesen, welcher alle Klassen und Funktionen aus dem neuen Bundle ausschließt.

Listing 3: Pointcut eine Funktion mit EventObject als Übergabe

```
1 pointcut exEventObject() : execution(void *(EventObject+)) &&
      noAspect();
2
3 before(Object aspectTarget, EventObject eve) : exEventObject()
      && target(aspectTarget) && args(eve){
4   [..]
5 }
```

Nach dem Pointcut folgt der Advice (siehe Listing 3 Zeile 3ff). Dieser verwendet den Pointcut und ergänzt ihn in den darauf folgenden Aufrufen um Informationen aus diesem Pointcut. An erster Stelle wird der behandelnden Event-Listener und an zweiter Stelle dessen behandeltes Objekt als Übergabe für das Advice zur Verfügung gestellt. Zur Verwendung dieser Werte wurden in den Parametern des Advice die beiden Variablen mit dem passenden Typ angelegt. Anstelle des Auslassungszeichen folgen die genaueren Verzweigungen welches Objekt wie verarbeitet wird. Dazu lassen sich die Objekte der allgemeinen Klasse EventObject noch genauer über dessen weniger allgemeine Klasse, den Inhalten und zugänglichen Stati bestimmen und für die Logs aufbereiten.

Da hier mehrere Stellen des Programms parallel angesprochen werden, können die einzuflechtenden Inhalte nur eine gemeinsame Schnittstelle der anzusprechenden Programmstellen nutzen, welche die Aktionen, Zustände und Rückgaben der Nutzer-Interaktion enthalten muss. Um weiterhin das Prinzip Separation of Concerns zu gewährleisten werden lediglich kurze Inhalte mit Verweisen auf Funktionen aufbauend auf einem passenden ListenerInterface eingewoben.

All diese Listenerfunktionen sind in einer einzigen Klasse wie in Abbildung 9 dargestellt vereint, da viele gemeinsame Funktionen und sehr ähnliche Prüfungen nötig sind. Zu diesen Funktionen und Prüfungen gehören unter anderem die Folgenden:

- Der Vergleich von Events auf Gleichheit ohne eine zeitliche Betrachtung. Viele Events wurden vielfach ausgelöst und behandelt, wobei oft der einzige Unterschied in dem Zeitstempel liegt. Da die Log-Daten in etwa jeweils einer Aktion des Nutzers entsprechen soll sind diese Wiederholungen unbrauchbar und müssen diese gefiltert werden.
- Zusammenführen von Events; Erst das Auslösen mehrerer Events entsprach öfters nur einer Nutzeraktion. (z.B. KeyDown -> KeyUp = Type)

- Starten und Stoppen der Auswertung bzw. der Speicherung von Aktionen bedarf einer zentralen Schnittstelle.
- Extrahieren sinnvoller Inhalte aus den Events und deren Quellen (Kontext).
- Werten und ggf. verwerfen von Events oder einem Teil dieser, wegen Irrelevanz (Mehr Klassen als die der Oberfläche werfen Events und haben keinen Bezug zu den Nutzeraktionen).
- Alle erzeugten Log-Einträge müssen über eine gemeinsame Schnittstelle zu einem Log-Objekt zusammengeführt und gesammelt werden.

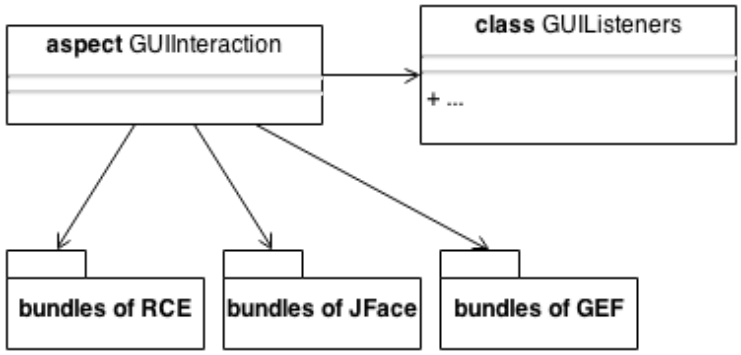

Abbildung 9: Die Verbindung der Bundles mit dem gemeinsamen Listener

Anfänglich wurden alleine die Klassen von RCE selbst um die Aspekte erweitert, dabei zeigte sich jedoch, dass viele Bibliotheken und Klassen bereits ihre eigenen Event-Listener implementieren und somit ihre Aktionen selbst behandeln. Aus diesem Grund wurden die Aspekte zuerst auf die Bibliotheken JFace und SWT erweitert, doch bei dem Übermaß und einigen vielfach auftretenden Events genügte bereits JFace zur Erkennung von Nutzer-Aktionen in den in RCE genutzten Assistenten und vorgefertigten Dialogen.

Weitere Funktionen bei Eclipse RCP werden aus XML-Dateien erst zur Kompilierzeit erstellt wie zum Beispiel die Menüs der Menüleiste und die Kontextmenüs. In der Theorie können diese über Wildcards identifiziert werden, doch in der Praxis müssen die Objekte bereits bei der ersten Ausführung der Aspekte existieren um mögliche Einstiegspunkte zu markieren.

Wegen dem fehlschlagen solcher aspektorientierten Ergänzungen wurde auf die Hierarchie von SWT zurückgegriffen. Bei der Erstellung von grafischen Oberflächen in SWT sowie dem darauf aufbauenden JFace entstehen zwangsläufig Hierarchien. Jedes Element abgesehen von einem Display und einer Shell hat

einen Parent und jeder Parent kann 0 bis n Children erhalten. Es lässt sich an jeder Stelle des Quellcodes das Display abfragen und somit ist es möglich von dem Display an dessen Shell und darüber zu allen Children zu gelangen.

Die genannte Menüleiste und die Kontextmenüs sind auch mit dieser Methode nicht zu finden. Wird zu einem gewissen Zeitpunkt der Baum an SWT-Elementen durchlaufen, so ist es möglich, dass die Menüs noch nicht existieren. Grundsätzlich existiert allerdings eine Grundstruktur, welche Events auslöst und somit über Aspekte auffindbar ist. Ab dem Auslösen des Events können dann dessen neue Children durchlaufen werden und mit einem Menü-Listener versehen werden.

Zusätzlich ließ sich die Open-Source Bibliotheken Grafical Editing Framework (GEF) nicht mittels der bisherigen Pointcuts ansprechen oder über die Hierarchie von SWT. GEF ist ein auf Polygonen und Pixeln basierte grafische Bibliothek für Zeichenflächen mit Werkzeugleisten, welche ihre eigene Hierarchie besitzt und nicht nur mittels geworfenen Events intern interagiert. Somit konnte eine Schnittstelle in RCE zu einem das gesuchte Objekt enthalteden Objekt über einen neuen Pointcut gefunden werden und dank Open-Source über die Hierarchie eine Art Listener ergänzt werden.

Anhand all dieser gesammelten Daten und dessen Auswahl ist das Verhalten des Nutzers deutlich zu erkennen und nachvollziehbar. Lediglich wenige Bereiche der GUI wurden identifiziert, in denen keine Logs erzeugt werden. Diese sind beispielsweise die selten vorkommenden Standard-Kontextmenüs zum Kopieren und Einfügen von Text. Auch Standardfenster, welche ohne Funktion oder auch für RCE irrelevante Elemente von Eclipse sind, wurden von dem Aspekt nicht behandelt.

Das Sammeln der Informationen ist also mit ausreichendem Umfang vollständig. Diese Daten müssen allerdings noch temporär aufbewart werden. Dazu wird eine einfache dynamische Liste verwendet, welche Objekte vom Typ LogEntry als Log-Einträge entgegennimmt. Diese Einträge haben drei Felder, eines für einen Typ der Log-Nachricht, einen für den eigentlichen Inhalt und einen für beliebige Informationen, welche später noch hilfreich sein könnten, aber vorerst nur gespeichert werden.

Für die Typisierung bietet sich der Datentyp Enum an, welcher durch sprechende Namen einfach zu nutzen und sehr effizient verarbeitbar ist. Als zweites Feld soll ein *String* genügen, welcher für eine vereinfachte Handhabung der Log-Elemente geeignet ist und als Informationsfeld dient der Datentyp *Object*, der jedes beliebige Objekt entgegen nehmen kann. Damit immer eine Reihenfolge und Zeitliche Abfolge erkennbar ist, wird jedem neu erzeugten Log-Element der

aktuelle Zeitstempel ergänzt. Dieses Feld ist nicht setzbar, um Tippfehlern und Formatsfehlern vorzubeugen. Falls ein Event erst später zu einem Log wird kann dieser auch sofort erzeugt werden und erst später in der Liste angehängt werden. Zu Debug-Zwecken wird eine Funktion zur Umwandlung des Objekts in Text erzeugt, sowohl unter als auch ohne Berücksichtigung des Informationsfeldes. Eine Vergleichsfunktion für zwei Log-Einträge ignoriert ebenso, das Informationsfeld und für die spätere Analyse ebenso das Feld des Zeitstempels.

Anhand dieser Klassen und deren Verhalten entstehen somit die in Abbildung 10 gezeigten Beziehungen und deren Eigenschaften.

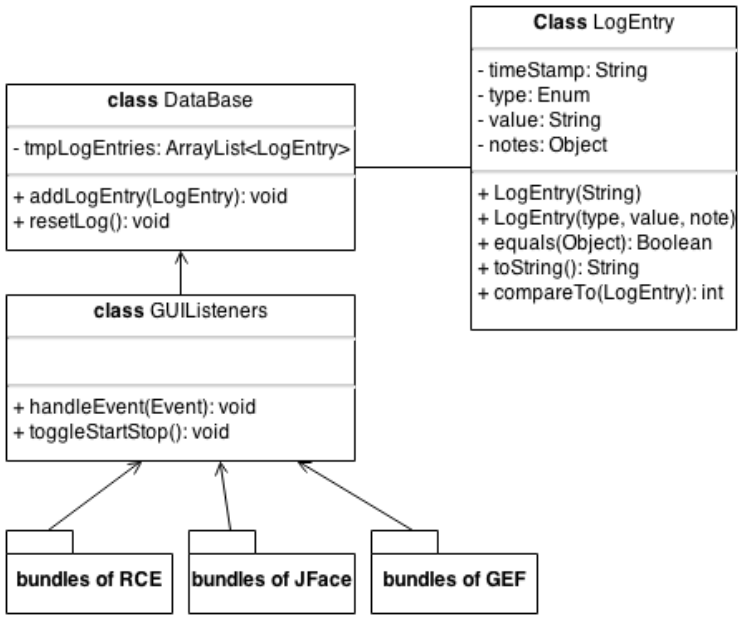

Abbildung 10: Klassenbeziehungen in RCE zur Log-Erzeugung

Um wie beschrieben den Log nur unter Kenntnis des Nutzers zu erzeugen, wird eine Schaltfläche in der Werkzeugleiste integriert, welche erst nach Aktivierung die Erzeugung der Daten startet und wieder beendet. Dabei werden alle nötigen Instanzen gebildet, Listen geleert und die vereinzelte Ergänzung um Listener vorbereitet. Wird das Loggen wieder ausgeschaltet, so werden die bis dahin zwischengespeicherten Daten wie im folgenden Kapitel beschrieben persistiert.

5.3 Persistenz

Die dynamischen Listen enthalten bisher alle gesammelten Daten, diese zu persistieren ist nun die Aufgaben. Eine besonders simple Variante ist es diese Liste

direkt binär zu schreiben. Dabei bleiben alle Informationen erhalten und diese können simpel wiederhergestellt werden, jedoch müssen dafür alle enthaltenen Klassen serialisierbar werden. Dies ist je nach Klasse nicht möglich. Für die Möglichkeit der Nutzer die Daten einzusehen und so selbst auf Sensibilität bewerten zu können bietet es sich dennoch an Textdokumente zu erzeugen. Grundsätzlich ist dies kein Problem, da der Typ, der Zeitstempel und der Inhalt leicht zu überführen sind, lediglich die zusätzlichen Informationen sind schwierig zu überführen. Da diese für die Auswertung nicht beachtet werden sollen, sondern nur zur Problemlösung eine Hilfestellung bieten können, werden diese Daten mittels Standardfunktion in Text umgewandelt. Dabei gehen zwar viele Informationen verloren doch Hinweise bleiben und es ist bezogen auf den Aufwand die beste Lösung.

Der Einfachheit halber wird eine CSV-Datei erstellt, welche über die Java eigene Zip-Funktion komprimiert wird. Da bei der CSV-Datei die üblichen Trennzeichen Komma und Semikolon, auch in den Texten vorkommen können, sollen anstelle dessen drei Unterstriche hintereinander als eindeutige Trennung der Spalten dienen.

Mit dieser Methode wird also zeilenweise ein Textdokument erstellt. Dazu muss noch beachtet werden, dass die einzelnen Textausgaben keine Absätze enthalten oder diese durch Platzhalter ersetzt werden. Zur Förderung der Handhabbarkeit in der Analysesoftware fehlt es der Datei noch an gewissen Prüfmöglichkeiten.

Für Weiterentwicklungen zu ermöglichen und eine Prüfung der Datei auf Log-Daten zu vereinfachen soll als erstes Log-Objekt ein Verifikationsstring „RCPLogAnalyser V#.#“ inklusive einer Versionsnummer am Ende und eine Log-ID der Log-Schnittstelle als eigenes Log-Objekt in der Liste gespeichert werden. Darüber kann einerseits geprüft werden, dass die Datei Log-Daten enthält, ob die Log-Daten Version zu der Analyse-Version passt und ob diese Daten bereits analysiert wurden. Eine genauere Prüfung auf Konsistenz, Integrität und Authentizität ist nicht vorgesehen, da bei einer Manipulation der Daten kein erkennbarer Nutzen des Manipulators vorstellbar ist. Für die Anonymität der Daten und des Nutzers hat die Log-ID keinen Bezug zu dem Nutzer selbst und dazu soll es genügen mit den Standardfunktionen von Java eine pseudozufällige, 32-stellige, hexadezimale Zahl zu erstellen und zu verwenden. Zudem ist eine automatische Übertragung der Daten noch nicht geplant und der Nutzer muss seine Daten selbst dem Usability-Tester übergeben.

Eine Verschlüsselung ist somit auch nicht nötig, wobei diese bereits mit den Mitteln von Java bis zu 128 Bit Schlüssellänge möglich ist und mit der Bibliothek Zip4J bis zu 256 Bit AES Verschlüsselung unterstützt wird (vgl. [19] und [14]). Diese Formate sind mit den meisten Komprimierungsprogrammen kompatibel.

Für eine reelle Absicherung gegen Fremdzugriffe sind diese Schlüssellängen nicht ausreichend und somit wird selbst eine Vorbereitung für die Verschlüsselung nicht erstellt.

Das Zwischenergebnis der Klassenbeziehungen und dessen Inhalte können der Abbildung 11 entnommen werden.

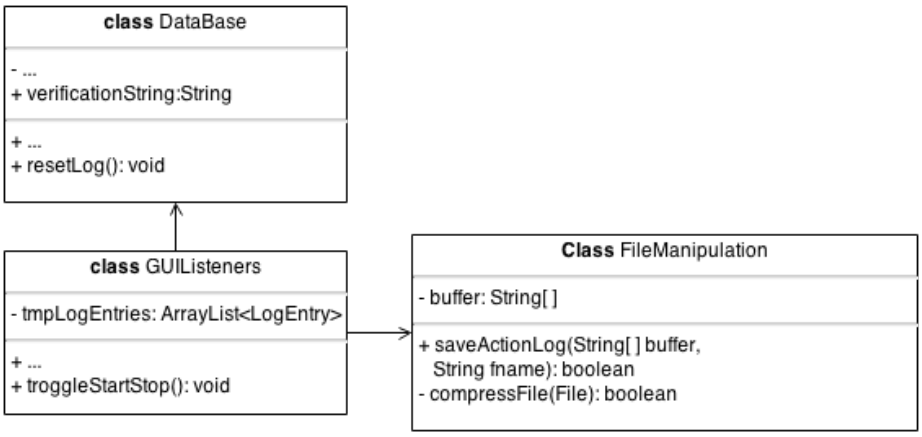

Abbildung 11: Klassenbeziehungen in RCE nach der Log-Erzeugung

Die komprimierten Daten, sowie auch unkomprimierte können nun mit dem in den folgenden Kapiteln entstehenden Analyse-Werkzeug geöffnet und analysiert werden. Dazu werden sie als Textelemente in den Log-Einträgen eingesetzt. Bei der Überführung wurde somit nur der Zusatzinformationsteil von einem beliebigen Objekt zu einem Text umgewandelt. Ansonsten bleiben alle Daten unverändert und können wie im folgenden Kapitel beschrieben analysiert werden.

5.4 Analyse der Log-Daten

Für die prototypische Entwicklung dient eine von Eclipse RCP 4.0 aus einer XML-Datei erstellten Oberfläche. Die Erstellung solch einer Datei erfolgt stark von der Entwicklungsumgebung geführt und bildet schnell eine funktionsfähige, grafische, eigenständige Anwendung mit einer Menüleiste, Symbolleiste und einem Hauptfenster. Das Fenster wird um eine textuelle Ausgabe innerhalb eines scrollbaren Bereiches erweitert, welche später dynamisch befüllt und wieder geleert werden kann. Menüpunkten und den einzelnen Werkzeuge der Symbolleiste zu erstellen und Funktionen zuzuweisen geschieht auch über die XML Datei und wird per Maus geführt (siehe Abbildung 12).

Da die eingelesenen Daten der Analysesoftware mit minimalen Abweichungen den Originalen entsprechen und diese möglicherweise nicht in der richtigen Rei-

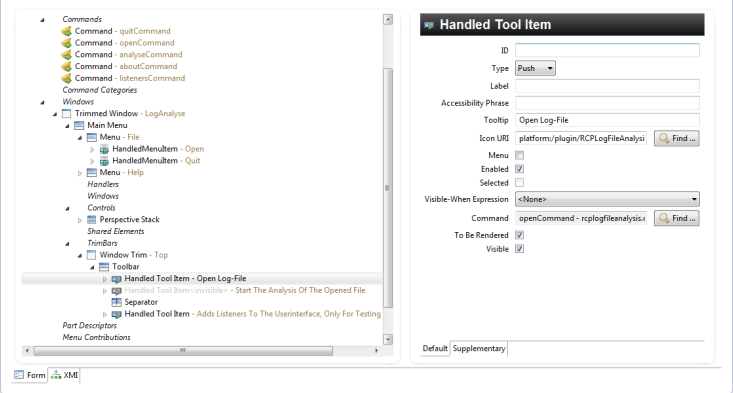

Abbildung 12: Eine e4xmi Datei (XML) im zugehörigen Editor von Eclipse

henfolge sind, müssen vor der Analyse diese sortiert werden. Dazu wird ein Vergleichsoperator für die Log-Einträge compareTo() erstellt und so kann die Liste bereits mit Java-Mitteln sortiert werden.

Als Möglichkeit die Auswertung der Daten noch zu beeinflussen können an dieser Stelle noch Log-Einträge gefiltert, zu mehreren aufgesplittet oder auch zusammengeführt werden, sodass aus der Aufzeichnung noch vorhandene Wiederholungen nicht direkt als Probleme hervorgehoben werden.

Diese sortierte Liste enthält noch viele Objekte, die für die schnelle Verarbeitung also vielfachem lesen, schreiben und kopieren zu umfangreich sind. Daher werden die Objekte zu positiven Ganzzahlen konvertiert. Dabei wird zuerst eine Legende erstellt, welche in der Liste von vorne begonnen jedem noch nicht verhandenen, gleichwertigen Legenden-Eintrag eine Zahl zuordnet. Mit dieser Legende wird dann eine abstrahierte Liste für die Verarbeitung erzeugt (siehe Abbildung 13).

Nach dem Aufruf der Startfunktion über die Symbolleiste werden die Algorithmen zur Erkennung von Pattern gestartet, dessen zusätzliche Eingaben abgefragt und dessen Ergebnisse in dem Fenster dargestellt.

Für die Arbeit wurden zwei Pattern ausgewählt, zum einen ein Algorithmus zur Erkennung des bereits beschriebenen Maximum Repetition Pattern (MRP) und zur Erkennung von Wiederholungen vorgegebener Länge.

Für das MRP hat bereits Siochi in [25] einen Algorithmus beschrieben, welcher effizient große Datenmengen verarbeiten kann. Dabei werden wiederholte Eingabefolgen zurückgegeben, welche sich auch überschneiden können. Dies ist für die Anwendung sinnvoll, da der reelle Zusammenhang von Befehlen zu Aktionen in dieser Analyse nicht erkennbar ist. Dennoch kann die Wiederholung zeigen, dass ein potenzieller Zusammenhang existiert.

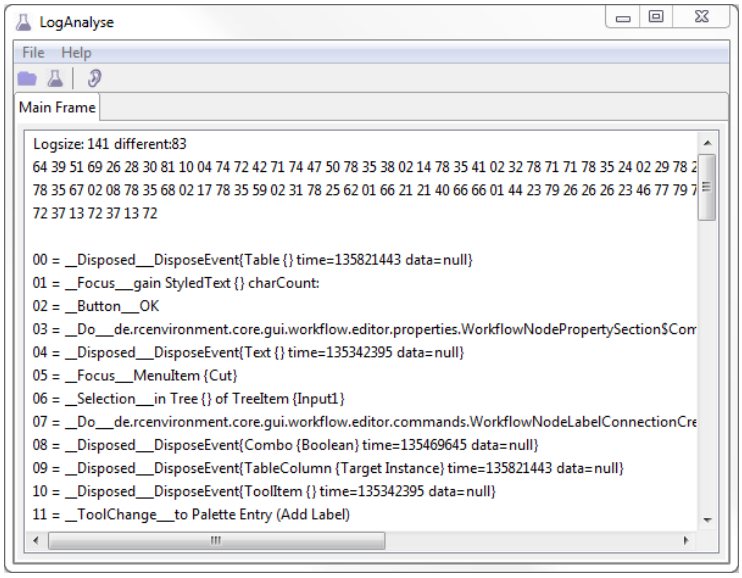

Abbildung 13: Die Analysesoftware aus der genannten e4xmi Datei

$$cbababababcba$$

Die hier dargestellte Zeichenkette soll abstrakt drei verschiedene Log-Einträge in einer Abfolge darstellen, diese Abfolge weist die folgenden verschiedenen Wiederholungen auf.

$$\underbrace{cbababababcba}_{\underbrace{\underbrace{\quad}_{}\quad}_{}} \quad \underbrace{cbababababcba}_{\underbrace{\quad}_{}} \quad \underbrace{cbababababcba}_{}$$

Von Links nach Rechts alle Wiederholungen der Länge 2, der Länge 3 und der Länge 4. Bei der letzten Wiederholung sind vorige Wiederholungen mit umschlossen, womit diese mehrfach als potenzielles Problem auftreten und so einen Zusammenhang besitzen oder auch ein Usability-Problem sein können.

Bei dem MRP wird ein Suffix Baum (engl. Position Tree) erzeugt, welcher in nur $O(n^2)$ vollständig erzeugt werden kann und eben soviel Speicher benötigt. Ohne diese spezielle Datenstruktur würde in einem Brute-Force Ansatz für die Prüfung auf Wiederholungen beliebiger Länge $O(n^3)$ benötigt. Bei der Erzeugung des Baums werden gleichzeitig die Ergebnisse erzeugt und dessen am tiefsten liegender Knoten gespeichert, sodass der Baum nicht zusätzlich nach seinen längsten Ästen durchsucht werden muss. Das Ergebnis wird ab den gespeicherten Knoten durch den eindeutigen Weg zur Wurzel beschrieben.

Ein Suffix Baum besteht aus Knoten ohne Inhalt, Kanten mit den Eingabezeichen und den Startpositionen der Zeichenketten in den Blättern. Für die Erzeugung des hier auf Zahlenketten angepassten Suffix Baumes wird die Eingabekette um ein Endzeichen erweitert und dann von Anfang bis Ende durchlaufen. Bei jedem der Eingabeelemente wird eine Position in dem Baum gesucht und dort eine Verzweigung, am Blatt die Startposition aus der Eingabekette und der Rest als eine Position der Eingabe gespeichert. Dadurch müssen einzigartige Eingabekettenteile, welche hier nicht von Interesse sind, nicht gespeichert werden. Da die Ketten nicht jedesmal vollständig gespeichert werden, muss an jedem erreichen Blatt dieses mit dem nächsten Element seiner Kette eine Ebene tiefer geschoben werden. Zusätzlich wird die Tiefe des Baumes gespeichert, sodass direkt erkannt werden kann wie tief der tiefste Knoten ist. Wird bei dem Einfügen einer Kette die tiefste Stelle überschritten, so wird der neue Wert gespeichert und der Knoten als tiefster Knoten ebenso. Wird hingegen eine genauso tiefe Stelle erreicht wird der Knoten zu den tiefsten Knoten hinzugefügt. Damit muss der Knoten für die Ausgabe des Ergebnisses nicht erneut gesucht werden. Die tiefste Stelle ist also immer ein Blatt, wobei dessen Vorgängerknoten eine Verzweigung ist und somit eine längste wiederholte Kette darstellt. Zur Ausgabe dieser Ketten müssen nur noch die eindeutigen Wege von den tiefsten Knoten zu der Wurzel ausgegeben werden.

Das zweite Muster lässt sich zwar mit nur wenigen Zeile eigenständig programmieren, doch da der Baum Bereits existent ist, wird dieser zur Hilfe genommen und es muss in diesem lediglich in der Ebene, welche der Länge der zu suchenden Wiederholung entspricht, ein passender Knoten gefunden werden. In dieser Ebene sucht der Algorithmus den Knoten mit den meisten nachfolgenden Blättern. Diese Daten zu notieren wurde dem obigen Algorithmus ergänzt. Das bedeutet während der Positionssuche für die einzufügenden Äste im obigen Algorithmus werden bei dem Betrachten eines Knoten ein Zähler erhöht. Am Ende des Algorithmus entspricht dies der Anzahl der Blätter. Der Aufwand in O-Notation wird dadurch nicht beeinflusst.

Ausgegeben werden die erkannten Wiederholungen in abstrakter Form und in Textform, um bei geringen Log-Größen das Ergebnis einfacher validieren zu können und nach der Analyse eine zu verbessernde Handlungsfolge in lesbarer Form zu erhalten. Über die Übergabeparameter lässt sich die gesuchte Wiederholungslänge und somit die Erfolgs- oder Miserfolgsquote für die Anzahl gefundener Wiederholungen anpassen. Ist die gewünschte Länge zu kurz können selbst einfache Handlungsstränge als Probleme ausgegeben werden; ist die Länge zu groß werden möglicherweise nur unproblematische Abfolgen erkannt, welche auf der Aufgabenstellung beruhen. Ist dies eine übliche wiederholte Aufgabe in RCE, so

würde für die Steigerung des *Ease Of Use* ein alternativer kürzerer Weg sinnvoll erscheinen.

5.4.1 Abstraktion in Konfigurationsdateien

Die Erkennung der Pattern umfasst mehr als nur eine zu suchenden Abfolge von Befehlen. Hier sind vollständige Verhaltensmuster gemeint, welche nur durch Algorithmen erkannt werden können.

Weitere Algorithmen in das Programm zu integrieren ließe sich über eine Integration von Scriptsprachen, Bundles für die Integration über das OSGi Equinox Framework oder im einfachsten Fall über Klassen mit einer einheitlichen Schnittstelle. Die Anforderung sah nicht vor die neuen oder anderen Algorithmen ohne den bisherigen Quellcode zu gestalten. Somit genügt eine einheitlich nutzbare Schnittstelle für die Algorithmen und eine einheitliche Ausgabe. Diese sind aus weiteren Klassen nutzbar und sofern die neue Klasse der gewünschten Schnittstelle entspricht, lässt diese sich schnell in die Auswahl der Algorithmen integrieren.

Da die Software bisher nur prototypisch implementiert wurde ist es möglich, dass für manche Algorithmen nicht alle nötigen Informationen zur Verfügung stehen und das Programm nochmals überarbeitet werden müsste. Beispielsweise für einen Vergleich mehrerer Log-Dateien besteht bisher noch keine gute Möglichkeit. Auf Grund der übersichtlichen Struktur der Eclipse RCP Anwendung sollte dies jedoch gut möglich sein. Um das Problem zu umgehen ist die Ausführung eines Algorithmus mit verschiedenen Log-Daten nacheinander möglich, wobei der Algorithmus die Daten zwischenspeichert und erst nach genügend Datensätzen ausgeführt wird.

Das Endergebnis der Klassenbeziehungen und auch die eigenständigen Algorithmen können in der Grafik 14 gesehen werden. Anhand der Ähnlichkeiten zwischen den Grafiken 10, 11 und 14 sollte bemerkbar sein, dass diese sowohl in der Analysesoftware wie auch in RCE selbst verwendet wurden. Wegen dieser Verhaltensähnlichkeiten wurden diese bewusst als vollständige Klassen implementiert und verlinkt anstelle von Kopien, sodass Änderungen an der Aufzeichnung auch direkt in die Analyse mit eingehen und es zu möglichst wenigen Fehlern kommen kann.

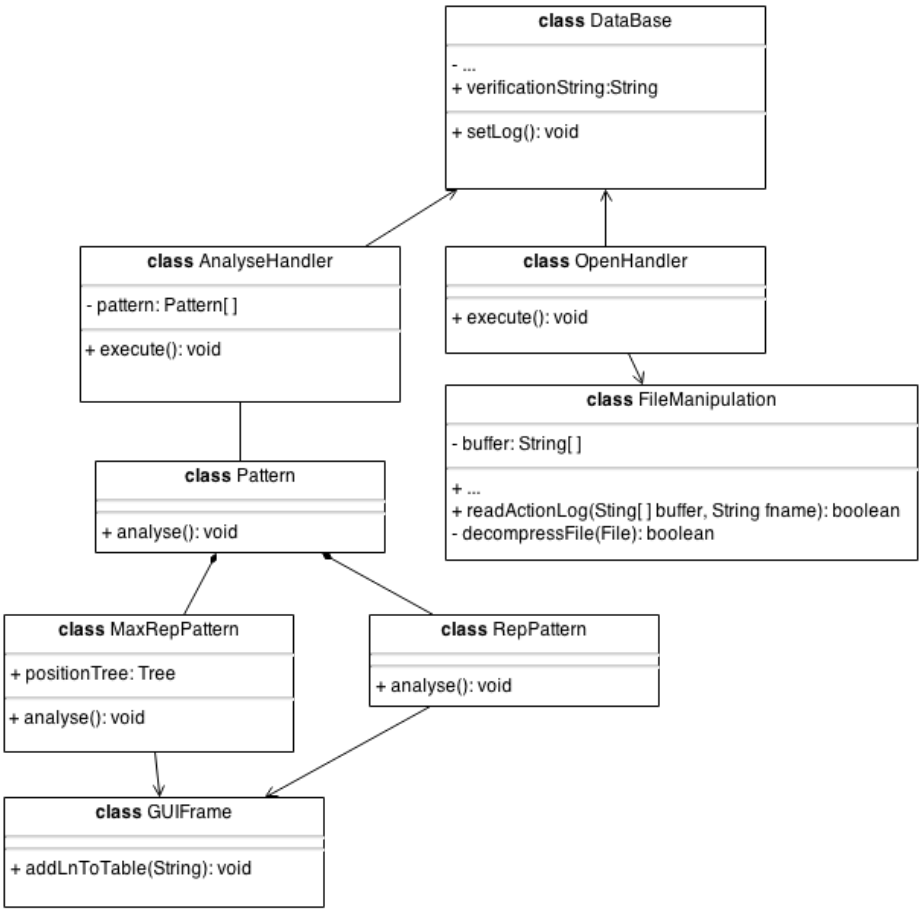

Abbildung 14: Die Klassen bezogen auf die Analysesoftware und ihre Beziehungen

6 Die Nutzerstudie: Log-Erzeugung verglichen mit der Think-Aloud-Methode

In diesem Kapitel wird tendentiell gezeigt, dass die Ergebnisse aus der Log-Analyse in die gleiche Richtung weisen, wie Teilergebnisse aus der Anwendung der Think-Aloud-Methode.

6.1 Versuchsaufbau

Um anhand einer Nutzerstudie Tendenzen erkennen zu können wurden lediglich zwei kleine Nutzergruppen von je vier Personen ausgewählt. Zuerst wurde eine unerfahrene Gruppe zusammengestellt, bei der die Personen RCE noch nicht bedient haben und nur geringe Erfahrungen zu Workflows aufwiesen. Die zweite Gruppe besteht aus Entwicklern der Anwendung RCE selbst. Diese Personen kennen die Teilbereiche der Anwendung für dessen Entwicklung sie selbst verantwortlich sind sehr gut, jedoch sollte die Praxiserfahrung der vollständigen Anwendung und der Benutzeroberfläche nicht vollständig überdeckend sein. Entsprechend dieser Vorgaben ist die Aufgabenstellung auf die selten genutzte und somit auch weniger getestete Excel-Komponente von RCE ausgerichtet. Die Bedienung dieser Komponente entspricht jedoch in etwa der üblichen, allerdings nicht völlig fehlerfreien Bedienung von Komponenten in RCE.

Die Gruppe von völlig unerfahrenen Nutzern bleibt unbeachtet, da die Erstellung von wissenschaftlichen Workflows so speziell ist, dass zu diesem Zweck eine Schulung zur Erzeugung von Grundvorstellungen im Vorhinein notwendig wäre. Solch eine Schulung sollte auf Grund eines anderen Fokus der Arbeit, des Aufwands und möglicherweise auch verfälschten Ergebnissen vermieden werden.

Die erste Gruppe besitzt aus einer Nutzerstudie des vergangenen Jahres noch Hintergrundwissen. Dieses wurde in der Zwischenzeit nicht weiter gefördert und die Bedienung von RCE wurde bereits optimiert und angepasst. Daher sollte das Vorwissen lediglich Grundlagen für das Verständnis bieten und Probleme, welche nach einer kurzen Einführung der Anwendung entstanden wären, bleiben höchst wahrscheinlich erhalten.

Die zweite Gruppe ist durch ihre Arbeit an der Software deutlich erfahrener, doch intuitive Fehler können dennoch auftreten. Zudem kann ein Vergleich der Verhaltensweisen beider Gruppen eine Lerntendenz zeigen beziehungsweise ein bewusstes oder unbewusstes Umgehen von potenziellen Fehlerquellen darstellen. Ebenso kann durch den Vergleich der beiden Nutzergruppen festgestellt werden,

ob Problemstellen der Komponente eine ähnliche Auswirkung auf das Verhalten und somit der Auswertungsergebnisse haben.

Die Gruppengrößen sollten eine Tendenz darstellen und sehr grobe Abweichungen ausschließen können. Da durch die Anwendung dieser automatisierten Usability-Problem-Erkennung je Durchlauf maximal ein Fehler auffindbar ist gelten die sinnvollen Gruppengrößen für eine Fehlerabdeckung bei Usability-Expertentests hier nicht. Auch wenn jede Person ein anderes Problem findet, so kann dies dennoch die Validität der Analyseergebnisse zeigen. Es ist jedoch wünschenswert ein mehrfaches Auftreten eines Fehlers mit ähnlichen Auswirkungen auf die Auswertung des Benutzer-Logs zu beobachten, damit dieser durch verschiedene Probanden bestätigt und dessen Relevanz verdeutlicht wird.

Die im Folgenden dargestellte Aufgabenstellung 15 bietet keine alles umfassende Nutzung der RCE Anwendung aber bereits Berührungspunkte mit möglichen Problemstellen in der Nutzeroberfläche und ebenso in der Analysemethode. Ein weiterführender Test würde nur länger dauern und bezüglich der Bedienung kaum weitere Erkenntnisse bieten. Durch die wiederholten nur leicht modifizierten Aufgabenteile mehrere Inputs zu erzeugen soll gezeigt werden, dass die Aufgabenstellung selber auch ein erkennbares Problem darstellen kann. Die Excel-Komponente zeigt bei der Nutzung teilweise ein unerwartetes Verhalten und so sollte die Reaktion darauf sich in der Analyse zeigen. Trotz der recht genauen Handlungsschritte wurden exakte Anweisungen, welche Schaltflächen zu wählen und welche Inhalte genau einzugeben sind, vermieden und so sollten die Aufgaben ausreichend Spielraum für Fehler, Interpretation und auch andere Lösungswege bieten. Diese Varianz soll die Robustheit der Analyse bestätigen.

Die Nutzerstudie soll möglichst geringe äußere Einwirkung und in möglichst ähnlichem Umfeld geschehen. Einerseits über die Think-Aloud-Methode und andererseits über die Log-Erzeugung sollen die Handlungen des Nutzers dokumentieren werden. Daher wurden die Probanden einzeln für die Aufgabe in einem Büro an einen dafür vorgesehenen Rechner gesetzt und nur unter Beobachtung des Studienmoderators durchgeführt. Da die Probanden alle im selben Büro, denselben Geräten und denselben sonstigen Umständen beeinflusst wurden, werden hier nur einige, wenige Randmerkmale aufgeführt.

Die Probanden sollten an einem Sitzarbeitsplatz an einem aktuellen Computer mit Microsoft Windows 7, einer Drei-Tasten-Maus mit Scoll-Rad, Tastatur und einem Bildschirm mit matter Oberfläche, einer Diagonalen von 24 Zoll (61 cm) und einer Auflösung von 1920x1200 Pixeln arbeiten. Zusätzlich wurde die Aufgabenstellung als Din A4 Ausdruck zur Verfügung gestellt, sodass der Computer nur für die Bedienung von RCE und nicht noch für zusätzliche Zwecke verwendet werden musste. Während der Aufgabe wurden die Probanden gebe-

Aufgabe

Öffnen Sie dieses Dokument auf einem zweiten Monitor oder legen Sie den Ausdruck vor sich, sodass keine Interaktionswechsel zwischen RCE und einem anderen Programm stattfinden müssen.

1. Starten Sie die Aufzeichnung mit dem Ohrsymbol in der Werkzeugleiste
2. Erstellen Sie einen Workflow mit einem beliebigen Namen
3. Fügen Sie die vier unten aufgelisteten statischen Dateninputs in den Workflow ein. Es müssen dazu keine 4 Komponenten erzeugt werden:
   - Integer 5
   - Float 2,5
   - Integer 2
   - Integer 9
4. Fügen Sie nun eine Excel-Komponente ein und verbinden Sie es mit der auf dem Desktop befindlichen Tabelle. Dazu müssen Sie zunächst die Tabelle in ihr Projekt einfügen.
5. Deklarieren Sie für die Komponente die ersten vier untereinander befindlichen Zellen der ersten Tabelle mit den obigen Datentypen als Inputs.
6. Wählen Sie noch die Zelle B1 der ersten Tabelle als Output vom Typ Boolean
7. Verbinden Sie die Inputs der Reihe nach mit den Excel Inputs
8. Starten Sie die Ausführung

Ergebnis

Sollte die Ausführung erfolgreich sein, so erscheint an allen Elementen eine Zielfahne. Im Fehlerfall erscheint ein roter Kreis mit einem X. Sollte der Workflow Fehler enthalten, können Sie versuchen diese zu beheben und einen erneuten Durchlauf zu starten. Sobald Sie erfolgreich sind oder aufgeben möchten, speichern Sie bitte den Workflow ab und stoppen Sie die Aufzeichnung.

Vielen Dank für Ihre Teilnahme.

Abbildung 15: Aufgabenstellung für die Nutzerstudie

ten entsprechend der Think-Aloud-Methode soweit möglich ihre Gedanken laut auszusprechen. Sobald solche Äußerungen ausblieben, wurden die Probanden durch kurze, auf ihre Gedanken und vergangene Handlungen bezogene Fragen zu erneutem, die Handlung begleitenden Äußerungen motiviert.

Damit möglichst wenig Informationen verloren gehen, wurde mit der kostenfreien Open Source Software *CamStudio Portable* der Firma *Rare Ideas* jede Aktion auf dem Bildschirm aufgezeichnet. Mit diesen Hilfsmitteln kann die Auswertung und Aufzeichnung getrennt von Wertung und Analyse geschehen. Dennoch werden beobachtete Gesichtszüge und nicht geäußerte Verhalten notiert und im Nachgespräch angesprochen. Die Nutzung einer Kamera wurde bewusst nicht gewählt, sodass nicht zu viele interpretierbare und somit nicht eindeutige Informationen gesammelt werden. Zusätzlich kann eine Videoaufzeichnung während der Studie zwar genau wie eine direkte Beobachtung die Aussagen verdeutlichen oder ergänzen, doch diese kann dazu verleiten Eindrücke hinzunehmen ohne sie zu hinterfragen und von dem Probanden bestätigen zu lassen. Nach einem gewissen zeitlichen Abstand kann sich der Eindruck der eigenen Gefühlslage des Probanden bereits geändert haben und so nicht mehr verwertbar sein.

Die beobachteten Verhaltensweisen der verschiedenen Probanden werden im folgenden Kapitel strukturiert aufgeführt und genauer untersucht.

6.2 Beobachtungen

Die Tests wurden alle im Nachmittag, mindestens eine Stunde nach dem Mittagessen erstellt, sodass Lichtverhältnisse und die Aufmerksamkeit der Probanden möglichst vergleichbar sind. Für die Tests wurde eine Dauer von etwa 15 Minuten erwartet. Die Probanden ohne große RCE Kenntnissen lagen dabei durchschnittlich bei 12 Minuten mit einer Abweichung von maximal 40 Sekunden und die Probanden mit RCE Kenntnissen bei durchschnittlich 8,5 Minuten mit einer Abweichung von maximal -60 Sekunden (vgl. Tabelle 1). In der Tabelle sind die Probanden ohne RCE Kenntnisse links mit den Buchstaben von A bis D bezeichnet und auf der rechten Seite die Probanden der RCE erfahrenen Gruppe mit den annotierten Buchstaben A_{RCE} bis D_{RCE}.

Proband	A	B	C	D	A_{RCE}	B_{RCE}	C_{RCE}	D_{RCE}
Dauer	12:43	11:50	11:31	11:37	07:35	08:44	08:52	08:52

Tabelle 1: Dauer der jeweiligen Nutzertests

RCE Anfänger waren zusätzlich zu den Unkenntnissen in RCE teils ungeübt mit dem großen Display, so schien das 24 Zoll Display für manche der Probanden

nicht vollständig überschaubar zu sein (siehe Abbildung 16 mit den erkennbaren Details in Abbildung 17). Möglicherweise haben die Tests dadurch einen geringen Einfluss genommen und die Dauer sowie die Notwendigkeit Schaltflächen zu suchen wurde möglicherweise erhöht.

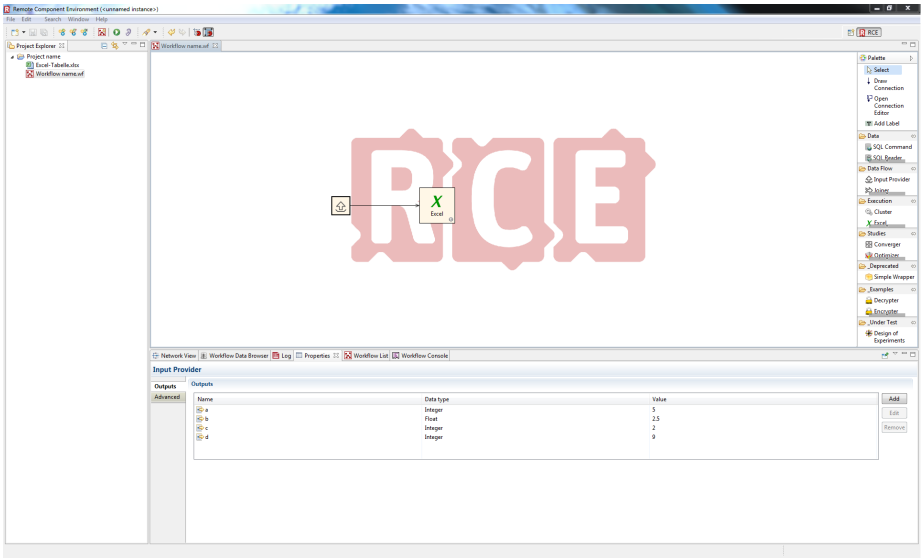

Abbildung 16: Die Anwendung RCE im originalen Größenverhältnis

Das Einschalten des Loggens lief vermutlich wegen der sehr detailliert Erklärung und dem in der Aufgabenstellung dargestellten Symbols einwandfrei und so folgte der zweite Aufgabenteil.

Das Erstellen eines Workflows funktionierte auf Anhieb gut und ohne Umwege. Die Probanden ohne Vorwissen zu RCE nutzten einheitlich den Eintrag im Menü *File* und die Probanden mit RCE Erfahrung nutzten das entsprechende Symbol in der Werkzeugleiste. Probanden mit guten Eclipse und ohne RCE Erfahrung standen nicht in ausreichender Zahl zur Verfügung und somit könnte dies den Umstand erklären, dass Niemand, wie es in Eclipse üblich ist, versuchte ein Projekt vor dem Workflow anzulegen. Dies wird bei den zwei anderen Varianten der Workflowerzeugung von dem Assistenten automatisch im Hintergrund erledigt.

Eine Suche nach den „Inputs“ aus dem dritten Aufgabenteil fiel den Probanden schon schwieriger. RCE Kenner nutzten entweder direkt den *Input Provider* (siehe Abbildung 17 rechts mittig) oder begannen nachzufragen, da es den Input ohne eine Komponente nicht gibt. Diese Probanden konnten sich nicht vorstellen welche Komponente mit einem Input erzeugt werden sollte. Ohne RCE Erfahrung wurde länger nach Möglichkeiten gesucht Komponenten zu dem Workflow einzufügen. Sie erkannten die Arbeitsfläche mit dem RCE-Logo im Hintergrund

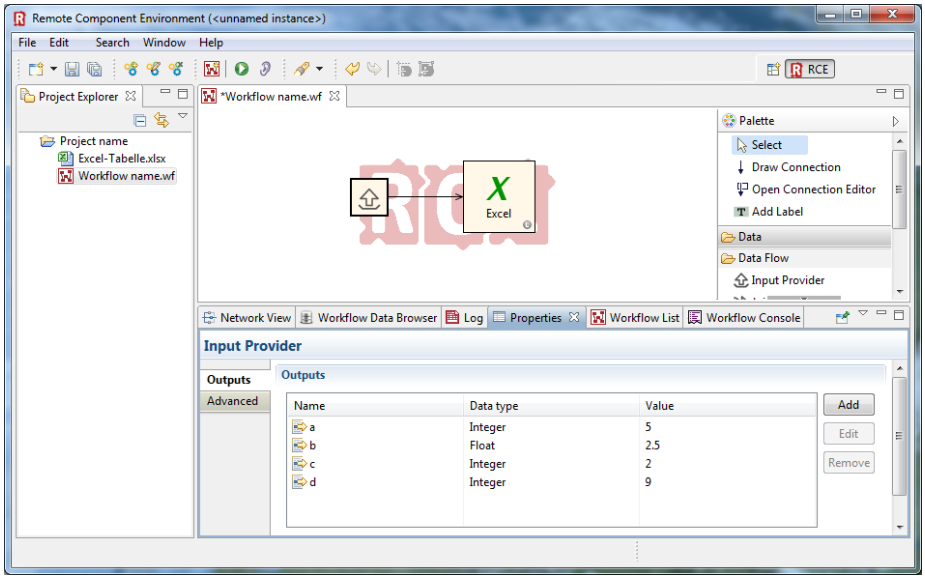

Abbildung 17: Die Anwendung RCE vergrößert mit einem *Input Provider* im Workflow

nicht als solche und somit auch nicht die an der rechten Seite platzierte Palette mit den verfügbaren Komponenten. Nachdem diese als Mittel zum Einfügen von Komponenten erkannt oder der Name *Input Provider* gefunden wurde, konnte die Aufgabe flüssig gelöst werden.

Für die richtige Lösung war lediglich ein *Input Provider* mit mehreren Inputs nötig. Diese Möglichkeit nutzten alle RCE-Kenner, wobei die RCE-Anfänger so viele *Input Provider* erzeugten wie an Inputs nötig waren (siehe Abbildung 18).

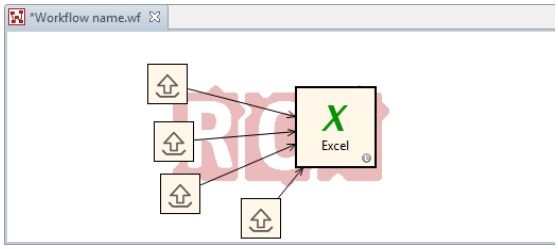

Abbildung 18: Der Workflow der Lösung mit vier *Input Providern*

Bei dieser Teil-Aufgabe kam es bei einem Probanden ohne RCE Kenntnisse noch zu einer weiteren Unklarheit. Bei der Auswahl eines *Input Providers* in der Palette wurde versehentlich das Symbol für die Komponente *Output Writer* doppelt geklickt. Daraus folgte das unbemerkte Erscheinen dieser Komponente auf der Zeichenfläche oben links, recht weit entfernt von der Position der Palette. Nachdem der *Input Provider* auf die Zeichenfläche gezogen wurde, bemerkte

der Proband das Problem und versuchte mit der Rückgängig-Schaltfläche das Objekt zu beseitigen, was jedoch auf Grund eines über den Test entdeckten Fehlers von RCE nicht möglich ist. Mittels Auswahl und entfernen konnte die Aufgabe dann flüssig beendet werden.

Für RCE Kenner war der nächste Schritt die Inputs den *Input Providern* mit Datentypen und Inhalten vorzugeben klar und schnell abgearbeitet. Für die RCE unerfahrenen Probanden wurden in mehreren Fällen die bereits offenen Eigenschaften der Komponente erneut geöffnet. Der Fokus-Wechsel wurde dann zwar erkannt doch die Schaltfläche für das Einfügen der Inputs an der rechten Seite der Liste (siehe Abbildung 17) anscheinend erst verzögert wahrgenommen.

Der erste Schritt des Aufgabenteils vier wurde von allen Probanden schnell gelöst, da sie nun alle nötigen Vorkenntnisse hatten. Lediglich das Verbinden der Excel-Tabelle mit der Komponente zeigte die Schwierigkeit diese vorerst in das Projekt zu kopieren oder einen Link dorthin zu erstellen.

Dies ist beispielsweise über die folgenden drei Wege möglich:

- Mittels Drag and Drop auf das Projektordner-Symbol
- Durch einen Import aus dem Datei System über die Menü-Folge
  Import →File system →Browse (Verzeichniswahl) →(Dateiwahl)
- Über die Menü-Folge
  File →New →Other →File →Advanced →Link to file in the file system →Browse (Dateiwahl)

Die erste Lösung bevorzugten die drei jungen, RCE unerfahrenen Probanden ohne Probleme, vier nutzen teilweise stockend die Import-Funktion und nur ein Proband aus dem RCE Team nutze die letzte Möglichkeit ohne Probleme. Das Zuordnen der Datei zu der Komponente stellte dann keine weiteren Hindernisse bereit, da bereits beim Erstellen der Komponente in den Eigenschaften das nötige Feld durch eine rote Hinterlegung auffiel.

Aufgabenteil fünf wies einige problematische Stellen auf, da gerade die Excel-Komponente Usability-Schwächen hat. Diese Komponente fordert einen auf die Adresse der Zelle je In- und Output einzugeben. Dies kann per Text oder auch über eine Schaltfläche in Excel geschehen, doch diese Schaltfläche ist nicht im passenden Kontext der Text-Eingabefelder zu finden. Excel öffnet sich daraufhin unscheinbar im Hintergrund, ist somit nur auf der Taskleiste zu sehen und für die Auswahl der Zelle muss zuvor eine von der standardmäßig ausgewählten Zelle abweichende Zelle markiert werden. Wird die Excel-Tabelle verändert oder

auch nur die Bearbeitung einer Zelle begonnen, so müssen die Änderungen zuvor verworfen werden bevor die gewählte Zelle in RCE übernommen werden kann.

Aus diesen Problematiken heraus wählten die erfahreneren Probanden nur eine Zelle über den genannten Weg aus und nutzten dann die erstellte Adresse mit leichten Änderungen weiter. Die weniger erfahrenen Probanden wiederholten die Schritte immer wieder mit leichter Verärgerung bis alle Inputs zugeordnet waren.

Da die In- und Output Erstellung identisch ablaufen, gab es bei Aufgabenteil sechs keine zusätzlichen Probleme.

Im Aufgabenteil sieben bestand zuerst die Problematik den *Connection Editor* zu öffnen. Diesen bei den Komponenten in der Palette (siehe Abbildung 17 rechts im oberen Viertel) zu finden bedarf bei den unerfahrenen Probanden zuerst etwas Sucharbeit. Danach wurde unter diesen Probanden meistens das Zeichnen von Verbindungen (*Draw Connection*) gewählt. Dadurch wir der *Connection Editor* auf die zwei an der gezeichneten Verbindung beteiligten Komponenten eingeschränkt und so wird die Aufgabe mehrere Komponenten zu verbinden aufwändiger als nötig. Der *Connection Editor* musste daher für alle vier erzeugten *Input Provider* geöffnet werden, anstelle nur einmalig beim direkten nutzen der Schaltfläche *Open Connection Editor* (siehe Vergleich im Schaubild 19). Ebenso haben die Probanden vom RCE-Team letzteren Editor nicht verwendet, wobei dies bei diesen Probanden lediglich zwei zu verbindenden Komponenten waren und dann auch keinen Unterschied macht.

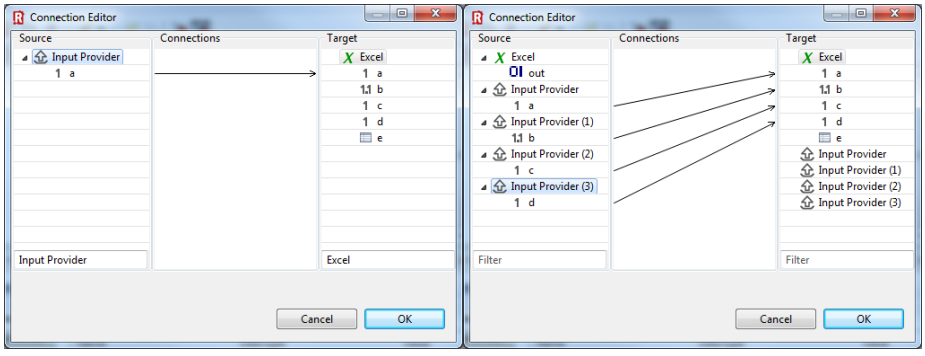

Abbildung 19: Der Connection Editor im Vergleich (links über das Zeichen-Werkzeug und rechts direkt geöffnet)

Obwohl nur die Unerfahrenen auf Grund der vielen *Input Provider* das Auto-Connect Feature nicht nutzen konnten, wurde dies ebenso nicht von dem RCE-Team verwendet. Das Feature verbindet identisch benannte, kompatible In- und Outputs ohne jede Verbindung einzeln ziehen zu müssen indem statt der In- und

Outputs die Komponenten im *Connection Editor* als Quelle und Ziel gewählt werden. Die Möglichkeit bot sich bei drei der vier Probanden aus dem RCE-Team.

Die Möglichkeit und der Erfolg der Ausführung von dem erstellten Workflow erfreuten letztlich alle Probanden und der Erfolg blieb auch bei keinem dieser aus.

Aus den Log-Daten wurden mit den beiden Algorithmen verschiedene Aktionsfolgen gefunden, welche problematisch schienen. Diese Aktionsfolgen sind durch einige Rückgaben keine reinen Aktionsfolgen und somit möglicherweise auch nicht die nach dem MRP gesuchten. Daher wurden noch weitere Filter implementiert, welche einige weitere Rückgaben aus den Log-Daten filtern sollten. Bezüglich der Laufzeit gab es keine empfundene Analyse-Dauer. Die Analyse war ohne bemerkbare Verzögerung abgeschlossen nachdem sie gestartet wurde und somit sind diese Algorithmen auch für deutlich umfangreichere Log-Daten geeignet.

Nach dieser Prozedur konnten die gültigen Aktionsfolgen interpretiert werden.

Bei der ersten Analyse von Proband A zeigte sich eine stark gehäufte Nutzung der *Cancel* Schaltfläche bei dem Setzen der Zellen-Adresse. Diese Nutzung ist sowohl in der MRP, wie auch in der einfachen Wiederholungserkennung mit einer Länge von bis zu 17 Wiederholungen deutlich erkennbar.

Proband B zeigte vor allem bei dem Hinzufügen von In- und Outputs zu den verschiedenen Komponenten Wiederholungen. Diese hatten eine Länge von bis zu sechs Aktionen. Das Besondere an diesen Wiederholungen ist der Bezug zu der Aufgabe, so wurde die Wiederholung für die insgesamt acht In- und Outputs gefunden. Somit wurde scheinbar von der ersten richtigen Handlungsfolge, bis auf die Bezeichnung nicht mehr abgewichen.

Bei Proband C und D zeigte sich eine leichte Variation, sodass weniger In- und Outputs sehr ähnlich mit einer Länge von bis zu 7 Aktionen verliefen.

Ähnlich und untereinander nahezu identisch sehen die Wiederholungen bei den Probanden A_{RCE} bis D_{RCE} aus.

Da in den Wiederholungen die Aufgabenstellung wie erwartet sehr stark zu erkennen war wurden dessen Auswirkungen durch eine nachträgliche Beschneidung der Log-Daten angepasst. Diese Log-Daten enthielten weiterhin die Handlungsfolgen der Aufgabenstellung, jedoch soweit möglich auf nur einen In- und Output beschränkt. Dadurch sollte die Aufgabenstellung nicht mehr in den Analysen zu erkennen sein. Bei dieser Beschneidung wurde zusätzlich bei Proband

A die Folge an *Cancel* Aufrufen entfernt, sodass ein anderes Muster erkannt werden kann.

Die Log-Daten wurden durch die Einschränkung an zwei bis drei Stellen verändert wobei die Anzahl der Aktionen wie in Tabelle 2 zu erkennen ist deutlich sank. Prozentual sind die Unterschiede in der Anzahl unterschiedlicher Aktionen im Vergleich zu der Gesamtzahl jedoch kaum verändert worden, da die gelöschten Log-Elemente hauptsächlich Wiederholungen bereits vorhandener Aktionen darstellen.

Proband	A	B	C	D	A_{RCE}	B_{RCE}	C_{RCE}	D_{RCE}
Aktionszahl$_{alt}$	268	206	172	333	160	138	184	184
Unterschiedliche	79	70	55	109	61	54	76	76
Aktionszahl$_{neu}$	144	148	99	254	94	68	108	101
Unterschiedliche	56	64	46	95	51	34	63	61
MRP Länge	5	4	4	7	2	4	3	3

Tabelle 2: Beschränkung der Log-Daten auf die Erstellung von nur einem In- und Output

In den neuen Ergebnissen konnten auch bei fast allen Probanden (sechs der acht Probanden) nicht mit der Aufgabe zusammenhängende Wiederholungsstellen identifiziert werden. Bei Proband A zeigten die Log-Daten einige Klicks auf dem Arbeitsbereich mit der Tastenfolge *LRLLL*, wobei L für die linke und R für die rechte Maustaste steht. Dies war einmal nachdem die Komponenten zu dem Workflow hinzugefügt wurden und einmal direkt nach dem Einfügen des Outputs. Bei dieser Folge wird die aktuelle Auswahl der Komponenten aufgehoben und das Kontextmenü geöffnet und wieder geschlossen. Dazu hätte bereits die Tastenfolge LRL ausgereicht. Möglicherweise sollte noch ein Doppelklick durchgeführt werden, welcher zu langsam für die Erkennung durchgeführt wurde. Da jedoch keine weiteren Aktionen zwischen den Klicks existieren brachte die Aktion anscheinend nicht das gewünschte Ergebnis und entspricht somit einem Suchmuster. Die Daten von Proband B und C zeigten ebenso ein Klickmuster mit der Tastenfolge *LR* gefolgt von einer Mausbewegung über die Menü-Einträge *Undo/Redo* des Kontext-Menüs und ebenso einmal direkt nach dem Einfügen der Komponenten und einmal vor dem Verbinden dieser. Die Aktionen hier entsprechen von der angedeuteten Absicht den obigen. Es zeigt sich ebenso ein Suchmuster durch das Deselektieren der Komponenten gefolgt von dem Öffnen des Kontext-Menüs und dem Überfahren der ersten Menü-Einträge. Proband D zeigte deutlichere Probleme in den Log-Daten, wie in der Tabelle 2 anhand der vom MRP gefundenen Wiederholungslänge von sieben zu erkennen ist. Genauer betrachtet zeigen die Log-Daten noch weitere ähnliche Stellen, welche für das MRP mit den gegebenen Filtern nicht genügend Ähnlichkeiten aufwiesen.

Teilweise stehen die Stellen mit einem Fokussieren verschiedener Menü-Einträge ohne einen genutzten Eintrag im direkten Zusammenhang. Zu den Stellen gehören die folgenden:

- Zweimal direkt nach dem Erstellen des Workflows als die Inputs eingefügt werden sollten.
- Einmal vor dem Verbinden der Komponenten.

Bei dem Verhalten werden verschiedene Bereiche des Fensters aktiviert, doch ohne ein daraus folgendes Resultat. Demnach und auch wegen dem Zusammenhang mit dem Öffnen der Menüs ist dies ein Suchmuster, welches das Fehlen von Informationen oder die nicht intuitiv erwartete Position der gesuchten Schaltflächen aufzeigt.

Wie in Tabelle 2 zu sehen ist hatte Proband A_{RCE} „Wiederholungen“ der Länge 2 und diese waren zu dem nicht häufig. Das lässt auf eine zufällig auftretende Wiederholung schließen und daher einem problemlosen Verlauf der Aufgabe.

Proband B_{RCE} zeigte weiterhin die übrig geblieben Wiederholung aus der Aufgabenstellung, so sind sich die Erstellung des In- und Outputs ähnlich doch mit der Länge von nur vier Aktionen unproblematisch. Auch bei der einfachen Wiederholungserkennung mit geringerer Länge zeigten sich keine anderen Wiederholungen, womit der Aufgabenverlauf als problemlos anzusehen ist.

Die hier vorliegenden beiden ineinander verschachtelten Wiederholungen der Nutzung der *Cancel*-Schaltfläche bei Proband C_{RCE} deuten auf eine fehlende Reaktion auf die Nutzeraktion hin. Der Vorfall ereignete sich während der Eingabe der Tabellenreferenzen zu Excel. Alleine auf Grund der Log-Daten solle die Aktionsfolge nachvollzogen werden, wodurch das Problem bei einer versehentlichen Editierung der Tabelle erkannt werden sollte.

Der letzte Proband D_{RCE} zeigte die gleichen Wiederholungsstellen, wie bei Proband B_{RCE} und C_{RCE}. Somit gelten hier die gleichen Aussagen.

Zunächst wurden wie erwartet Problemstellen durch wiederholte nur leicht unterschiedliche Aufgabenteile kaschiert, doch ohne diese sind ein problemloses und ein problematisches Verhalten erkennbar. Die Betrachtung und genauere Analyse der Problemstellen ist trotz der Automatisierung noch recht aufwendig; dies ließe sich anhand weiterer Analysen mit einer Anzahl an ähnlichen Analyseergebnissen zu dieser Wiederholung mit einer Gewichtung versehen. So müssen erst bei häufigerem Auftreten von ähnlichen Problemen die Log-Daten genauer und manuell betrachtet und analysiert werden.

6.3 Analyse im Vergleich zur Beobachtung

Die Anwendung der Think-Aloud-Methode stellte sich schwieriger dar als erwartet. Trotz vieler Anregungen die Gedankengänge zu äußern und immer wiederholter Fragen zu dem was die Probanden taten, fiel es diesen schwer diese unübliche Art des Handelns umsetzen. Die Probanden ließen sich zwar auf Fragestellungen ein, welche vermutlich eine ähnliche Aussage ergaben wie geäußerte Gedanken ergeben hätten, doch dadurch wurden die Probanden stärker beeinflusst als dies beabsichtigt war. Die Probanden ließen sich völlig aus der Benutzung der Anwendung ablenken und wirkten in ihrem Handeln stark gestört. Für weitere Nutzerstudien sollten zuerst Übungen mit den Probanden zu dieser Methode durchgeführt und dafür ein Vielfaches der Zeit eingeplant werden. Wegen der nicht darauf ausgerichteten Zeitplanung wurden die Tests mit den zusätzlichen Übungen in diesem Rahmen nicht durchgeführt.

Anhand der hier entsprechend gewählten Aufgabenstellung konnte sowohl eine Kaschierung der gesuchten Probleme durch die Aufgabenstellung gezeigt werden als auch durch eine entsprechende Anpassung der Log-Daten die gesuchten Probleme an sich. Eine andere Erklärung für dieses Verdecken der Ergebnisse könnte auch die genutzte Methode in Verbindung mit der geringen Praxiserfahrung des Moderators und der Probanden mit dieser sein. Die Fragen und wenigen Hinweise waren vielleicht zu viele oder wurden zu früh gegeben, sodass Wiederholungen und Suchmuster nicht so deutlich entstehen konnten, wie es die Erkennung neben der Aufgabenstellung benötigte.

Auf Basis der angepassten Log-Daten konnten alle Probleme aus dessen Analyse ebenso in der Beobachtung erkannt werden. Diese Probleme sind mit allen in der Beobachtung erkannten Problemen verglichen die Schwerwiegendsten. Daher sollten an den entsprechenden Programmstellen noch einmal durchdacht werden wie diese intuitiver gestaltet werden können. Der Verlauf der Aufgaben bei den RCE erfahrenen Nutzern war wie erwartet bis auf die den Probanden unbekannten Probleme mit der Excel-Komponente reibungslos, was sich in den Log-Daten und der Beobachtung gleichermaßen zeigte.

Die genutzte Suche nach Wiederholungen hob vorwiegend Suchmuster nach Funktionen hervor, welche Problemen von unerfahrenen Nutzern entsprechen. Somit wurde das angestrebte Ziel mit der Analyse vor allem auf Probleme des „ease of learning“ einzugehen erreicht.

Wegen der geringen Größe der Probanden zeigten sich keine unterschiedlichen Lösungswege mit sich gleichenden Problemen, somit konnte die Robustheit der Methode nicht gezeigt werden. Dennoch konnte durch die sich stark ähnelnden Muster bei den verschiedenen Probanden die Relevanz des gefundenen Problems verdeutlicht werden. Eine Lösung dieser Probleme wurde auch bereits in Teamgesprächen angesprochen und wird somit in naher Zukunft angegangen.

7 Zusammenfassung und Ausblick

Die Aufgabenstellung eine Erweiterung für Programme mit einer auf SWT basierten Benutzeroberfläche zu erstellen, um Usability-Schwächen anhand von Log-Daten erkennen zu können, wurde in dieser Arbeit erreicht.

Die Erweiterung greift die Mittel von aspektorientierten Programmierung auf und ergänzt somit eine auf Eclipse RCP aufbauende Software *Remote Component Environment RCE* des *Deutschen Luft und Raumfahrtzentrum DLR* ohne dessen Quellcode vorher zu verändern oder dafür vorzubereiten. Mit diesen Mitteln werden hauptsächlich die bereits vorhandenen Funktionen, welche auf die *Events* der Benutzerinteraktion reagieren, um ein Log-Verhalten erweitert. Die Gemeinsamkeiten der Namensgebung und Übergaben dieser Funktionen bieten dazu die genutzten Ansatzpunkte für das Einweben des neuen *Aspekts* auszulösen. Bei dynamisch zur Laufzeit erstellten Objekten wie den Menüs wurde zusätzlich der hierarchische Aufbau von SWT genutzt und bei anderen Bibliotheken dessen Hierarchien.

Die Veränderung des Verhaltens der Ursprungssoftware RCE wird für das Einweben während der Laufzeit um den *Aspekt* des Loggens angepasst, wobei *AspectJ* als Sprache und die zugehörigen Werkzeuge aus dem *AspectJ Develepment Toolkit AJDT* genutzt wurden. AspectJ und das AJDT sind eines der am meisten genutzen Hilfmittel in Java für die auspektorientierte Programmierung.

Auf die gewonnen Log-Daten können mit der zusätzlich erstellten Software verschiedenen Algorithmen angewendet werden, welche über weitere Klassen nach dem gegebenen Vorbild erweiterbar und modifizierbar sind. Für diese Algorithmen können zusätzlich noch Filter angewendet werden, um unwichtige oder das zu erkennende Verhaltensmuster unkenntlich machende Log-Daten zu entfernen. Hier wurde eine einfache Wiederholungserkennung mit vorgegebener Länge und eine Erkennung von Wiederholungen maximaler Länge implementiert.

In der Nutzerstudie konnte gezeigt werden, dass bereits dieser Algorithmus Hinweise auf mögliche Usability-Schwächen und ebenso auf eine problemlose Nutzung geben kann. Zusätzlich wurde auch eine Schwachstelle der genutzten Muster aufgezeigt, sodass bereits aufgabenbedingte Wiederholungen die Erkennung der problematischen Stellen stark verfälschen kann. In einem Ergebnis können so die problematischen Stellen kaschiert sein, da mit dieser Methode nur die Schwachstelle aufgezeigt werden kann, welche sich in einem Muster maximaler Länge zeigt.

Für den praktischen Einsatz der Log-Erstellung und der Analyse ist noch etwas Optimierung nötig, sodass nicht jeder Log-Datensatz einzeln von dem Nutzer

übertragen werden muss und daraufhin nicht jedes einmalig auftretende Muster zu einer genauen Betrachtung der Log-Daten und somit viel Aufwand führt.

Zu dem ersten Problem ließe sich die Sammlung der Daten durch ein einfaches Bestätigen des Nutzers gefolgt von einer webbasierten, automatischen Übertragung zu einem Server der Entwickler vereinfachen. Bisher ist dies nur über eine manuelle Weitergabe der Log-Datei möglich.

Durch Kommentare zu den eingesandten Log-Daten oder eine Bewertung der aktuellen Zufriedenheit mit der Bedienung könnte das zweite Problem angegangen werden und so unnötige oder unerwünschte Anpassungen verhindert werden.

Die Algorithmen könnten um vielfältige weitere erweitert werden, wie zum Beispiel eine Aufrufhäufigkeit von Schaltflächen, Menüpunkten und Ähnlichem um die weitere Entwicklung und Fehlerbehebung mehr an den genutzten Fähigkeiten der Software auszurichten. Ebenso kann das zeitliche Verhalten der Nutzereingaben weitere Schlüsse über das Verhalten dieses zulassen.

Die Filter von Aktionen ließen sich von der globalen Anwendung auf eine Analyse basierte Anwendung anpassen, da für manche Verhaltensmuster gewisse Aktionen störend sind, welche für andere Muster wiederum essentiell sind.

Das Log-Verhalten könnte auch Bewegungsmuster der Maus wie Schleifen, gezielte Auswahlen über kurze Wege, lange Intervalle ohne Bewegung oder auch Lesemuster (zeilenweise Bewegung des Mauszeigers über Texte) mit einbeziehen. Dadurch ließen sich möglicherweise weitere Nutzerprobleme erkennen.

Es gibt also noch vielfältige Erweiterungsmöglichkeiten für die Anwendung der Log-Erzeugung und Analyse und daher bleibt es weiteren Arbeiten vorbehalten das Thema einer automatisierten Usability-Analyse erneut aufzugreifen und zu verfeinern.

Glossar

Android Developement Toolkit (ADT) ADT ist eine integrierte Entwicklungsumgebung basierend auf Eclipse RCP zur Entwicklung von Apps für Adroid Geräte. 14

API Eine Anwendungsprogrammierschnittstelle (engl. Application Programing Interface) ist eine Schnittstelle, welche anderen Programmen zur Interaktion zur Verfügung gestellt wird. 15

AspectJ Development Toolkit (AJDT) Siehe Kapitel 2.2.2. 11, 13

Aspekt Orientierte Programmierung (AOP) Siehe Kapitel 2.2.1. 13

BPML Die Business Process Modeling Language (BPML) ist eine XML-basierte Metasprache zur Beschreibung von Geschäftsprozessmodellen. 17

BPMN Business Process Model and Notation (BPMN) ist eine grafische Beschreibungssprache für Geschäftsprozessen beziehungsweise Arbeitabläufen. 17

Bundle Die Eclipse RCP basierten Anwendung RCE besteht aus einem Kernstück, der OSGi Plattform mit Equinox Framework. Dabei ist Equinox das Bindeglied, welches alle Bundles verbindet und je nach Bedarf startet und wieder beendet. Ein Bundle ist somit ein Baustein oder auch eine Erweiterung einer so aufgebauten Anwendung. In der IDE Eclipse stellt sich solch ein Bundle als eigenes Bundle Projekt dar und wird durch eine zentrale Startkonfiguration oder Abhängigkeiten zu andren Bundles gestartet. 29

Compiler Programm, welches Quellcode zu Bytecode oder Maschinensprache übersetzt. 14

Cross-Cutting Concern (C-CC) Ein Cross-Cutting Concern ist eine gemeinsame Eigenschaft verschiedener Objekte und Elemente eines Produktes. Beispielsweise kann das gemeinsame Verhalten der Aktualisierbarkeit aller grafischen Benutzerschnittstellen Elemente als solcher beschrieben werden. 12

DLR Abkürzung für Deutsches Zentrum für Luft und Raumfahrt. 3

Ease of Learning Die Usability Analysen können den Fokus auf *Ease of Learning* legen, wenn es das Ziel ist einem neuen, nicht geschulten Nutzer den Einstieg zu vereinfachen und attraktiver zu gestalten. 26

Ease of Use Die Usability Analysen können den Fokus auf *Ease of Use* legen, wenn es das Ziel ist einem geschulten, professionellen Nutzer das Arbeiten zu vereinfachen und attraktiver zu gestalten. 26

Enum Der Datentyp Enum ist ein Enumerator und somit ein mit Sprechenden Namen versehener Stellvertreter für eine Nummerierung. Vergleichbar ist der Enum mit einer Sammlung von Konstanten mit fortlaufendem Wert, welche für den Zugriff auf Arrays, die Verwendung in Switch–Case Strukturen und der Zuordnung von Zuständen, Kategorien oder anderer Eigenschaften besonders geeignet ist. 34

Grafical Editing Framework (GEF) Das GEF dient bei RCE zur darstellung des Workflows in einem Diagramm. Über dieses Framework wird ebenso eine Werkzeug-Palette, das Drag and Drop und Snapping-Verhalten sowie die Wegfindung der Elementverbindungen integriert. 34

Integrated Developement Environment (IDE) Eine integrierte Entwicklungsumgebung ist ein Programm, welches viele verschiedene Tools zu einem gemeinsamen Ziel in einer Umgebung/Programm vereint. Bei der PRogrammierung zählen dazu meistens ein Texteditor, ein Compiler oder Interpreter, ein Linker und ein Debugger. 14

Maximum Repetition Pattern (MRP) Ein Pattern beschrieben von [25] um Usability-Probleme anhand eines User-Logs effizient zu identifizieren. Deutet darauf hin, dass die Bedienung, welche zu dem angestrebten Ziel führt, nicht intuitiv zu finden ist. 2, 38

Open Source Open Source ist ein Vermarktungsprinzip, welches dafür garantiert, dass jeder Nutzer kostenfrei an den Quellcode der Software kommt. Somit lassen sich Sicherheitsaspekte jederzeit kontrollieren. Die Finanzierung solcher Software wird teilweise über Support oder über bezahlte neue Features geregelt. 16

Pattern Aus dem Englischen bedeutet das Wort Muster und wird auch ebenso übergreifend verwendet. Es kann sowohl eine exakte Abfolge von Aktionen, wie auch eine durchschnittlich abdeckende Nutzung aller Interaktionselemente sein. 26

Remote Component Environment (RCE) Siehe Kapitel 2.4. 1

Rich Client Platform (RCP) Siehe Kapitel 2.3. 14, 17

Runtime Bibliothek Dies ist eine Bibliothek, welche von einem kompilierten Programm verwendet wird, um Standardaufgaben auszuführen. Dabei enthält das kompilierte Programm an vielen stellen Aufrufe von Funktionen in der Bibliothek. 14

Separation of Concerns Dies ist ein Programmmierparadigma, welches thematisch nicht zusammenhängende Programmteile soweit möglich trennt. Dies kann sowohl durch Objektorientierung oder auch Aspektorientierung geschehen. 32

Standard Widget Toolkit (SWT) Siehe Kapitel 2.3.1. 1, 3, 14

Usability Expertentest Siehe Kapitel 2.1.1. 2

Wildcard Wildcards sind in Ausdrücken oder Beschreibungen Platzhalter ähnlich den regulären Ausdrücken. Beispielsweise steht ein '*' für kein oder mehrere beliebige Zeichen und ein '?' steht für mindestens ein beliebiges Zeichen. 31

Literatur

[1] Alemneh, Esubalew: Current States of Aspect Oriented Programming Metrics. In: *International Journal of Science and Research* School of Computing and Electrical Engineering, Bahir Dar University, 2014, S. 142–146

[2] Cifaldi, Frank: *Take Two Estimates Lowered After Disappointing Duke Sales*. 2011. – gamasutra.com/view/news/35674/Take_Two_Estimates_Lowered_After_Disappointing_Duke_Sales.php. Einsicht: 31.05.2014

[3] Crenguta Madalina Bogdan, Luca Dan S.: Formal Modeling of Concurrent AOP Programs. In: *INTERNATIONAL JOURNAL of COMPUTERS, COMMUNICATIONS & CONTROL* Agora University, 2006, S. 92–97

[4] Emil Alégroth, Lisa R. Robert Feldt F. Robert Feldt: *Visual GUI testing in practice: challenges, problems and limitations*. Springer US, 2014. – 1ff S. ISSN 1382–3256. – link.springer.com/article/10.1007/s10664-013-9293-5#page-1. Einsicht: 14.07.2014

[5] Erich Gamma, Lee N. ; Wiegand, John: *Eclipse AspectJ: Aspect-Oriented Programming with AspectJ and the Eclipse AspectJ Development Tools*. 1. Addison Wesley Pub Co Inc, 2005 (Eclipse). – ISBN 9780321245878

[6] Foundation, The E.: *eclipse*. 2014. – www.eclipse.org. Einsicht: 01.07.2014

[7] Foundation, The E.: *Eclipsepedia*. 2014. – wiki.eclipse.org. Einsicht: 29.04.2014

[8] Gilan M. Saadawi, Olga Medvedeva Girish C. Elizabeth Legowski L. Elizabeth Legowski ; Crowley, Rebecca S.: A Method for Automated Detection of Usability Problems from Client User Interface Events. In: *AMIA 2005 Symposium Proceedings* University of Pittsburgh, 2005, S. 654–658

[9] GmbH, Beck I.: *ONE Workbench*. 2012. – www.beck-ipc.com/de/products/sc2x3/oneworkbench.asp. Einsicht: 01.07.2014

[10] Grigoreta Sofia Moldovan, Adriana Mihaela T.: Developing an Usability Evaluation Module Using AOP. In: *INTERNATIONAL JOURNAL of COMPUTERS, COMMUNICATIONS & CONTROL* Agora University, 2006, S. 320–325

[11] H. Brau, F. S.: *Methoden der Usability Evaluation: wissenschaftliche Grundlagen und praktische Anwendung*. Huber, 2011 (Aus dem Programm Huber: Wirtschaftspsychologie in Anwendung). books.google.de/books?id=Qw1PSwAACAAJ. – ISBN 9783456848839

[12] Hempel, Rolf: *Simulations- und Softwaretechnik*. o.J.. – www.dlr.de/sc/desktopdefault.aspx/tabid-1185. Einsicht: 03.07.2014

[13] Inc., Google: *Android SDK | Android Developers*. 2014. – developer.android.com/sdk/index.html?hl=sk. Einsicht: 01.07.2014

[14] Lingala, Srikanth R.: *Zip4j - Java library to handle Zip files*. 2013. – www.lingala.net/zip4j. Einsicht: 06.08.2014

[15] Ltd., Adobe Systems Software I.: *Adobe Flash Builder 4.7 Premium*. 2014. – www.adobe.com/de/products/flash-builder.html. Einsicht: 01.07.2014

[16] o.A.: *DLR im Überblick*. 2013. – www.dlr.de/dlr/desktopdefault.aspx/tabid-10443/637_read-251. Einsicht: 03.07.2014

[17] o.A.: *Software*. o.J.. – www.dlr.de/sc/desktopdefault.aspx/tabid-1189. Einsicht: 03.07.2014

[18] Okada, Hidehiko ; Asahi, Toshiyuki: GUITESTER: A Log-Based Usability Testing Tool for Graphical User Interfaces. In: *Transactions on Information and Systems, VOL.E82-D* IEICE, 1999, S. 1030–1041

[19] Oracle: *KeyGenerator*. o.J.. – docs.oracle.com/javase/7/docs/api/javax/crypto/KeyGenerator.html. Einsicht: 06.08.2014

[20] Robert LiKamWa, Nicholas D. L. Yunxin Liu L. Yunxin Liu: Can Your Smartphone Infer Your Mood? In: *PhoneSense Workshop* Microsoft Research Asia, 2011, S. 1–5

[21] Rubin, Jeffrey ; Chisnell, Dana: *Handbook of Usability Testing*. 2. John Wiley & Sons, 2008. – ISBN 9780470185483

[22] S. J. Westerman, C. Alder C. W. Wyatt-Millington N. M. Shryane C. M. Crawshaw G. R. J. H. S. Hambly H. S. Hambly: Investigating the human-computer interface using the Datalogger. In: *Behavior Research Methods, Instruments & Computers* University of Hull, University of Bradford, 1996, S. 603–606

[23] Schwalbe, K.: *Introduction to Project Management, Second Edition*. Course Technology, 2008 books.google.de/books?id=3Bvo1CVCsVoC. – ISBN 9781423902201

[24] Shah, I.: Event Patterns as Indicators of Usability Problems. In: *J. King Saud University, Vol. 20* College of Computer & Information Sciences, King Saud University, 2007, S. 31–43

[25] Siochi, Antonio C. ; Ehrich, Roger W.: Computer Analysis of User interfaces Based on Repetition in Transcripts of User Sessions. In: *ACM Transactions on Information Systems, Vol. 9, No. 4* Christopher Newport College and Virginia Tech, 1991, S. 309–335

[26] Team, RCE D.: *RCE Distributed, Workflow-driven Integration Environment*. 2014. – code.google.com/a/eclipselabs.org/p/rce. Einsicht: 01.07.2014

[27] Terence S. Andre, H. Rex H. ; Williges, Robert C.: Determining the Effectiveness of the Usability Problem Inspector: A Theory-Based Model and Tool for Finding Usability Problems. In: *HUMAN FACTORS, Vol. 45, No. 3* U.S. Air Force Academy and Virginia Polytechnic Institute and State University, 2003, S. 455–482

[28] Vasundhara, B.: Improving the Software Quality using AOP. In: *CVR JOURNAL OF SCIENCE & TECHNOLOGY* CVR College Of Engineering, 2013, S. 13–17

[29] Vogel, Lars: *The complete guide to Eclipse application development*. Second Edition. Lars Vogel, 2013. – 734 S. – ISBN 978–3–9437–4707–2. – www.vogella.com/tutorials/EclipseRCP/article.html. Einsicht: 26.04.2014

[30] Vogel, Lars: *Eclipse 4 RCP*. 2013. – www.vogella.com/tutorials/EclipseRCP/article.html. Einsicht: 29.04.2014

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde eingereicht worden.

Aachen, August 2013

Aachen, im Monat Jahr (vollständige, handschriftliche Unterschrift)