Philipp Rausch, Bakk. techn.

Methoden der Agilen Softwareentwicklung als Einzelentwickler am Beispiel einer Location-Based-Service Applikation mit besonderem Augenmerk auf Möglichkeiten zur Umsetzung von GUI-Testing

MASTERARBEIT

zur Erlangung des akademischen Grades Diplom-Ingenieur

Masterstudium Informatik

Technische Universit Graz

Betreuer:
Univ.-Prof. Dipl.-Ing. Dr.techn. Wolfgang Slany

Institut für Softwaretechnologie

Graz, im März 2013

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am ....

(Unterschrift)

Danksagung

Hiermit möchte ich jenen Personen danken, die mich während meines Studiums und beim Verfassen dieser Arbeit unterstützt und begleitet haben. Ich danke meinen Eltern, die mich stets motivierten und förderten, ich danke meine Lebenspartnerin, die mir fortwährend - auch in schwierigen Situationen - beistand. Ebenso möchte ich der Firma AIONAV und den Mitarbeitern des Instituts für Bauinformatik der TU Graz unter der Leitung von Professor Ulrich Walder für die gute und herzliche Zusammenarbeit danken. Und nicht zuletzt gilt mein dank Professor Wolfgang Slany für die ausgezeichnete Betreuung dieser Arbeit.

Diese Arbeit behandelt technische Themen aus der Informatik und der Softwareentwicklung. Die englische Sprache ist in diesen Bereichen allgegenwärtig. Viele Begriffe lassen sich nicht ohne weiteres ins Deutsche übersetzten. Sie werden deshalb ohne Übersetzung im originalen Wortlaut übernommen.

Zusammenfassung

Im Rahmen dieser Masterarbeit des Studienganges Informatik an der Technischen Universität Graz wurde eine Desktop-Applikation zum Aufbereiten von digitalem Kartenmaterial im Auftrag der Firma AIONAV Systems Inc. (im Folgenden kurz AIONAV genannt) entwickelt. Die Applikation sollte das simple Einzeichnen von interessanten Orten auf Plänen und dessen Verknüpfung mit textuellen und multimedialen Inhalten ermöglichen. Eine mobile App sollte als Gegenstück zu der Desktop-Anwendung die Navigation von Benutzern auf dem aufbereiteten Kartenmaterial ermöglichen. Die App wurde parallel zu dieser Arbeit von Entwicklern der Firma AIONAV umgesetzt. Schließlich wurden von der Desktop-Anwendung und der mobilen Anwendung Prototypen im Rahmen eines Projektes mit dem schweizerischen Handelskonzern Migros entwickelt. Das Projekt *Einkaufen der Zukunft* sollte dadurch realisiert werden. Einkaufsmärkte sollten mit der Desktopanwendung digital aufbereitet werden. Kunden der Märkte sollten mit der mobilen App zwischen Geschäften und Produkten navigieren können. Die Desktop-Anwendung wurde mittels der Zuhilfenahme von agilen Methoden der Softwareentwicklung umgesetzt. Problemstellungen, wie Einzelentwicklung mittels Extreme Programming, Erweiterung von nicht-agil-entwickelter Software mittels Test Driven Development und testgetriebene Entwicklung von Grafischen User Interfaces wurden analysiert, umgesetzt und evaluiert. Im Zuge dieser Arbeit wird außerdem gezeigt, wie agile Methoden der Softwareentwicklung angepasst werden können, wenn keine optimalen Bedingungen für deren Anwendung vorliegen. Neben dem Projekt *Einkaufen der Zukunft* sind weitere Projekte für den Einsatz der beiden Applikationen denkbar bzw. in Planung.

Abstract

As part of a master thesis in computer science at the Technical University of Graz a desktop application has been developed in collaboration with the company AIONAV Systems Inc. (shortly: AIONAV). AIONAV is specialised on indoor and outdoor navigation. The application was developed on the base of a software framework by AIONAV. Basic requirements for the application were the possibility to build models of geographical areas containing detailed information of special places called points of interest. The application had to enable the creation of points of interest on particular places in a plan and the addition of information, such as text, audio or video material. Additionally, a mobile application (shotly app) was developed by AIONAV which allows users the navigaion on plans between points of interest. The swiss retail group Migros ordered prototypes of the desktop application and the app to realize a project evaluating these applications in shopping malls. With the help of the desktop application a mall should be virtualized. The mobile application should allow to navigate between stores or to find particular products. The desktop application was developed using agile software development methods. The setting was not optimal for using these methods as only one person was implementing the desktop application as a single developer and a framework not developed with agile methods, thus not constantly tested, had to be used as basis of development. Finally the possibilities of Test Driven Development regarding GUI testing were investigated. All these methods hav been explored, used and evaluated. Specific challenges and problems regarding the use of these methods and adaptions to the development process are demonstrated in this work.

Inhaltsverzeichnis

1. Einleitung

Die folgende Arbeit beschreibt die Umsetzung eines Projektes im Rahmen einer Masterarbeit für das Masterstudium Informatik an der Technischen Universität in Graz. Im Auftrag der Firma AIONAV Systems Inc. (im Weiteren einfach AIONAV genannt) sollte die Implementierung einer Applikation, aufbauend auf einem bestehenden Software-Framework, erfolgen. AIONAV entwickelt Positionierungs-Lösungen für den Indoor- und Outdoorbereich. Die zu entwickelnde Applikation, der AIONAV LBS Editor, sollte es ermöglichen, Kartenmaterial mit gewissen Zusatzinformationen zu interessanten Orten zu versehen. Parallel zu der in dieser Arbeit beschriebenen Applikation wurde von AIONAV außerdem eine mobile App auf Android entwickelt. Diese setzt auf dem selben Software-Framework auf. Die App sollte es ermöglichen, Benutzer auf dem vom AIONAV LBS Editor aufbereiteten Kartenmaterial zu navigieren. Der AIONAV LBS Editor sollte möglichst generisch implementiert werden, um ihn für viele verschieden Anwendungsfälle nutzen zu können.

Im Laufe des Projektes kristallisierte sich ein konkreter Anwendungsfall für die Anwendungen heraus. Der schweizerische Handelskonzern Migros beauftragte AIONAV mit der Entwicklung einer personalisierten Variante des AIONAV LBS Editors und der mobilen App, um ein Projekt zum Thema *Einkfauen der Zukunft* zu verwirklichen. Mit dem AIONAV LBS Editor sollte es möglich sein, Kaufhäuser von Migros virtuell zu gestalten. Kunden sollten sowohl zwischen Geschäften hin- und her navigieren können, als auch innerhalb von Migros-eigenen Lebensmittelmärkten einzelne Produkte auffinden können.

Zentrales Thema bei der Umsetzung sollte die Anwendung agiler Methoden der Softwareentwicklung sein. Als konkrete Methodik wurde Extreme Programming ausgewählt. Der Sinn hinter der Verwendung agiler Methoden war unter anderem, auf die extrem veränderlichen Anforderungen an die zu entwickelnde Applikation eingehen zu können. Es galt auch gewisse Herausforderungen im Rahmen des Projektes zu meistern, da nicht die optimalen Bedingungen für die Verwendung der Methodik des Extreme Programming vorlagen. So wurde die Applikation etwa - untypisch für Extreme Programming - von einer Einzelperson alleine entwickelt (allerdings in Zusammenarbeit mit dem restlichen Entwicklerteam von AIONAV). Außerdem mussten Methoden gefunden werden, um mittels Test Driven Development auf das nicht testgetrieben entwickelte Framework von AIONAV aufzusetzen. Die Methoden des Extreme Programmings mussten auf die veränderten Projektbedingungen angepasst werden. Weiters wurden im Zusammenhang mit Test Driven Development verschiedene Methoden und Tools evaluiert und getestet, um auch die grafische Oberfläche testgetrieben zu entwickeln.

Abschnitt 2 befasst sich mit theoretischen Aspekten der agilen Softwareentwicklung.

Die Methodik des Extreme Programmings wird als Beispiel für agile Methoden vorgestellt. Weiters werden Vorgehensweisen der agilen Softwareentwicklung kritisch betrachtet und Möglichkeiten zur Anpassung an veränderte Projektbedingungen analysiert. Abschnitt 3 beschäftigt sich mit den theoretischen Aspekten des Testens von Software. Wichtige Grundprinzipien, wie Testkriterien oder Testabdeckung werden erläutert. Vorgehensweisen, wie das Test Driven Development und wichtige Konzepte für dessen Umsetzung werden vorgestellt. Weiters werden spezielle Problemstellungen im Bezug auf das Testen von grafischen User Interfaces analysiert. Außerdem werden verschiedene GUI-Test-Tools vorgestellt, analysiert und miteinander verglichen. Abschnitt 4 beschreibt schließlich die Umsetzung des AIONAV LBS Editors im Detail. Abschließend wird in Abschnitt 5 gezeigt, wie agile Methoden und GUI-Testing im Rahmen dieser Arbeit zum Einsatz kamen und welche Herausforderungen und Probleme sich im Laufe des Projektes ergaben.

2. Agile Softwareentwicklung

2.1. Überblick

Der Begriff der *Agilen Softwareentwicklung* fasst Vorgehensweisen und Methodiken zusammen, welche sich dadurch auszeichnen, dass gewisse, *agile Verfahren*, wie sie im Folgenden beschrieben werden, zum Einsatz kommen. Das Wort *agil* leitet sich vom lateinischen Wort *agilis* ab, was soviel heißt wie *rasch, schnell oder flink*. In der Softwareentwicklung bedeutet *sich agil verhalten*, dazu bereit zu sein, rasch zu reagieren und auf Veränderungen einzugehen. Eine allgemeingültige Definition oder ein globales Konzept der Agilen Softwareentwicklung existiert jedoch nicht. Es gibt vielerlei verschiedene Ansätze, wobei einer der bedeutendsten *Extreme Programming* nach *Kent Beck* ist (siehe Abschnitt 2.3 über Extreme Programming). Weitere Ansätze sind etwa *Scrum*, *Crystal Methods*, *Feature-Driven Development* oder *Adaptive Software Development* [1].

2.1.1. Das Agile Manifest

Das *Agile Manifesto* [22] beschreibt die Grundzüge der Agilen Softwareentwicklung. Es wurde 2001 von Autoren wichtiger Werke zum Thema Agile Softwarentwicklung verfasst. Es beschreibt eine Sichtweise bei der gewisse Eigenschaften der Softwareentwicklung anderen vorgezogen werden. Konkret würdigt man bei der Agilen Softwareentwicklung...

1. ...**Individuen und Interaktionen** mehr als Prozesse und Werkzeuge
2. ...**Funktionierende Software** mehr als umfassende Dokumentation
3. ...**Zusammenarbeit mit dem Kunden** mehr als Vertragsbedingungen
4. ...**Reagieren auf Veränderung** mehr als das Befolgen eines Plans

Ad Punkt 1: In der Agilen Softwareentwicklung wird mehr Wert auf Beziehungen und Gemeinschaft zwischen Entwicklern anstatt auf Prozesse und Entwicklungswerkzeuge gelegt. Dies zeigt sich etwa durch enge Zusammenarbeit von Teammitgliedern, beispielsweise durch räumliche Nähe (gemeinsames Büro etc.).

Ad Punkt 2: Das Ergebnis des Entwicklungsprozesses ist es, unablässig getestete Software zu entwickeln. In kleinen bis sehr kleinen Zeitabständen (Stunden, Tage bis Monate) werden neue Releases der Software herausgebracht. Jeder Release bringt ein neues Stück funktionierender Software hervor.

Ad Punkt 3: Stetige Zusammenarbeit zwischen Entwicklern und Stakeholdern ist wichtiger als das strikte Festhalten an (einmalig ausdiskutierten) Vertragsbedingungen. Die

in Punkt 2 erwähnten Releases erreichen den Kunden regelmäßig, um sicherzustellen, dass die Anforderungen Schritt für Schritt erfüllt werden. Die Relevanz eines gut aufgesetzten Vertrages wird hierdurch jedoch nicht in Frage gestellt.

Ad Punkt 4: Unter der Voraussetzung, dass beteiligte Entwickler und Kunden die nötige Kompetenz besitzen, um Anpassungen an einem Projekt zu bestimmen und vorzunehmen, sollen geänderte Anforderungen an eine Anwendung möglichst schnell in den Entwicklungsprozess (nachfolgender Release) einfließen. [1]

2.1.2. Charakterisierung der Agilen Softwareentwicklung

Es folgen nun Auflistungen wichtiger Gesichtspunkte der Agilen Softwareentwicklung aus der Sicht von nennenswerten Autoren zu der Thematik: [1]

Nach *Cockburn* [8] sind die wesentlichen Punkte der Agilen Softwareentwicklung, die ein erfolgreiches Projekt ausmachen, die folgenden:

- Zwei bis acht Personen in einem Raum
- Einbezug und ständige Anwesenheit von Stakeholdern
- Inkrementelle Entwicklung mit kurzen Zyklen
- Automatische Testmethoden (Regressionstests)
- Erfahrene Entwickler im Team

Miller und Lee [9] charakterisieren Agile Softwareentwicklung wie folgt:

- Modularität auf Entwicklungsebene
- Kurze, iterative Entwicklungszyklen zur schnellen Verifikation (und Korrektur) von Anforderungen
- Zyklen von 1 bis 6 Wochen
- Von Release zu Release - Schritt für Schritt - eine funktionierende Anwendung entwickeln
- Gezielte Einsparungen im Entwicklungsprozess ersparen überflüssige Tätigkeiten
- Anpassung an mögliche, neue Risiken so schnell wie möglich durchführen
- Orientierung der Entwicklung am Mensch, nicht an Prozessen oder Technologien
- Kooperative Arbeitsweise

Favaro [17] sieht in der agilen Softwareentwicklung vor allem das Reagieren auf sich ändernde Anforderungen als zentralen Angelpunkt. Anforderungen werden von Iteration zu Iteration eingeführt, geändert oder auch wieder entfernt. Seiner Ansicht nach ist es auch wichtig, die Gestaltung von Verträgen an die sich ändernden Anforderungen anzupassen. Das detaillierte Festhalten von Anforderung in Verträgen ist demnach nicht sinnvoll.

Im Bezug auf die richtige Ausarbeitung des Vertrages weisen *Higsmith und Cockburn* [19] darauf hin, dass es letztendlich wichtiger ist, den Kunden zum Zeitpunkt der Auslieferung von Software zufrieden zu stellen, als zum Projektbeginn. Dies wiederum ist nur durch Anpassung des gesamten Entwicklungsprozesses umsetzbar. Veränderungen der Anforderungen müssen eingeplant und korrekt gehandhabt werden. Es darf nicht versucht werden, Veränderung von Anforderungen zu unterbinden. Erst dann kann man den Stakeholder auf lange Sicht hin zufriedenstellen. Sie fassen außerdem agile Methoden wie folgt zusammen:

- Bereits in den ersten Wochen werden Prototypen produziert,um schnell Feedback zu erhalten
- Es werden einfache Lösungen gefunden, die demnach auch leicht zu verändern sind
- Die Design Qualität wird pausenlos erhöht
- Es wird fortlaufend getestet, um Fehler möglichst früh zu erkennen

Ambler [3] hebt 5 wichtige Aspekte hervor, welche für agile Softwareentwicklung bestimmend sind:

- Menschen sind ein zentraler Punkt in der Softwareentwicklung
- Agile Methoden ermöglichen es, weniger zu Dokumentieren
- Kommunikation ist eine wichtige Angelegenheit
- Modellierungs-Tools sind nicht so wichtig, wie einst gedacht
- Up-Front Design ist nicht notwendig

Zusammengefasst legt man bei agilen Methoden also sehr viel Wert auf Schlichtheit, Geschwindigkeit, aber auch auf Flexibilität. Wichtige Funktionen werden zuerst geliefert, Feedback darüber trifft rasch ein. Bestimmend für agile Softwareentwicklung sind vor allem die **inkrementelle Entwicklung**, sehr **kooperative Prozesse**, **Einfachheit** und **Anpassungsfähigkeit**. [1]

2.1.3. Praktiken der agilen Softwareentwicklung

Um zu veranschaulichen, was agile Softwarentwicklung in der Praxis ausmacht, folgt ein Überblick durch eine exemplarische Aufzählung gängiger Praktiken nach *Sletholt et al.* [36]. Es handelt sich dabei um Praktiken aus dem Extreme Programming sowie aus Scrum. Auf einige davon wird im Abschnitt über Extreme Programming nochmals detailliert eingegangen.

- Tägliches Standup Meeting
- Regelmäßige kleine Releases
- Das Projekt ist in viele Iterationen aufgeteilt
- Verwaltung von Prioritäten und Anforderungen
- Rollenverteilung im Entwicklerteam
- Aufwandsabschätzung von einzelnen Aufgaben
- Regelmäßige, kurze Meetings
- User Stories (niedergeschrieben)
- Offene Gestaltung des Arbeitsplatzes um gute Kommunikation zu ermöglichen
- Messung der Projektgeschwindigkeit
- Starker Kundenkontakt (möglichst als Teil des Entwicklerteams immer anwesend)
- Coding nach einheitlichen Standards
- Test First Development
- Pair Programming
- Häufige Integration von Code ins Gesamtsystem
- Code ist gemeinsames Eigentum aller Entwickler
- Einfaches Design
- Regelmäßiges Refactoring von Code
- Hohe Testabdeckung
- Tests laufen regelmäßig

2.2. Vergleich traditioneller und agiler Methoden

Agile Methoden der Softwareentwicklung unterscheiden sich in gewissen Punkt stark von traditionellen Methoden der Softwareentwicklung (wie etwa Wasserfallmodell oder Spiralmodell). Die Situation beim Einsatz agiler Methoden ist sowohl für Entwickler, als auch für Management und Kunden eine veränderte. Vielen Gesichtspunkten wird höhere Wichtigkeit zugemessen, andere werden vernachlässigt. Traditionelle Methoden sind meist sehr *Prozess-orientiert*. Es gibt Phasen mit Personen in festgesetzten Rollen, wobei das Ergebnis jeder Phase fix festgelegt ist. Kunden werden nur in bestimmten

Phasen stärker einbezogen (Anforderungsanalyse). Die Kommunikation findet in formaler Form über Dokumente statt (Pflichtenheft, Spezifikation etc.). Im Gegensatz dazu sind agile Methoden *Personen-orientiert*. Die einzelne Person und ihre Kreativität und Leistung steht mehr im Mittelpunkt, als der Prozess selbst. Es gibt kurze, schnelle Iterationen. Kunden und ihr Feedback sind wichtig und vor allem regelmäßig verfügbar. Kommunikation steht im Vordergrund und findet über den gesamten Projektzeitraum intensiv statt. [32] Tabelle 2.1 zeigt nochmals die wichtigsten Unterschiede zwischen traditionellen und agilen Methoden der Softwareentwicklung auf.

2.3. Extreme Programming

Im folgenden Abschnitt wird der Ansatz von *Extreme Programming* (XP) detailliert erläutert. Es werden die Grundwerte dieser Philosophie vermittelt, sowie auch Methoden, die für XP wesentlich sind, vorgestellt.

Extreme Programming ist ein bekannter und bedeutender Ansatz der Agilen Softwareentwicklung. Als Begründer des *Extreme Programming* gilt Kent Beck, welcher in mehreren Werken die Grundzüge der Methodik beschreibt. (z.B. [6], [5]). Mit einer Parallele zum Straßenverkehr versucht Kent Beck den Leitgedanken von XP zu beschreiben:

"Beim Autofahren geht es nicht darum, ein Auto geradlinig und exakt in eine Richtung auszurichten und es fahren zu lassen, sondern darum, es mit viel Aufmerksamkeit zu steuern und immer wieder kleine Korrekturen vorzunehmen."
Kent Beck über das Paradigma des Extreme Programming [6]

2.3.1. Grundwerte des XP

XP ist eine eigene Philosophie, um Software zu entwickeln. Die Grundwerte des XP sind **Kommunikation**, **Feedback**, **Einfachheit**. Kommunikation wird als ein wichtiges Element in der Team-Entwicklung hervorgehoben: Ideen, Lösungen und Erfahrungen müssen von Teammitgliedern untereinander ausgetauscht werden. Wissen, welches nur eine Person alleine hat, ist für das Team nicht (oder nur bedingt) von Nutzen. Ein System nach dem Prinzip der Einfachheit zu entwickeln, ist ebenso eine große Herausforderung. In diesem Sinne stellt sich überhaupt die Frage: Ist eine Lösung einfach bzw. wie mache ich eine Lösung einfach? Natürlich können Probleme nicht einfacher gemacht werden, als sie sind. Das Prinzip der Einfachheit bezieht sich nur darauf, dass man keine zusätzliche, unnötige Komplexität in ein Projekt einbringen sollte. Die Einfachheit kann z.B. durch Einhaltung und Ausnutzung der zuvor genannten Kommunikation erreicht werden: Werden Probleme früh genug diskutiert und Anforderungen zeitgerecht an die Realität angepasst, ergibt sich dadurch Einfachheit. Feedback ist ebenso ein wichtiger Bestandteil der Philosophie des XP. Es geht vor allem darum Feedback so schnell

	Traditionell	**Agil**
Grundsätze	System kann komplett spezifiziert werden, Abläufe und Anforderungen sind vorhersagbar. Massive Planung im Vorhinein wird durchgeführt.	Qualitative und anpassungsfähige Software kann von kleinen Teams durch kontinuierliche Implementierung neuer Funktionalitäten und durch diszipliniertes Testen entwickelt werden. Rasches Feedback und schnelle Veränderungen sind die Grundlage.
Fokus	auf Prozesse	auf Personen
Management Stil	Anordnungen und Kontrolle durch Führungsperson	Führung aber auch Koordination und Zusammenarbeit durch Führungsperson
Rollenverteilung	Individuelle Aufgabenverteilung auf einzelne Personen	Teamarbeit, Rollen und Aufgaben wechseln
Kommunikation	Formell (über Dokumente)	Informell (Face-to-Face, häufige Meetings)
Rolle des Kunden	Wichtig (Kunde ist in frühen Phasen stark eingebunden, später weniger)	Kritisch (Kunde begleitet das gesamte Projekt, gibt durchwegs Feedback)
Zyklus	Bestimmt durch Aufgaben und Aktivitäten	Bestimmt durch Produktfeatures
Entwicklungsmodell	Life Cycle Modelle (wie z.B. Wasserfallmodel)	Evolutionäre Modelle
Organisationsstruktur	Bürokratisch mit hoher Formalisierung	Organisch, flexibel und anpassungsfähig, sozial geprägt
Eingesetzte Technologien	Keine Einschränkungen	Bevorzugt Objektorientiert

Tabelle 2.1.: Die wesentlichen Unterschiede traditioneller und agiler Methoden [32]

wie möglich zu bekommen. Kurze Release-Zyklen sind deshalb zweckdienlich. Sobald Feedback vom Kunden eintrifft, können Probleme beseitigt werden. Mit Courage ist im Allgemeinen ein bestimmtes Verhalten in einer einschüchternden oder erschreckenden Situation gemeint. Im Sinne von XP bedeutet es ganz einfach, auf bekannte Probleme einzugehen, anstatt sie zu ignorieren. Problematisch ist nur, wenn auf Probleme in unangemessene Weise reagiert wird. Deshalb ist ein Zusammenspiel von Courage mit den anderen Grundwerten des XP extrem wichtig. Als weiterer, wichtiger Grundwert des XP wird **Respekt** genannt. Damit ist vor allem gemeint, dass jede Person im Entwicklerteam gleich behandelt werden soll. Menschlichkeit ist oberstes Gebot und erhöht zugleich die Produktivität. [6]

2.3.2. Hauptpraktiken des XP

Es folgt nun eine Beschreibung einiger Methoden, die beim Ansatz des XP von besonderer Bedeutung sind. [6]

Zusammensitzen

Ein zentraler Arbeitsplatz für alle Mitglieder eines Entwicklerteams ist laut der Philosophie des XP essentiell. Teammitglieder müssen die Möglichkeit haben, sich untereinander auszutauschen, zusammen an Problemen zu arbeiten und zu diskutieren. Durch die räumlich nahe Anordnung der Arbeitsplätze wird die Kommunikation untereinander verstärkt. Natürlich lässt sich diese Praxis nicht für beliebig viele Entwickler in einem Raum umsetzen. Ein Projekt lässt sich aber auf mehrere Teams aufsplitten. Die Grundaussage des XP dazu ist jedoch: Je mehr Zeit Team-Mitglieder zusammen verbringen, umso mehr steigt die Produktivität. Im Kontext von XP gibt es spezielle Meetings, auch *Planing Games* genannt. Es geht dabei darum, mit Kunden, Entwicklern und Management gemeinsam Anforderungen zu erfassen, abzuschätzen und zu priorisieren.

Stories

Anforderungen bzw. Use Cases werden beim XP mit Hilfe von Stories beschrieben. Eine Story wird auf einer Storycard festgehalten. Stories erhalten einen kurzen Namen und eine kurze Beschreibung oder graphische Skizze. Der Aufwand um eine Story umzusetzen, sollte möglichst früh abgeschätzt werden und auf der Karte ebenso festgehalten werden. Durch diese Information können Stories in entsprechender Reihenfolge zur Umsetzung ausgewählt werden. Die Storycards werden auf Basis des *Planning Games* entworfen.

Teamarbeit

XP ist vor allem für Team-Projekte geeignet. In erster Linie sollten die Teammitglieder so gewählt werden, dass die Teammitglieder sich mit ihren verschiedenen Kompetenzen gegenseitig ergänzen. Im Team ist ein Gefühl der Zusammengehörigkeit wichtig. Auch die Größe eines Teams muss gut durchdacht sein. Ein Team, welches täglich zusammenarbeitet sollte aus nicht mehr als 12 Mitgliedern bestehen. Für große Teams ist eine

Maximalgröße von 150 Personen empfohlen. Personen in mehreren Teams zugleich einzusetzen, ist nicht empfehlenswert, da der Aufwand des *Task-Switchings* relativ hoch ist.

Informativer Arbeitsplatz

Der Arbeitsplatz soll im Sinne des XP das aktuelle Projekt repräsentieren. Eine Person, die den Arbeitsplatz betritt, sollte sich im Idealfall sofort ein Bild darüber machen können, an was an diesem Ort gearbeitet wird. Eine gute Möglichkeit um das zu verwirklichen, ist das Aufhängen von Storycards auf einem Storyboard. Die Storycards können dabei in verschiedene Kategorien aufgeteilt werden, z.B. *Erledigt, Diese Woche, Dieser Release, Zukünftiges* etc. Anhand der Bewegung der Storycards auf dem Storyboard lassen sich auch Aussagen über den Projektfortschritt treffen (Beispiel: es dauert lang, bis Storycards in die Kategorie *Erledigt* kommen - daraus lassen sich Konsequenzen ziehen).

Energievolle Arbeit

Grundsätzlich gilt: Ein Entwickler sollte nur solange arbeiten, solange er produktiv sein kann. Überarbeitung sollte vermieden werden. Ebenso ist es unproduktiv, etwa im Falle einer Erkrankung zu arbeiten. Nur im ausgeruhten und gesunden Zustand ist man für das Team auch nützlich. Richtzeit für die maximale Arbeitszeit sind in etwa 40 Stunden pro Woche.

Pair Programming

Beim Pair Programming wird von zwei Personen an einem PC programmiert. Eine Person programmiert, die andere verfolgt die Schritte des Anderen. Maus und Tastatur werden abwechselnd von beiden bedient. Pair Programming ist ein Dialog zwischen zwei Personen, die zusammen analysieren, designen und testen. Durch Pair Programming entstehen entscheidende Vorteile: Beide Partner behalten im Auge, was der andere macht; Ideen werden klarer ausgedrückt. Kommt ein Partner nicht mehr weiter, kann der andere seinen Platz übernehmen - Frustration tritt nicht so schnell auf.

Wöchentliche Zyklen

Die Arbeit an einem Projekt sollte durch wöchentliche Zyklen bestimmt werden. Zu Anfang jeder Woche sollte in einem beginnenden Meeting...

- ...der aktuelle Status und Fortschritt des Projektes bewertet werden
- ...der Fortschritt mit dem von vergangenen Wochen verglichen werden
- ...der Kunde/Benutzer Storycards auswählen, die im Laufe der Woche behandelt werden
- ...jede Story in kleinere Tasks aufgeteilt werden, die den Teammitglieder zugewiesen werden

Zu Wochenbeginn sollten außerdem automatisierte Tests passend zu den gewählten Storycards geschrieben werden. Am Ende der Woche sollten die Tests im Idealfall erfolgreich durchlaufen werden können. Das Ziel der Entwicklung soll aber nicht sein, die Tests zu passieren, sondern auslieferbare Software zu entwickeln. Sollte im Laufe der Woche klar werden, dass sich gewisse Stories nicht mehr ausgehen, so kann immer noch umgeschwenkt werden, um zumindest die wichtigsten Stories einer Woche zu erfüllen.

Quartals-Zyklen

Parallel zu den wöchentlichen Zyklen sollte am Ende jedes Quartals ein größeres Meeting stattfinden, welches den Fortschritt und Status eines oder mehrere Projekte in einem größeren Rahmen darlegt. Wichtige Punkte dabei sind u.a.:

- Das Finden von Engpässen/Problembereichen in der Entwicklung
- Lösungen für etwaige Probleme finden
- Allgemeine Ziele der Organisation planen

Spielraum einplanen

In jedem Plan sollte Platz für Unvorhergesehenes gelassen werden. Der Wegfall einiger weniger wichtiger Stories sollte zum Beispiel möglich sein. Umgekehrt sollte es auch möglich sein, im Nachhinein noch Stories hinzufügen zu können. Die richtige Auslastung aller Team-Mitglieder ist hier ebenso ein wichtiger Punkt. Sind Mitarbeiter mit zu vielen Punkten beschäftigt, leidet die Qualität der entwickelten Software.

10 Minuten - Build

Im Idealfall sollte ein System während der Entwicklung alle 10 Minuten rebuildet und getestet werden. Ein längerer Build-Vorgang ist nicht mehr von Vorteil und würde wahrscheinlich nicht mehr so oft ausgeführt werden. Durch regelmäßige Builds erhält man auch regelmäßig Feedback. Im Idealfall erfolgt ein Build des ganzen Systems und der Durchlauf aller Tests **automatisch**. Die automatische Ausführung ist viel zuverlässiger als regelmäßige, manuell angestoßene Builds. Mit steigendem Stresslevel der Entwickler würde nämlich ziemlich sicher der Antrieb fehlen, regelmäßig (manuelle) Builds durchzuführen und diese zu testen.

Continuous Integration

Das Prinzip der Continuous Integration besagt, dass neu geschriebener Code möglichst schnell ins gesamte System integriert und getestet werden sollte. Die Integration von neuem Code in ein System ist oft recht aufwändig. Je früher die Integration erfolgt, umso weniger Aufwand ist dafür nötig. Zusätzlich wird neuer Code schnell getestet und dessen Qualität dadurch erhöht. Man kann zwischen **asynchronem** und **synchronem** Vorgehen unterscheiden. Bei der synchronen Variante wird nach jeder längeren

Pair-Programming Session Code in das gesamte System integriert. Nach einem Build kann überprüft werden, ob alle Tests durchlaufen. Bei der asynchronen Variante werden regelmäßig auch kleinere Änderungen ins Gesamtsystem integriert. Der nächste 10-Minuten-Build gibt einem bereits Feedback über den zuvor integrierten Code.

Test-First Programmierung

Beim Test-First Programming wird vor dem eigentlichen Code, der eine Problemstellung lösen soll, ein Test geschrieben. Es hat mehrere bedeutende Vorteile:

- Mit dem Schreiben des Tests wird implizit eine Anforderung für ein Stück Software formuliert.
- Tests für entkoppelten und guten Code sind in der Regel einfach zu schreiben. Hat man Probleme damit, einen Test zu schreiben, weist einen dies möglicherweise auf ein grundsätzlicheres Design-Problem hin.
- Andere Entwickler können sich auf sauberen Code, dessen Funktionsfähigkeit durch einen Test belegt ist, verlassen.
- Test-First Programmierung gibt einem Programmierer eine Richtung vor. Schließlich verliert man beim Programmieren manchmal auch das eigentliche Ziel aus den Augen.

Einbindung von Kunden

Im Rahmen der Entwicklung mit Extreme Programming ist es sehr wichtig, regelmäßigen Kundenkontakt zu haben. Konkret sollte der Kunde selbst Teil des Entwicklerteams sein und so oft wie möglich für Fragen und Feedback zur Verfügung stehen. Im Idealfall gibt es von einer auftraggebenden Firma also eine Person, die eigens zu diesem Zwecke abgestellt wird. Einerseits ist dies eine der größten Herausforderungen bei Anwendung des XP andererseits ein großes Problem, wenn der Kundenkontakt nicht stark genug ist.

Dies waren die wichtigsten Praktiken von Extreme Programming nach *Beck*. [6] In den folgenden Abschnitten werden Vor- und Nachteile von agilen Methoden aufgezeigt, aber auch mögliche Anpassungen für veränderte Situationen (Einzelentwicklung, veränderte Projekteigenschaften etc.) aufgezeigt. Zuvor werden noch die verschiedenen Rollen der Projektmitglieder im XP beschrieben.

2.3.3. Rollen der Projektmitglieder

In einem Team, welches im Sinne des Extreme Programming entwickelt, gibt es für die verschiedenen Personen auch unterschiedliche Rollen. Die Rollenverteilung ist aber nicht zwingend in dieser Form umzusetzen, sondern eher eine Art Richtlinie. Die folgenden Rollen, haben sich in vielen Projekten etabliert: [6]

- **Programmierer:** Programmierer sind das Kernstück des Teams und auch diejenigen, die Extreme Programming maßgeblich umsetzen. Neben der Umsetzung von Methoden der Softwareentwicklung ist aber auch Kommunikation eine wichtige Aufgabe der Programmierer.

- **Kunde:** Selbstverständlich nimmt der Kunde eine sehr wichtige Rolle im Sinne des Extreme Programming ein. Der Kunde ist vor allem dafür zuständig, zu vermitteln, was umgesetzt werden soll. Ebenso gibt er Feedback zu bestehenden Komponenten. Der Kunde trifft Entscheidungen und beeinflusst damit den weiteren Projektverlauf. Im Idealfall sollten Kunden zudem funktionale Tests anfertigen können.

- **Tracker:** Der Tracker des Teams ist in gewissem Maße *das Gedächtnis des Teams*. Er behält den Überblick über Ausgaben, Aufwände, Ressourcen, Zeit uvm. und kann somit auch Aussagen über den aktuellen Stand einer Iteration machen, wie zum Beispiel Abschätzungen, ob sich gewisse Tasks bis Iterations-Ende ausgehen.

- **Tester:** Da das Testen eine wichtige und häufige Aufgabe im Extreme Programming darstellt, sollte es Personen geben, die die Hauptverantwortung für das Testen tragen. Personen in dieser Rolle sollten auch Kunden beim Anfertigen von funktionalen Tests unterstützen. Die Rolle kann auch von Personen, die bereits eine andere Rolle übernehmen, zusätzlich ausgeübt werden.

- **Coach:** Eine Person in der Rolle des Coaches übernimmt vor allem Verantwortung über Vorgänge und Prozesse auf sich. Ein Coach muss ein sehr detailliertes Projekt-Verständnis aufweisen, um in Problemfällen die richtigen Vorgehensweisen zur Problemlösung wählen zu können. Er ist sozusagen ein Koordinator der Programmierer.

- **Big Boss:** Der Big Boss ist gewissermaßen der Teamchef und übernimmt vorwiegend Management-Aufgaben. Seine Aufgabe besteht vor allem darin, das Team zu motivieren, zu leiten und zu führen.

2.3.4. Anpassung von XP an Einzelentwicklung

Extreme Programming ist im Allgemeinen eine Methodik, die am besten für die Entwicklung in kleinen Teams geeignet ist. Viele der angewandten Praktiken bauen darauf auf, dass mehrere Personen daran beteiligt sind. Zum Beispiel der Review von bestehendem Code, oder die Praktik des Pair Programming, die als solches von einer Person alleine natürlich gar nicht anwendbar ist. *Agarwal et al.* [2] haben sich mit der Thematik auseinandergesetzt und eine Form der Softwareentwicklung von XP abgeleitet, die sich auch für Einzelentwickler (also Entwicklerteams bestehend aus genau einer Person) eignet - **PXP** (*Personal Extreme Programming*). Grundlegende Abläufe sind ähnlich bzw. gleich wie beim konventionellen Extreme Programming. Es gibt eine Planungsphase sowie eine Entwicklungsphase, in welcher wie beim Extreme Programming gewisse Storys

ausgewählt werden, in Tasks heruntergebrochen werden, und schließlich umgesetzt werden. Es wird iterativ entwickelt, ebenso gibt es regelmäßige kleine Releases. Allerdings müssen gewisse Vorgehensweisen angepasst werden, um sie für eine Einzelperson brauchbar zu machen. Personal Extreme Programming baut auf den selben Grundwerten wie Extreme Programming auf - **Kommunikation**, **Feedback**, **Einfachheit** auf. Im Folgenden wird darauf eingegangen, wie gewisse Praktiken angepasst werden müssen bzw. welche problemlos übernommen werden können.

Planning Game

Die Aufgaben beim Planing Game sind das Schreiben, Abschätzen und Priorisieren von Story Cards. Auch als Einzelentwickler ist dies möglich und sogar förderlich. Das Aufteilen von Aufgaben auf Storys ist eine gute Hilfe beim Unterteilen des Projektes. Die Herausforderung des Entwicklers ist es allerdings, alle Rollen des Entwicklungsteams zu übernehmen. Abschätzungen von Priorität und Aufwand werden nur von einer Person getroffen. Wichtige Aspekte können übersehen werden und zu Fehleinschätzungen führen. Die Erfahrungswerte und Einwände anderer Teammitglieder fehlen möglicherweise bei Entscheidungen. Eventuell muss der Einzelentwickler außerdem die Rolle des Kunden übernehmen. Es läuft in jedem Fall darauf hinaus, dass er die Aufgaben und Anforderungen aus verschiedenen Perspektiven betrachten können muss.

Viele, kleine Releases

Das Herausgeben von neuen Programmversionen in kurzen Abständen ist als Einzelentwickler problemlos umsetzbar.

Einfaches Design

Auch das Beibehalten eines simplen Designs ist als Einzelentwickler gut möglich. In gewisser Weise kann es sogar einfacher sein, weil eine Person immer den gesamten Überblick über das Projekt hat.

Test Driven Development

Test Driven Development ist ohne Einschränkungen auch für einen Einzelentwickler umsetzbar. Tests werden vor dem eigentlichen Code geschrieben, nur getesteter und funktionierender Code wird ins Projekt aufgenommen.

Refactoring

Regelmäßiges überarbeiten von Code ist gängige Praxis in der agilen Softwareentwicklung. Auch als Einzelentwickler ist dies möglich. Zu bedenken ist, dass natürlich dieselbe Person, die den ursprünglichen Code geschrieben hat, auch das Refactoring durchführt. Mögliche Denkfehler und strukturelle Probleme könnten so übersehen werden.

Pair Programming

Pair Programming ist eine wichtige Praktik des Extreme Programming. Code wird durchgehend von zwei Personen geschrieben und somit auch fortlaufend während der Implementierung überprüft und beurteilt. Schwierige Problemstellungen werden gemeinsam und durch gegenseitigen Austausch oft einfacher gelöst. Pair Programming erhöht die Code Qualität. Als Einzelentwickler ist Pair Programming zwar nicht möglich, zum Zwecke des Reviews könnten aber Kollegen herangezogen werden. Gespräche und Austausch über Problemstellungen im Vorhinein können auch hilfreich sein.

Collective Ownership

Der gemeinsame Besitz und die Zuständigkeit für Quellcode ist eine wichtige Methodik des Extreme Programming. Als Einzelentwickler ist dies ohnehin gegeben. Der Nachteil als Einzelentwickler ist, dass es keine anderen Entwickler gibt, die Code überarbeiten können und so ihre Ideen, Verbesserungen und Bugfixes einbringen können.

Continuous Integration

Bei der Entwicklung in Teams arbeiten oft nicht alle an der Hauptversion eines Programmes, sondern haben eine lokale Version (z.B. bei Nutzung von Versionierungstools, wie SVN[1] oder GitHub[2]). Das regelmäßige Integrieren von neuem Code in die Hauptversion des Programmes ist wichtig, um die Konsistenz der Software zu wahren. Auch als Einzelentwickler sollte darauf geachtet werden, bei der Verwendung von Versionierungstools regelmäßig neuen Code zu integrieren. Es sollten keine neuen Aufgaben begonnen werden, solange eine alte Aufgabe noch nicht abgeschlossen ist, da auch hier sonst die Gefahr von Inkonsistenzen im Code entstehen.

Energievolles Arbeiten

Auch für Einzelentwickler gilt: Sobald man überarbeitet ist, sollte man pausieren. Die Produktivität sinkt ansonsten.

Kundenkontakt

Regelmäßiger Kontakt zum Kunden ist wichtig und sollte auch als Einzelentwickler gehalten werden. Ist dies persönlich nicht möglich, so sollte es zumindest per Telefon oder E-Mail geschehen. Wenn der Entwickler zugleich Kunde ist, kann es hilfreich sein, gewisse Dinge aufzuschreiben oder im Extremfall sogar vor sich herzusagen.

Coding Standards

Zuletzt: Auch als Einzelentwickler ist die Einhaltung gewisser Standards nötig, auch wenn man sich an seinen eigenen Stil halten kann. Letztendlich fördert die Einhaltung

[1] http://subversion.apache.org/, Dezember 2012
[2] https://github.com/, Dezember 2012

von Standards die Code - Qualität und Übersichtlichkeit.

Agarwal et al. [2] zeigen somit, dass sich die meisten Praktiken des Extreme Programming problemlos auch für den Einzelentwickler eignen, sofern kleine Anpassungen an die veränderte Situation vorgenommen werden. Auch Einzelentwickler können somit von den Vorteilen von XP profitieren.

2.4. Agile Softwareentwicklung in der Praxis

2.4.1. Kritische Betrachtung

Mittermeir et al. [20] betrachteten die Thematik der agilen Softwareentwicklung von einer eher kritischen Seite. Im Folgenden finden sich einige Aspekte der agilen Softwareentwicklung, die in der Praxis durchaus Probleme verursachen können.

Agile Methoden sind universell einsetzbar

Geht man nach Befürworten von agilen Methoden, so sind diese universell einsetzbar. *Mittermeir et al.* geben jedoch zu bedenken, dass agile Methoden auf regelmäßigem und raschem Feedback aufbauen. Problematisch wird dies, wenn Anwendungen entwickelt werden, die kein ausgeprägtes Userinterface enthalten. Feedback vom Kunden ist hier schwerer möglich. Ebenso erschwert ist der Einsatz agiler Methoden in Organisationen mit großen Entwicklerteams. Agile Methoden sind letztlich auf kleine Entwicklerteams ausgelegt (siehe *Beck* [6]).

Der Wechsel zu agilen Methoden ist einfach

Auch wenn in der Theorie die Anwendung bzw. der Wechsel zu agilen Methoden oft als einfach dargestellt wird, müssen doch einige Punkte beachtet werden, die den Wechsel von traditionellen zu agilen Methoden schwieriger machen. Der Übergang muss einerseits im Vorhinein evaluiert und auch sorgfältig geplant werden. Menschliche Faktoren, wie die Bereitschaft zu Veränderung oder die Überzeugung von der Methodik müssen auch in Betracht gezogen werden [8]. Des Weiteren kann die Einführung einer neuen Entwicklungsmethodik nicht von einem Tag auf den anderen geschehen. Es ist eher ein längerer Lernprozess, der sich – je nach Erfahrung der Entwickler im Team – erst festigen muss. Ebenso ist zu bedenken, dass die Rollenverteilung bzw. der Aufgabenbereich der Mitarbeiter oft in agilen Methoden anders aussieht, als in traditionellen Methoden.

Die Produktqualität verbessert sich immer, die Anforderungen werden korrekt erfüllt

Prinzipiell sollte sich die Produktqualität durch regelmäßiges Feedback durch den Kunden erhöhen. Dies hängt allerdings auch vom Wissensstand des Kunden ab. Ist er zum Beispiel *nur* in Vertretung mehrerer Stakeholder dem Projekt beigestellt, kann es durchaus sein, dass sich durch Wissenslücken seinerseits Abweichungen von den eigentlichen

Anforderungen ergeben. Ebenso gibt es gewisse Anforderungen, die nicht ohne Weiteres im Nachhinein in bestehende Software eingebaut werden können, zum Beispiel die Verbesserung der Gesamtperformanz oder der Usability eines Systems.

Eine schnellere Entwicklungszeit wird erreicht

Prinzipiell soll mit agilen Methoden auch eine schnellere *time to market* erreicht werden, doch auch hier müssen einige Aspekte beachtet werden. Oft fordern Kunden fixe Vertragsbedingungen (Laufzeit, Preis etc.). Durch die iterative Entwicklung ist es oftmals nicht ganz einfach die Projektlaufzeit auf Langzeit genau abzuschätzen bzw. vorherzusagen. Dies kann Probleme mit der Budgetplanung ergeben. Genauso kann es problematisch sein, regelmäßig kleine Releases real auszuliefern. Beispielsweise wenn es viel Zeit in Anspruch nimmt, diese beim Kunden zu installieren. Dies kann wiederum das Feedback vom Kunden verzögern.

Probleme mit zusätzlichen und neuen Anforderungen

Durch die grundsätzliche Möglichkeit von zusätzlichen und veränderlichen Anforderungen, muss darauf geachtet werden, trotz allem im Vorhinein eine gute Anforderungsanalyse durchzuführen. Andernfalls kann es sehr aufwändig (bzw. teuer) werden, neue Anforderungen, die starkes Refactoring nötig machen, umzusetzen.

Agile Methoden benötigen kein Design

Es ist oftmals nicht korrekt, davon auszugehen, dass jegliches Design innerhalb gewisser Stories vorgenommen werden kann. Gerade bei sehr großen Systemen ist es oft auch nötig, gewisse Architektur-Entscheidungen im Vorhinein zu treffen bzw. umzusetzen (z.B. Design von Datenbankschemen etc.).

Mittermeir et al. [20] zeigen zumindest einige Probleme auf, zu denen es bei der Umsetzung von agilen Methoden kommen kann. Ihre Grundaussage ist, dass agile Methoden nicht immer ohne Anpassungen an die realen Gegebenheiten - sozusagen *blind* - nach theoretischen Vorgaben umgesetzt werden können.

2.4.2. Überblick über die Auswirkungen von Agilen Methoden auf die Softwareentwicklung für Forschungssoftware

Einen Überblick darüber, wie sich der Einsatz von Methoden der agilen Softwareentwicklung auswirkt, geben *Sletholt et al.* [36]. Sie führten 2011 eine Literaturrecherche durch und analysierten dabei verschiedene Projekte, bei denen Software für wissenschaftliche Zwecke mit der Hilfe von agilen Methoden entwickelt wurde. Software, die im wissenschaftlichen Kontext entwickelt wird, unterscheidet sich oft in gewissen Punkten von Business-Software, Administrativen Tools etc. Oftmals werden komplexe Berechnungen oder Simulationen durchgeführt. Die Software kann auch dazu dienen, wissenschaftliche Theorien zu beweisen. Nicht selten ist der korrekte Output eines Programmes im

Vorhinein gar nicht bekannt. Außerdem entsteht diese Art von Software oft durch jahrelange aufeinander aufbauende Arbeit verschiedener Forscher. Im Zusammenhang mit Softwareentwicklung ergeben sich dadurch gewisse Herausforderungen, z.B. bei der Anforderungsanalyse und Spezifikation. Anforderungen können sehr dynamisch sein, sich also schnell verändern. Genauso können gewissen Anforderungen im Vorhinein noch gar nicht klar oder bekannt sein und erst mit der Zeit relevant werden. Diese Eigenschaften legen ein entsprechendes Anforderungs - und Testmanagement nahe. Viele der Praktiken der agilen Softwareentwicklung sind genau dabei hilfreich.

Es wurden schließlich zehn Projekte analysiert. In den Projekten wurden viele der Praktiken der agilen Softwareentwicklung eingesetzt. Allerdings ergaben sich oftmals auch Einschränkungen. Beispielsweise war die Teamgröße von zwei Personen in einem Projekt nicht ideal. Es zeigte sich auch das Problem, dass ein expliziter Kunde in gewissen Projekten fehlte, weshalb die Entwickler dementsprechend selbst als Stakeholder fungieren mussten. Allerdings kam man bei allen analysierten Projekten zum Schluss, dass sich die Einführung der agilen Softwareentwicklung durchwegs positiv ausgewirkt hat. In fast allen Projekten machte sich die schnelle Reaktionsfähigkeit auf sich ändernde Anforderungen bemerkbar. Außerdem stieg bei vielen die Übersichtlichkeit bzw. die gesamte Code-Qualität.

Zuletzt muss erwähnt werden, dass bei den analysierten Projekten oft nicht bekannt war, wie detailliert und konsequent die agilen Praktiken angewendet wurden und wie mit eventuellen Problemen bei der Umsetzung umgegangen wurde. Der folgende Abschnitt beschäftigt sich nun genau mit diesem Thema: Wie kann man agile Methoden an Bedingungen, die nicht ideal sind, anpassen?

2.4.3. Anpassungen an reale Gegebenheiten

Viele agile Methoden der Softwareentwicklung, wie XP oder Scrum, sehen gewisse Verfahren und Maßnahmen vor. Streng genommen sollten dann z.B. bei der Anwendung von XP auch alle Praktiken und Vorgehensweisen genau umgesetzt werden. Laut *Hoda et al.* [21] sehen es zumindest viele Urheber der Methoden oder Autoren zum Thema so (es gibt beispielsweise auch den Scrum-Butt-Test[3] - ein Test der Auskunft darüber gibt, wie Scrum-konform man entwickelt)

Wenn der Kontext stimmt, lassen sich die Methoden zumeist auch umsetzen. *Hoda et al.* gingen in einer Studie mit 40 Projekten der Frage nach, wie agile Methoden in der Praxis angepasst werden können bzw. müssen, wenn die Bedingungen nicht ideal sind. Die häufigsten Probleme und gängige Lösungsstrategien dazu werden im Folgenden beschrieben.

Kundenkontakt In sehr vielen Projekten war es unmöglich, den Kunden regelmäßig und intensiv in die Entwicklung einzubinden. In vielen agilen Methoden ist aber genau dies ein sehr wichtiger Aspekt. Letztendlich hat fehlende Kundeneinbindung zur Folge, dass die Entwickler weder mit sich ändernden Anforderungen am laufenden ge-

[3] http://scrumbutt.me/, Jänner 2013

halten werden, noch genügend Feedback für Umsetzungen der Anforderungen erhalten. Dies kann im schlimmsten Fall zu zeitweisem Stillstand eines Projektes führen oder zumindest zu verminderter Produktivität. In einigen der untersuchten Projekten wurden dahingehend Anpassungen vorgenommen, sodass es z.B. für jede Story einen speziellen Zuständigen (Story-Owner) aus einer auftraggebenden Firma gab. Nur wenn dieser verfügbar war, wurde an der entsprechenden Story gearbeitet. Dies steht im Gegensatz zur Scrum-Praktitk des Product-Owners (einer Person, die stellvertretend für alle Stakeholder dem Projekt beiwohnt [35]) oder des Kunden im XP. Eine andere Lösung, die für das Problem des fehlenden Kundenkontaktes gefunden wurde, war die des *Customer Proxies*. Eine Person wurde dabei als Customer Proxy, also als Mittelsmann des Kunden, abgestellt. Dieser übernahm den Kundenkontakt und regelt den Informationsaustausch bezüglich Anforderungen und Feedback.

Das fixe Angebot Ein anderes Problem, das bei einigen Entwicklerfirmen auftrat, war, dass Kunden auf die Einhaltung fester Vertragsbedingungen bestanden (fixe Bezahlung, fixer Zeitrahmen, fixe Anforderungen etc.). Auch dies widerspricht eigentlich Prinzipien der agilen Softwareentwicklung (vgl. Abschnitt 2.1.1 - Agiles Manifesto: "Zusammenarbeit mit dem Kunden geht vor Vertragsverhandlungen"). Agile Entwicklungsmethoden sind gerade auf veränderliche Anforderungen ausgelegt. Aber auch zu dieser Problematik wurden von Entwicklerfirmen Lösungswege gefunden. So bot eine Firma ihren Kunden anstelle eines Langzeitprojektes nur eine gewisse Anzahl von Projekt - Iterationen an. Dies sollte vor allem Vertrauen in die Methodik schaffen und Kunden von Konzepten der agilen Softwareentwicklung überzeugen. Bei Gefallen (bzw. Erfolg) konnten natürlich weitere Iterationen vertraglich vereinbart werden. Eine andere Möglichkeit die gefunden wurde, war einfaches hinzufügen eines Spielraumes zum Projektzeitraum. Basis für die Abschätzung der eigentlichen Entwicklungszeit und des Spielraumes war die Projektgeschwindigkeit laut der verwendeten agilen Methodik. Durch diesen Spielraum wurde erhöhter Zeitanspruch, z.B. durch veränderte oder neue Anforderungen von Kundenseite, ausgeglichen.

Hoher Designaufwand Hoher Aufwand für Design lässt sich oft nicht mit schnellen und kurzen Iterationen vereinen. Hier wurden z.B. Lösungen wie die folgende eingesetzt: Jede Iteration, die die Entwicklung von Backend-Code betrifft, startet erst, wenn der entsprechende Frontend-Teil (GUI-Implementierung) fertiggestellt ist.

Hoher Dokumentationsaufwand Wenngleich laut agilem Manifest der Code und Tests als Dokumentation prinzipiell ausreichend sind, ist dies doch nur eine einfache Form von Dokumentation. Dies reicht für manche Projekte nicht aus. Auch hierzu wurden Lösungen gefunden. Beispielsweise gab es bei einer Entwicklerfirma die Vorgehensweise, bei der parallel zur entwickelten Applikation ein online editierbares Dokument vom Kunden gepflegt wurde. Die Kunden befüllten es mit gewissen, geschäftsrelevanten Ausdrücken sowie mit deren Erklärungen und Zusammenhängen. Die Entwickler setzten dies dann 1:1 in Sourcecode um. Sie wählten beispielsweise die Variablennamen im Quellco-

de entsprechend den Ausdrücken im Online-Dokument und setzten die Erklärungen und Zusammenhänge, wie sie im Online-Dokument beschrieben waren, logisch um.

Langsame Veränderung Agile Softwareentwicklung ist eigentlich darauf ausgelegt, schnell auf sich ändernde Anforderungen zu reagieren und flexibel zu sein. Es gab jedoch Projekte, an die von Anfang an sehr stabile Anforderungen gestellt wurden. Die Kunden hatten also schon sehr klare Vorstellungen. Reagieren auf veränderliche Anforderungen war also von vornherein nicht nötig. Das Problem wurde von den betroffenen Teams, die im Stile des XPs entwickelten, wie folgt gelöst: In einem initialen Planing Game wurden vom Kunden die Anforderungen vermittelt. Diese wurden dann auf Story Cards festgehalten. Durch die Kunden wurde die Priorität festgelegt und die Story Cards wurden der Reihe nach abgearbeitet. So ergab sich also eine leicht verminderte Version des XP, bei der es schlicht und einfach zu keinen großen Veränderungen der Anforderungen - und somit veränderten Prioritäten - kam.

Räumliche Abstände Bei einigen Teams waren manche Praktiken der agilen Softwareentwicklung nicht umsetzbar, da die Teammitglieder oft über mehrere Standorte (unterschiedliche Städte/Länder) verteilt waren. Man umging diese Probleme so gut wie möglich mit elektronischen Hilfsmitteln: Video - und Audiokonferenzen, E-Mail, Chat oder auch virtuellen Storyboards.

Fazit *Hoda et al.* [21] kamen zum selben Schluss, wie *Mittermeir et al.* (siehe Abschnitt 2.4.1). Die vollkommene Anwendung von agilen Methoden ist nur dann sinnvoll und praktikabel, wenn die Gegebenheiten stimmen. Auch wenn sich einige Praktiken problemlos umsetzen lassen, gibt es oftmals Probleme bei anderen. Die Situation ist ähnlich, wie bei der Anpassung von Extreme Programming (siehe Abschnitt 2.3.4) an Einzelentwickler. Viele der untersuchten Projekte von *Honda et al.* standen mit der Umsetzung ihrer Entwicklungsmethoden vor einem ähnlichen Problem, wie es die agile Softwareentwicklung selbst zu lösen versucht: Das Reagieren auf Veränderungen.

3. Testen von Software mit besonderem Augenmerk auf Grafische User Interfaces

Ganz im Allgemeinen ist der Zweck von Softwaretests, herauszufinden ob Programmcode Fehler enthält oder nicht. Ein Test läuft demnach nur dann erfolgreich durch, wenn der Programmcode keine Fehler enthält. Testen nimmt somit bei der Entwicklung von Programmcode neben den Disziplinen der Anforderungsanalyse, Spezifikation und Implementierungen, eine entscheidende Rolle ein. Es geht vor allem darum, die Korrektheit eines Programmes zu belegen. Im Laufe der Zeit haben sich gewisse Eigenschaften von Tests herauskristallisiert, welche entscheidend dafür sind, wie sinnvoll Tests im eigentlichen sind: Verlässlichkeit, Richtigkeit und vor allem Vollständigkeit (Testabdeckung). Ein Programm kann sich in vielen verschiedenen Zuständen befinden. Tests sollen diese Zustände und auch möglichst viele verschiedene Kombinationen daraus auf ihre korrekte Funktionsfähigkeit hin prüfen. [18]

"Durch Testen kann nur die Anwesenheit, jedoch nicht die Abwesenheit von Fehlern gezeigt werden"
- Edsger W. Dijkstra [11]

3.1. Testkriterien und Testabdeckung

Viele verschiedene Methoden zum Testen von Software haben sich Schritt für Schritt über die Zeit entwickelt. Schon in den 70ern beschäftigten sich *Goodenough und Gerhart* [18] mit der von Dijkstra getätigten Aussage zum Thema Anwesenheit und Abwesenheit von Tests. Sie versuchten zu zeigen, dass eine gut strukturierte Menge von Tests ausreichend sei, um die Abwesenheit von Fehlern in Software zu belegen. Die zentrale und entscheidende Frage, die diesbezüglich gestellt werden muss: Was sind (adäquate) Testkriterien für ein Test-Set? (Eine Menge von Test-Cases wird zu einem sogenannten Test-Set oder zu einer Test-Suite zusammengefasst). Ein Testkriterium beschreibt letztlich, welche inhaltlichen Aspekte von Sourcecode getestet werden sollen. Im Idealfall sollten diese Kriterien aus den Spezifikationen und Anforderungen für ein Programm hervorgehen. *Goodenough und Gerhart* heben zudem noch hervor, dass Testkriterien gewisse Eigenschaften erfüllen müssen - Zuverlässigkeit (engl. Reliability) und Gültigkeit (engl. Validity). Zuverlässigkeit bedeutet, dass ein Testkriterium konstante Ergebnisse liefern muss. Wenn also ein Programm mit einem Test-Set, welches ein Testkriterium erfüllt,

erfolgreich getestet wird, so müssen die Tests auch mit einem anderen Test-Set, welches das Kriterium erfüllt, erfolgreich sein. Die Zuverlässigkeit bezieht sich darauf, dass Tests immer aussagekräftige Ergebnisse geben. Gibt es einen Fehler im Programm, so muss er von einem Test im Test-Set, welches einen Testkriterium erfüllt, gefunden werden. In der Vergangenheit zeigte sich allerdings, dass diese Vorgaben von sehr theoretischer Natur sind und in der Praxis schwer zu erreichen sind.

Eng damit zusammen hängt auch die sogenannte *Testabdeckung*, welche beschreibt, in welchem Ausmaß eine Menge von Tests den zugrundeliegenden Code auf gewisse Kriterien hin testet. Es wurden viele verschiedene Testkriterien gefunden, nach denen die Testabdeckung gemessen werden kann. Es folgen einige Unterscheidungen für Testkriterien nach *Zhu et al.* [39]:

- **Statement Coverage** Die Statement Coverage gibt an, wie viele Statements des zu testenden Codes tatsächlich getestet werden. Ein Test-Set, bei dessen Ausführung jedes Statement im Programmcode zumindest einmal ausgeführt wird, erfüllt demnach das Testkriterium der Statement Coverage. Im Allgemeinen wird die Statement Coverage auch als Prozentsatz angegeben und beschreibt dadurch den Anteil der durch ein Test-Set getesteten Statements am gesamten Programmcode.

- **Branch Coverage** Ähnlich wie bei der Statement Coverage bezieht sich die Branch Coverage auf den Anteil der ausgeführten Verzweigungen während der Ausführung einer Test-Suite.

- **Path Coverage** Das Testkriterium der Path Coverage ist genau dann erfüllt, wenn alle möglichen Ausführungspfade eines Programmes durch ein Test-Set von Eintritts- bis zum Endpunkt ausgeführt wurden.

- **Mutationserkennung** Bei diesem Testkriterium geht es darum, veränderten Programmcode zu erkennen. Dabei wird absichtlich ein Fehler eingeführt bzw. eine Veränderung am Programmcode vorgenommen. Dann wird getestet, ob zumindest einer der Tests im Test-Set durch diese mutwillige Veränderung fehlschlägt. Eine Programmversion, die solch eine Veränderung enthält, wird Mutation (Mutant) genannt. Werden mehrere solcher Mutationen eines Programmes erstellt und getestet, kann man so aus dem Verhältnis der Gesamtanzahl von Mutationen und der Anzahl von entdeckten Mutationen die sogenannte *mutation score* errechnen. Auch diese kann als Maß für die Testabdeckung bzw. als Testkriterium verwendet werden.

Wie schon erwähnt, sind die Zuverlässigkeit und Gültigkeit theoretische Grenzen für Testkriterien. In der Praxis wird meist versucht, mit einem bestehenden Test-Set zumindest eine (prozentuell) recht hohe Erfüllung der Testkriterien zu erreichen. Einschränkend wirken in realen Projekten oft auch mangelndes Geld- und Zeitbudget zum Testen, wobei neuartige Ansätze zum Umgang mit Tests (siehe Abschnitt 3.2 über Test Driven Development) dem entgegenwirken sollen.

Es existieren nach *Zhu et al.* außerdem noch andere Ansätze, um Testkriterien voneinander zu unterscheiden:

- **Basierend auf Spezifikation** Als Testkriterium wird hier die Erfüllung von Anforderungen und Spezifikation einer Anwendung hergenommen. Sobald durch ein Test-Set belegt ist, dass die gestellten Anforderungen an eine Anwendung erfüllt sind (d.h. geforderte Features erfolgreich getestet wurden), ist das Testkriterium erfüllt
- **Basierend auf dem Programm** Das Programm wird als Referenz angesehen. Alle Funktionen eines bestehenden Programmes müssen getestet werden, um das Testkriterium zu erfüllen.
- **Kombination** In der Praxis wird oftmals eine Kombination aus den beiden oben genannten Testkriterien verwendet

Testkriterien können nach *Zhu et al.* weiters anhand der folgenden Punkte unterschieden werden:

- **Strukturelles Testen** - Das Testen auf die Erfüllung gewisser Punkte aus der Spezifikation eines Programmes (siehe oben)
- **Defekte suchen** - Testkriterium ist, dass durch ein Test-Set fehlerhaftes Verhalten, wie z.B. ein Absturz, gefunden wird
- **Fehler suchen** - Testkriterium ist, dass durch ein Test-Set falsche Ausgaben, Berechnungen, Darstellungen gefunden werden

Neben den genannten Unterscheidungen für Testkriterien nach *Zhu et al.*, existieren je nach Sichtweise noch weitere. Im Abschnitt über GUI - Testing wird speziell auf Testkriterien und Testabdeckung bzgl. Graphischen User Interfaces eingegangen.

Wie schon erwähnt, ist eine volle Abdeckung von Testkriterien nur theoretische erreichbar, da für viele Programme die Anzahl der möglichen Inputs praktisch unendlich ist. Deshalb gibt es Ansätze zur Filterung und Priorisierung von Testfällen. Durch geschickte Auswahl und Kombination von Testfällen sollen einerseits Testkriterien zu möglichst hohen Prozentsätzen erfüllt werden, aber auch Fehler möglichst schnell entdeckt werden. [26]

Auch bei der Umsetzung des Praxisprojektes im Rahmen dieser Arbeit wurde versucht, eine möglichst hohe Testabdeckung zu erreichen, wobei sich dies vor allem beim Testen des Graphischen User Interfaces als sehr schwierig herausstellte. Diese Thematik wird in späteren Abschnitten näher behandelt.

3.2. Test Driven Development

Test Driven Development (TDD) ist ein Verfahren in der Softwareentwicklung, in welchem ein System durch Tests spezifiziert wird. Die Tests werden vor dem eigentlichen Sourcecode geschrieben. Das Verfahren läuft in simplen, kurzen Zyklen ab, die sich durchgehend wiederholen. Jeder der kurzen Zyklen (Sekunden bis Minuten) besteht aus folgenden Schritten:

- Code wird in Form eines Unit Tests spezifiziert
- Erste Ausführung des Tests schlägt fehl
- Der dazugehörige Code wird geschrieben
- Der Test läuft (im Idealfall) erfolgreich durch
- Gegebenenfalls Verbesserung und Überarbeitung (Refactoring) des Codes

Alle Tests werden regelmäßig nach Veränderung oder Hinzufügen von Code ausgeführt. Neuer Code darf nicht dazu führen, dass irgendein Test im System fehl schlägt. TDD wirkt sich in Form der folgenden Aspekte positiv auf ein System aus:

- **Qualität** - Da der gesamte Code des Systems ständig getestet wird, verringert sich die Anzahl der Fehler im kompletten System. Zusätzlich führt TDD zu einem guten, entkoppelten Design. Eine positive Eigenschaft davon ist vor allem, dass Klassen nicht so stark voneinander abhängig sind.
- **Dokumentation** - Jeder Unit Test ist eine Dokumentation einer Klasse in Codeform
- **Flexibilität** - Die Codequalität kann regelmäßig durch kleine Änderungen verbessert werden. Die Unit Test bewahren einen davor, dadurch unbeabsichtigt neue Fehler ins System einzubringen.
- **Schnelles Feedback** - Durch die kurzen Entwicklungszyklen erhält man als Programmierer sehr schnell Feedback über Änderungen im Code oder aber auch des Tests.

TDD trägt also durchgehend positiv zur Erhöhung von Qualität in allen Sektoren der Softwareentwicklung bei. [24]

3.2.1. Ziele des TDD

Letztendlich ist das Ziel des Einsatzes von testgetriebener Entwicklung sauberer und gut wartbarer Code. Dieser hat sowohl während der Entwicklung, als auch später bei der Wartung von Software entscheidende Vorteile. Sauberer Code macht Programmierung zu einer berechenbaren (vorhersehbaren) Tätigkeit. Es lässt sich leicht erkennen, wann eine Aufgabe fertig implementiert ist. Eine langwierige Fehlersuche wird ebenso erspart. Sauberer Code entsteht nicht durch einmaliges Hinschreiben einer Implementierung. Er entsteht zumeist durch mehrmaliges Refactoring bzw. durch das Überdenken und Verbessern von Code. Durch die mehrmalige, intensive Beschäftigung mit einer Problemstellung, erhält man optimale Ergebnisse für ein Problem. Für Entwickler, die mit einem zusammenarbeiten, ist sauberer Code besser verständlich, nachvollziehbar und lässt sich ebenso einfacher verändern und anpassen. Sauberer Code führt letztendlich auch zu zufriedenen Kunden. [4]

Die Qualität von Software, welche mittels TDD entwickelt wurde, steigt auch deshalb, weil gut implementierte Tests die Validierung von Software erleichtern. Tests die vor dem eigentlichen Code geschrieben werden, können als eine Art Spezifikation und Dokumentation von Software (bzw. einer Softwarekomponente) angesehen werden. Zum Einen werden durch die Implementierung eines Tests Vorbedingungen gezeigt, die für die Verwendung einer Komponente nötig sind, andererseits werden auch die entsprechenden Nachbedingungen nach Ausführung aufgezeigt. Gewissermaßen bescheinigt das Durchlaufen aller Tests, dass ein Programm seine Spezifikationen erfüllt. Dies setzt jedoch auch entsprechend gut umgesetzte Tests voraus. Wie schon im Abschnitt über agile Softwareentwicklung beschrieben, ist es deshalb nötig, Programmcode und Tests regelmäßig anzupassen (Refactoring). [34], [4]

3.2.2. Unit Tests

Ein wichtiger Begriff im Zusammenhang mit Testen von Software ist der des *Unit Tests*. Der Begriff stammt eigentlich noch aus der Zeit bevor objektorientierte Programmierung üblich war. Er bezieht sich darauf, dass nicht das ganze Programm im Fokus des Testens steht, sondern nur ein kleiner Teil davon - *eine Unit*. Früher meinte man mit Unit meist eine *Prozedur*. Unter dem Paradigma der Objektorientierung reicht die Spannweite des Begriffs jedoch von einer *Methode* über *ganzen Klassen* bis hin zu *Subsystemen* oder ganzen *Systemen*. [25]

3.2.3. JUnit - Umsetzung in Java

Test Driven Development mit Hilfe von Unit Tests lässt sich für die Implementierung von Programmlogik relativ unkompliziert umsetzen. Es gibt für die meisten Programmiersprachen eigene Testumgebungen oder Frameworks, die Entwickler dabei unterstützen. Ein bekanntes Framework ist JUnit[1] für die Programmiersprache Java[2], mit dem Unit Tests umgesetzt werden können. Es besteht lediglich aus einer JAR-Datei, welche in ein Java-Projekt eingebunden werden muss. Mit JUnit können Testklassen geschrieben werden, welche Testmethoden beinhalten, die gewisse Komponenten in Java-Klassen testen. Hilfs-Methoden wie setUp() und tearDown() werden einmal automatisch vor und nach der Abarbeitung der eigentlichen Tests aufgerufen und können zum Beispiel dazu dienen, andere Programmkomponenten, die innerhalb von Tests verwendet werden, zu initialisieren. Schließlich können auch beliebig viele Test-Methoden geschrieben werden, welche die nötige Testlogik enthalten. Eine Test-Klasse dient üblicherweise dazu, eine Java-Klasse zu testen. Für Methoden einer Klasse werden die entsprechenden Testmethoden in der JUnit-Klasse umgesetzt. Das Framework stellt außerdem Methoden wie assert() oder assertEquals() und assertNotNull() zur Verfügung, welche zur Auswertung von Nachbedingungen verwendet werden können. So kann etwa getestet werden, ob das Ergebnis nach Aufruf einer Methode mit gewissen Parametern den richtigen Rückgabewert liefert. Mehrere Testklassen können schließlich in Test Suits zusammengefasst werden.

[1] www.junit.org, Jänner 2013
[2] http://www.java.com/, Jänner 2013

JUnit bietet sowohl eine Konsolenausgabe an, sowie eine grafische Benutzeroberfläche. Dadurch können Test und Test Suits sehr effizient und schnell gemeinsam ausgeführt werden. Die Ergebnisse werden übersichtlich aufbereitet. Laufen alle Tests erfolgreich durch, wird dies durch einen grünen Balken signalisiert, treten Fehler auf, ist der Balken rot. Gescheiterte Tests werden zusätzlich markiert. JUnit wurde von *Beck* und *Gamma* geschrieben. Es geht eigentlich auf ein ähnliches Framework zurück, welches für die Programmiersprache Smalltalk entwickelt wurde (sUnit). [25], [34]

Für die Entwicklungsumgebung Eclipse[3] gibt es zusätzlich ein Plugin, welches die Testausführung unterstützt (z.B. durch grafische Komponenten, die über Testergebnisse informieren - siehe Abbildung 3.1).

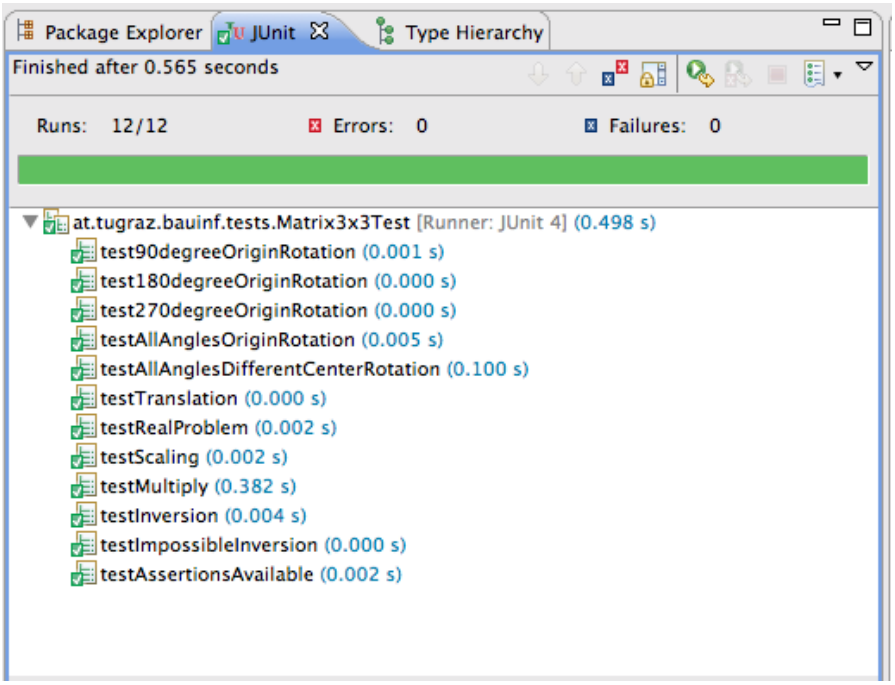

Abbildung 3.1.: *JUnit* - Fenster in der Entwicklungsumgebung Eclipse

3.3. GUI Testing

Grafische User Interfaces (GUI) sind in der heutigen Zeit allgegenwärtig, sei es in Form von Dialogen und Fenstern, oder aber auch in Form von Webinterfaces. Das GUI macht dabei oft einen sehr großen Teil einer Applikation aus. Oftmals sind zwischen 45% und 60% des Quellcodes GUI-bezogen [31]. Es ist der Teil der Software, mit dem Benutzer direkt in Kontakt kommen. Zudem haben grafische Komponenten oft andere Merkmale

[3] http://www.eclipse.org/, Jänner 2013

und Eigenheiten, als konventioneller Code (Hintergrund-Logik). Dies macht die Entwicklung und Gestaltung grafischer Oberflächen zu einer großen Herausforderung. Oft werden User Interfaces mittels *Rapid Prototyping* entwickelt. Dabei werden fortwährend Prototypen eines Systems entwickelt bzw. weiterentwickelt, welche jedoch noch nicht voll funktionsfähig sind, sondern zu Demonstrationszwecken von Funktionen bzw. um schnell Feedback zu erhalten, geschaffen werden. Folglich ergeben sich auch beim Testen von grafischen Oberflächen ganz eigene Probleme und Herausforderungen für Entwickler. [29], [33], [30]

3.3.1. Herausforderungen

Grafische User Interfaces haben gewisse Eigenheiten. Sie kommunizieren mittels Methoden-Aufrufen und Nachrichten mit darunterliegender Hintergrund-Logik. Da GUI-Code oft einen hohen Anteil des Quellcodes ausmacht, ist es folglich nötig, auch diesen Code gewissenhaft zu testen, gerade weil das User Interface auch entscheidenden Einfluss auf die Sicherheit, Usability und Robustheit des gesamten Systems hat. Entscheidende Problemstellungen beim Testen von User Interfaces sind u.a.: [28]

- **Wahl von entsprechenden Tools und Verfahren** - In den folgenden Abschnitten werden verschiedene Techniken genannt. Die Spannweite geht dabei von der Verwendung einzelner Test-Tools, über formale Verfahren, bis hin zur Veröffentlichung von Beta Versionen, um die Benutzer Tests durchführen zu lassen.
- **Wahl des passenden Testorakels** - Testorakel kommen zum Einsatz, um zu verifizieren, ob Tests korrekt ausgeführt wurden. Sie werden im folgenden Abschnitt noch näher beschrieben.
- **Finden und erreichen der optimalen Testabdeckung** - Es geht hierbei um die Frage, wann ein User Interface entsprechend gut und ausreichend getestet wurde. Die große Menge an Kombinationen von verschiedenen Aktionen auf dem User Interface erschwert das Erreichen einer hohen Testabdeckung.
- **Regressionstests** Es geht vor allem darum, wie sich Automatisierung von Tests umsetzen lässt und wie man auf Weiterentwicklung und Veränderung von User Interfaces reagiert. Die Automatisierung von Tests ist außerdem auch bei Verwendung von agilen Methoden relevant

Letztendlich strebt man beim Testen von User Interfaces ebenfalls nach Methoden, die möglichst allgemein einsetzbar und für eine große und breit gefächerte Anzahl von User Interfaces umsetzbar sind. [28]

Im Folgenden wird nun auf wichtige Aspekte beim Testen von User Interfaces detailliert eingegangen.

Testorakel

Xie und Memon [38] bringen sogenannte Testorakel in Zusammenhang mit GUI-Testing. Ein Testorakel bestimmt, ob Software für einen Test-Case korrekt ausgeführt wird. Es beeinflusst dadurch auch maßgeblich, wie effektiv ein Test-Case Fehler erkennen kann bzw. wie hoch der Aufwand für einzelne Test-Cases ist. Es wird also ein Set von Tests erstellt, sodass die gewünschten Testkriterien erfüllt werden und die geforderte Testabdeckung erreicht wird (siehe 3.1). Das Testorakel hilft schließlich dabei, zu bestimmen, ob die getestete Applikation auch richtig funktioniert.

Praktische Ansätze Es gibt verschiedene Ansätze dazu, wie ein Testorakel aussehen kann. Im Wesentlichen sehen die vier bekanntesten und gebräuchlichsten wie folgt aus: [38]

- **Manuelles Testen** Das einfachste und doch an Ressourcen sehr aufwändige Testorakel, ist das manuelle Testen. Dabei werden von einem Tester gewisse Aktionen auf dem Userinterface ausgelöst (Events) bzw. gewisse Eingaben getätigt. Die Ergebnisse werden rein visuell überprüft.

- **Capture/Replay Tools** Auch Capture/Replay Tools erfordern die manuelle Bedienung eines Userinterfaces. Im ersten Schritt bei der Nutzung von Capture/Replay Tools werden alle Aktionen eines Testers (also ausgelöste Events, Eingaben etc.) aufgezeichnet. Der Tester überprüft wiederum auf visuellem Wege, ob das Programm korrekt funktioniert. Die entsprechenden (korrekten) GUI-Zustände werden vom Tool gespeichert. Im nächsten Schritt kann das Caputer/Replay Tool die aufgezeichneten Aktionen automatisch wieder abspielen. Dies ist nun auch auf einer veränderten (bzw. weiterentwickelten) Version des Userinterfaces möglich. Der vorher gespeicherte GUI-Status nach jeder Aktion kann nun mit dem aktuellem Status des GUI während der Wiedergabe verglichen werden. Somit kann die korrekte Funktionalität eines GUI bestimmt werden.

- **Programmierte Test-Cases** Eine weitere Möglichkeit ist die Verwendung von programmierten GUI-Test-Cases. Ein Tester programmiert diese Test-Cases und gibt zeitgleich den gewünschten Output (GUI-Status nach Ausführung) an. Spezielle Programme können die Test-Cases im Anschluss ausführen und die Ergebnisse auswerten. Es gibt Frameworks, die einem Tester dabei helfen, dies umzusetzen. Sie bieten beispielsweise Funktionalitäten zum Auffinden von GUI-Elementen oder zum Auslesen ihrer Eigenschaften an (siehe Abschnitt 3.3.2 über verschiedene Tools).

- **Testen in der Business Logik** Es besteht auch die Möglichkeit, dass ein Tester gewissermaßen die Aufrufe, die bei Aktionen auf dem GUI ausgelöst werden, simuliert. Das heißt zum Beispiel, dass Methoden, die eigentlich durch ein spezielles Event ausgelöst werden, manuell aufgerufen werden. Dementsprechend muss allerdings auch die Architektur der Software aufgebaut sein, um dies zu ermöglichen.

Jegliche Programmlogik muss vom eigentlichen Userinterface entkoppelt sein. Die simulierten Aufrufe müssen genau so erfolgen, als würden sie vom Benutzer durch GUI-Events ausgelöst werden.

Letztendlich lassen sich auch Kombinationen der erwähnten Testorakel in der Praxis einsetzen. Sie haben alle gemeinsam, dass sie teils recht aufwendig sind und oftmals erheblicher Aufwand bei der Änderung der Benutzeroberfläche entsteht. [38]

Formale Ansätze Wie schon erwähnt, gibt es Unterschiede zwischen GUI-Code und Hintergrund-Logik. Im Konkreten ist vor allem der Input bzw. Output für einen GUI-Test anders, als bei konventionellen Tests von Hintergrund-Logik. User Interfaces sind Event-basierte Systeme, deren Input eine Sequenz von Events und deren Output ein spezieller GUI-Status ist (nimmt man es ganz genau, so beeinflusst jedes Event den GUI-Status auf eigene Weise - eine Kombination bzw. Hintereinanderausführung von Events ergibt ebenfalls wieder einen gewissen GUI-Status - darüber hinaus beeinflussen sich manche Events auch gegenseitig). *Memon et al.* [27] stellen in Bezug auf diese Problematik ein weiteres und komplett anderes Testorakel vor, das auf formaler Logik aufbaut. Sie versuchten, ein automatisches Testorakel zu entwickeln. Ihre Anforderungen daran waren, dass einerseits das Verhalten eines GUI modelliert werden kann, sodass der Status des GUIs nach Ausführung eines Tests automatisch ermittelt werden kann. Dazu muss es auch eine formale Repräsentation von GUI-Elementen und Aktionen geben. Ebenso muss es eine Repräsentationsform für den aktuellen GUI-Status geben, um diesen mit erwartetem Verhalten (bzw. Status) zu vergleichen. Auch dieser Ansatz ist bei der Erstellung von Test-Cases recht aufwändig. Der gesamte Zeitaufwand für die Umsetzung dieses Testorakels wird auf Basis eines Tests von den Autoren aber als gering beschrieben. Sie zeigten mit ihrem Projekt, dass also auch ein formales Testorakel durchaus in der Praxis verwendbar ist. [27]

Testabdeckung und Kriterien für GUIs

Auch im Bezug auf das Testen von grafischen User Interfaces stellt sich die Frage, ob mit den vorhandenen Tests auch eine entsprechende Testabdeckung erreicht wird. Die bekannten Kriterien der Statement-, Branch- und Pathcoverage (siehe 3.1) sind jedoch nicht adäquat für das Testen von User Interfaces oder zumindest nicht immer zu 100% einsetzbar. Gründe dafür sind unter anderem, dass bei der Entwicklung von User Interfaces bereits vorkompilierte Komponenten (GUI-Elemente wie Buttons, Labels etc.) verwendet werden. Der Quellcode dieser Elemente ist meist nicht verfügbar. Weiters bestehen Eingaben auf User Interfaces aus Abfolgen von Events. Die Anzahl an Permutationen solcher Events kann bei größeren Oberflächen sehr hoch werden. Hinzu kommen die verschiedenen Status, in denen sich GUI Elemente befinden können. Genau aus diesen Gründen schlagen *Memon et al.* [30] eine neue Art von Coverage Kriterien, basierend auf Events, vor. Da die Menge der Permutationen von Eingabe-Sequenzen so hoch ist, wird das User Interface in Subkomponenten aufgeteilt, die alle einzeln getestet werden und eine Unit im Sinne eines Unit Tests darstellen. Somit kann eine Subkomponente isoliert getestet

werden. Schließlich unterscheiden sie im Groben zwischen Intra-Komponent-Kriterien, und Inter-Komponent-Kriterien. Intra-Komponent-Kriterien beziehen sich auf die Testabdeckung bzgl. Events innerhalb einer Komponente. Es geht also um die Frage, wie viele Events oder Event-Sequenzen einer gewissen Länge innerhalb einer Komponente durch Tests abgedeckt werden. Die Inter-Komponent-Kriterien beziehen sich schließlich darauf, ob bzw. in welchem Ausmaß alle Interaktion zwischen den Subkomponenten getestet werden. Damit zeigen *Memon et al.* eine mögliche Art auf, auf veränderte Anforderungen bei der Ermittlung der Testabdeckung von GUI-Tests einzugehen.

Spezielle Herausforderungen in der Agilen Softwarentwicklung

Gerade bei der Verwendung agiler Methoden kommt es zu einer weiteren Herausforderung für Entwickler, die mit den Eigenheiten einiger der Methoden zusammenhängt. Schließlich ist beispielsweise ein fester Teil der Philosophie von Extreme Programming das Test Driven Development. Und genau das kann bei der Entwicklung von User Interfaces problematisch werden. Die Entwicklung eines User Interfaces geht über das einfache Umsetzen von funktionalen Anforderungen hinaus [7]. Es ist außerdem ein kreativer Prozess. Daher ist es schwierig, im Vorhinein einen Test zu formulieren. Mit Capture/Replay Tools ist dies unmöglich. Auf formalem Wege gibt es jedoch Möglichkeiten. Ebenso ist es möglich, die GUI-Test-Cases manuell zu schreiben. Voraussetzung ist im Vorhinein jedoch auf jeden Fall eine Definition und Planung gewisser GUI Komponenten.

Wahl der richtigen Methodik

Die Wahl der richtigen Methodik (Technologie, Testorakel, etc.) hängt von vielen Faktoren ab, z.B. wie wichtig ist Wartbarkeit oder wie lange dauert das Projekt. Capture/Replay-Tools wirken beispielsweise im ersten Augenblick als schnelle und gute Lösung, zeigen aber bei langfristiger Verwendung Nachteile, da die Skripten sehr fehleranfällig bei Veränderungen am User Interface sind (vor allem bei Layout-Änderungen). Manuelles Testen ist fehleranfällig, langwierig und äußerst zeitaufwändig. Methoden, bei denen Skripte und Test-Cases von Entwicklern geschrieben werden, sind für Test Driven Development geeignet, machen weniger Probleme bei Layout-Anpassungen und sind im allgemein gut wartbar. Allerdings muss vorab Zeit investiert werden, um die Test-Cases zu erstellen. Sie haben zudem in komplexen Systemen, in denen beispielsweise eigens entwickelte GUI-Komponenten verwendet werden, große Vorteile. Darüber hinaus stehen, sofern die Test-Cases in einer Programmiersprache geschrieben werden, auch die Vorteile und Möglichkeiten der jeweiligen Programmiersprache zur Verfügung. Letztendlich ist die Wahl der richtigen Methode abhängig von der Perspektive der Entwickler und von den Eigenschaften des Projektes. [10]

3.3.2. Tools

Auf dem Markt gibt es eine große Anzahl von Tools, die Entwickler beim Testen ihrer Anwendungen unterstützen sollen. Wie in Abschnitt 3.1 beschrieben, gibt es beispielsweise das JUnit-Framework, welches die Umsetzung von Unit Tests in Java ermöglicht.

Auch zum Testen von User Interfaces gibt es eine große Anzahl von Tools. Viele gingen aus akademischen Arbeiten hervor, andere sind kommerziell entstanden und werden von Firmen vertrieben. Es folgt nun die Vorstellung und der Vergleich einiger GUI-Testing-Tools für die Programmiersprache Java.

UISpec4J

UISpec4J[4] ist ein eine Open Source Java Library, die u.a. auf JUnit aufbaut. Sie ermöglicht Unit Testing der Grafikkomponenten von Java-Swing-basierten Applikationen. UISpec4J ermöglicht das einfache Schreiben von Testscripts und deren Ausführung. Es bietet außerdem folgende Funktionen:

- Eine eigene Testklasse zur Ausführung von Unit Tests auf Swing-Klassen
- Wrapper für GUI Elemente, wie Trees, Tables, Checkboxen etc.
- Mechanismen, um Dialoge und Popups abzufangen

Somit können mit UISpec4J schon vor der eigentlichen Entwicklung einer grafischen Oberfläche Testscripts geschrieben werden, welche GUI Elemente ansprechen bzw. deren Status auslesen können. Die Testscripts werden in Form von Testklassen verfasst, welche die Klasse *TestCase* des JUnit-Frameworks erweitern. Dadurch ist die Handhabung sehr einfach und eine Einbindung in eine bestehende Test-Suite mit JUnit-Test-Cases ist problemlos durchführbar. Für die Nutzung von UISpec4J wird auf der Website der Entwickler eine JAR-Datei zum Download angeboten. Diese kann in ein Java-Projekt eingebunden werden. Danach können die oben genannten Features mittels Funktionen der JAR-Datei genutzt werden. Abbildung 3.2 zeigt einen Screenshot einer Testklasse, die die UISpec4J-Library nutzt. Die Klasse auf dem Screenshot erweitert die Klasse UISpec-TestCase, welche wiederum von der Klasse JUnit aus dem JUnit-Framework abgeleitet wurde. Somit wird der Test vom JUnit-Framework wie ein JUnit-Test-Case behandelt (vgl. Abbildung 3.1 auf Seite 34). [16]

Pounder

Pounder[5] ist ein freies Tool, welches die Automatisierung von GUI Tests ermöglicht. Entwickler können damit Skripte aufzeichnen und in eine JUnit Test-Suite integrieren. Pounder kann als JAR-File von der Entwicklerseite heruntergeladen werden. Um die Aufzeichnung zu beginnen, muss das JAR File gestartet werden. Der Java-Classpath, in dem sich zu testende Klassen befinden, muss als Parameter übergeben werden. Das Pounder-Interface (siehe Abbildung 3.3) bietet die Möglichkeit, eine Testklasse auszuwählen. Mittels Klick auf "Record" startet die Aufzeichnung. Pounder registriert alle Maus- und Tastaturevents. Mit Klick auf "Stop" endet die Aufzeichnung. Nun kann der Test-Case mittels Klick auf "Play" wiedergegeben werden. Pounder unterstützt *visible execution*.

[4] http://www.uispec4j.org/, Februar 2013 - derzeit aktuellste Version: 2.4
[5] http://pounder.sourceforge.net/, Februar 2013

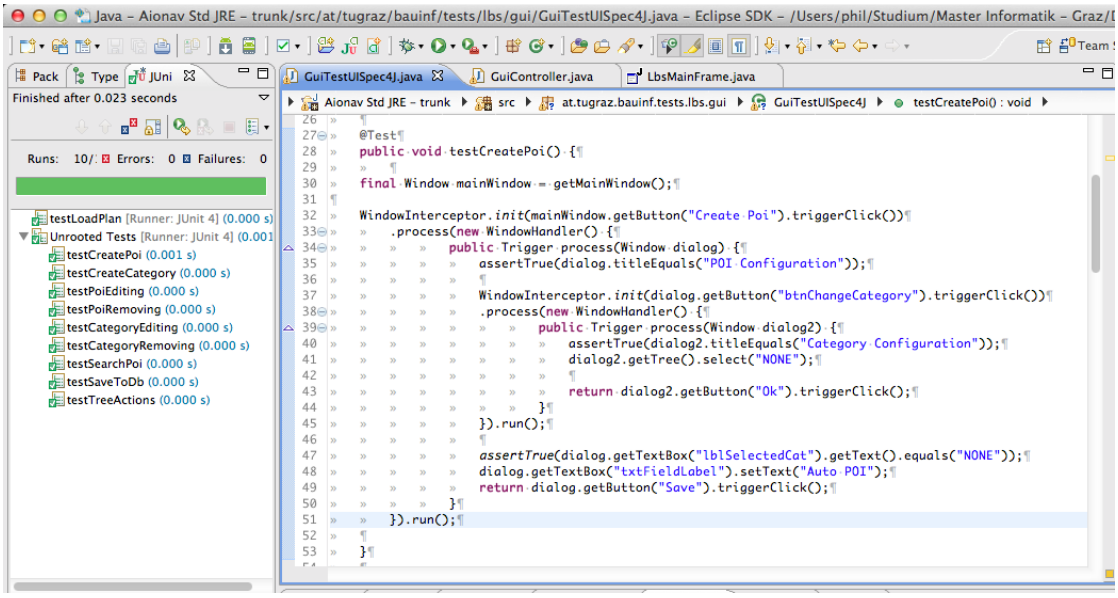

Abbildung 3.2.: UISpec4J Test-Case in der Eclipse-Entwicklungsumgebung

Das bedeutet, dass man live am Bildschirm beobachten kann, wie Pounder bei der Wiedergabe die aufgezeichneten Schritte ausführt. Aufgezeichnete Daten können außerdem als XML-Datei gespeichert und später wieder geladen werden. Es ist nicht vorgesehen, dass die XML-Files vom Entwickler selbst geschrieben werden können - es gibt also zur Erstellung von Test-Cases nur die Aufzeichnungsfunktion. Allerdings können die Skripten aus einem JUnit-Test-Case aus aufgerufen werden. Dies funktioniert, indem man die Pounder-Library in ein Java-Projekt einbindet. Nun kann eine Instanz der Player-Klasse erstellt werden, welche ein Pounder-XML-Skript laden und ausführen kann. Somit lassen sich auch Pounder-Skripte in ein bestehendes Test-Framework eingliedern. Die Erstellung von Tests ist jedoch nicht vor der Entwicklung des User Interfaces möglich, da zur Erstellung die Aufnahmefunktion nötig ist. [15]

jfcUnit

jfcUnit[6] ist eine Erweiterung des JUnit Frameworks, um grafische User Interfaces, die mit Hilfe von Java Swing erstellt wurden, zu testen. Es bieten Nutzern die folgenden Möglichkeiten:

- Fenster und Dialoge mittels Java Code finden und untersuchen
- Auffindung von Komponenten innerhalb der Komponenten-Hierarchie von Swing
- Auslösen von Events auf gefundenen Komponenten (z.B. Klicken, Schreiben etc.)

[6] http://jfcunit.sourceforge.net/, Februar 2013

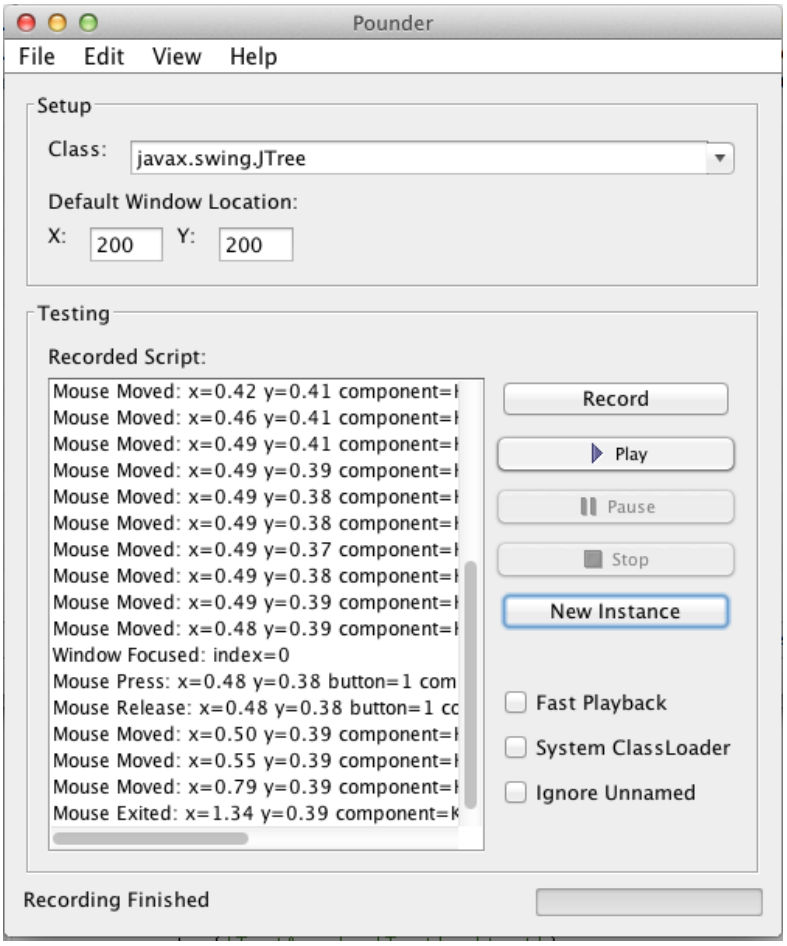

Abbildung 3.3.: Das User Interface von Pounder während der aktiven Aufzeichnung eines Test-Cases

- Thread Safety testen

Um jfcUnit nutzen zu können, müssen zwei Libraries ins Java Projekt eingebunden werden (jUnit selbst und Jakarta[7]). Ebenso gibt es entsprechende Plugins für Eclipse und JBuilder[8].

jfcUnit ermöglicht einerseits das Erstellen von Tests innerhalb eines JUnit-Test-Cases mit Hilfe von Library-Funktionen, andererseits können auch XML-Skripte erstellt werden. Zudem gibt es eine Recording-Funktion, mit welcher Events, die von einem Benutzer ausgelöst werden, aufgezeichnet werden können.

Leider wurde sowohl die Entwicklung von jfcUnit als auch von der erforderlichen Jakarta-Library schon vor relativ langer Zeit eingestellt (2004). Die genannte Entwickler-

[7] http://jakarta.apache.org/regexp/, Februar 2013
[8] http://www.embarcadero.com/products/jbuilder, Februar 2013

Website ist jedoch noch online und bietet Tutorials sowie eine API-Dokumentation. [23]

Abbot

Abbot[9] ist ein freies GUI-Test-Framework und eine Erweiterung von JUnit. Es bietet sowohl die Möglichkeit über Java-Code in Form von Unit Tests neue Test-Cases zu erstellen, als auch Tests in Form von XML-basierten Skripten zu schreiben. Außerdem gibt es eine Möglichkeit, Test-Cases aufzuzeichnen. Das Framework ist für Test First Development konzipiert und bietet die Möglichkeit, Referenzen auf Gui Komponenten zu erhalten. Außerdem enthält es Klassen, mit denen man Aktionen eines Benutzers auf GUI Komponenten simulieren kann. Fundamentale Komponente von Abbot ist eine Robot-Klasse, welche User Events generieren kann. Weiteres existieren für alle GUI Komponenten Tester-Klassen, mit welchen man auf den Komponenten Aktionen ausführen kann oder gewisse Eigenschaften auslesen kann. Um einen Unit Test zu erstellen, muss eine Library von der Entwicklerseite heruntergeladen und ins Java-Projekt eingebunden werden. Eine Subklasse vom Typ *ComponentTestFixture* (Tester-Klasse) muss erstellt und implementiert werden. In ihr befindet sich die Testlogik. Der Test kann mittels der Klasse TestHelper gestartet werden. [12]

Das Abbot Framework enthält außerdem einen eigenen Skript-Editor, mit welchem Skripte erstellt bzw. aufgezeichnet werden können (siehe Abbildung 3.4).

Fest - (Functional Swing GUI Testing)

Fest[10] ist eine Sammlung von Libraries zum Testen von User Interfaces, wobei eine Kompatibilität zu JUnit gegeben ist. Es handelt sich dabei um ein Open Source Projekt unter der Apache 2.0 Lizenz. Es besteht aus folgenden Modulen (siehe auch 3.5):

Funktionales Swing GUI Testing Dieses Modul simuliert Benutzer-Eingaben auf Betriebssystemebene und beinhaltet Funktionalitäten zur Suche von GUI Komponenten. Als besonderes Feature kann es eine Momentaufnahme des Bildschirms machen, wenn ein Fehler auftritt. So kann im Nachhinein noch besser nachvollzogen werden, warum ein Fehler aufgetreten ist.

Fluent Assertions Dises Modul ermöglicht es, Zusicherungen, wie man sie aus JUnit kennt, zu machen. Möglich ist dies durch die Verwendung der assertThat()-Methoden, welche boolsche Ausdrücke auswerten können.

Reflection Das Reflection-Modul hilft dabei, GUI-Komponenten ohne Probleme bzgl. Type-Casting oder Exception-Handling aufzufinden bzw. Referenzen darauf zu erhalten.

[9] http://abbot.sourceforge.net/, Februar 2013
[10] http://fest.easytesting.org/, Februar 2013

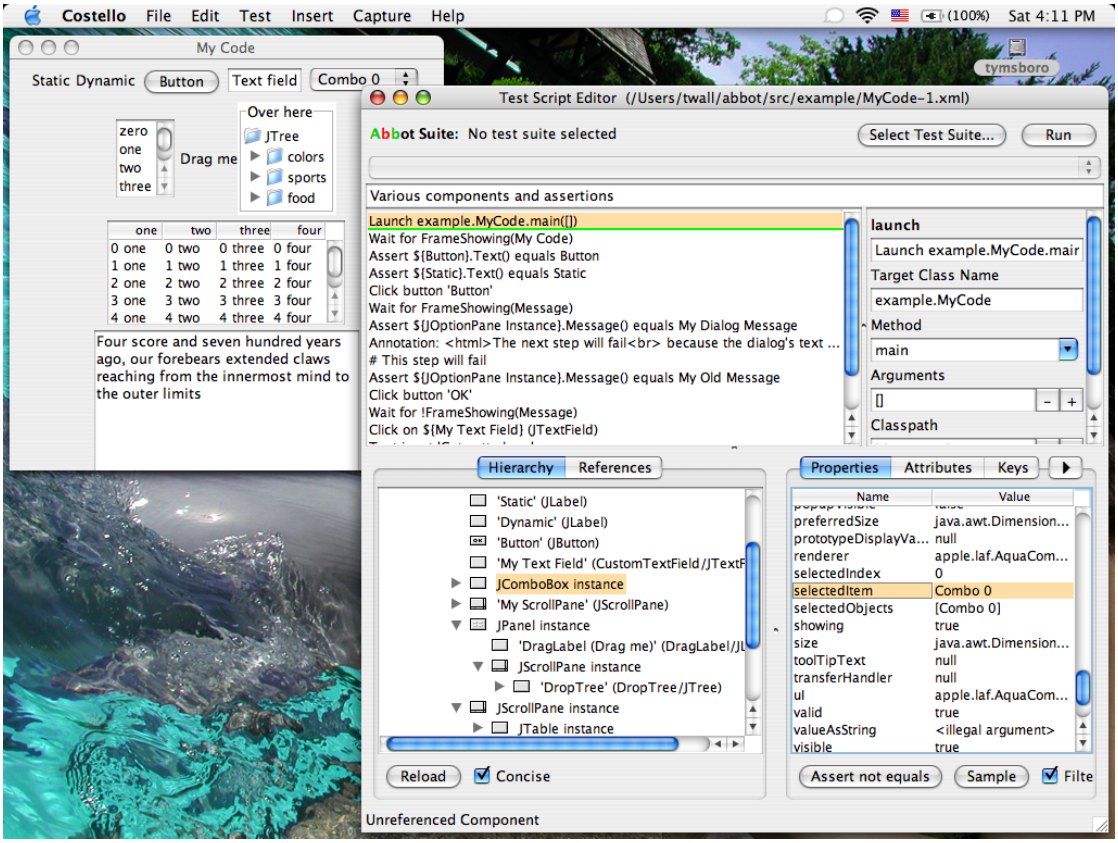

Abbildung 3.4.: Costello - Der Script-Editor von Abbot [http://abbot.sourceforge.net/doc/images/costello.png, Februar 2013]

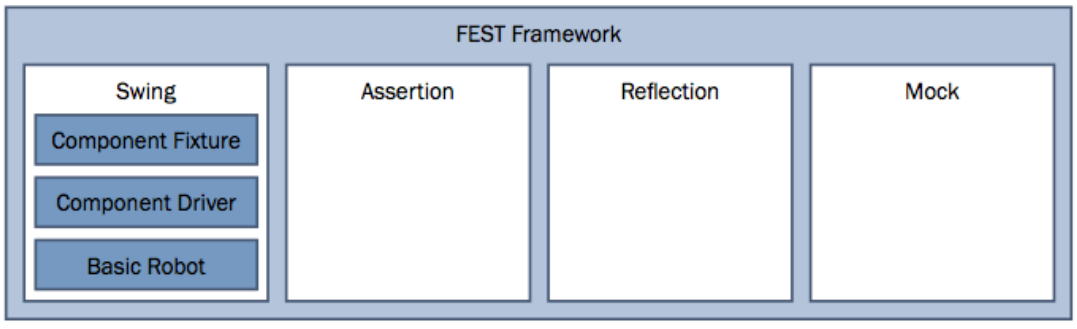

Abbildung 3.5.: Die Module von Fest [10]

EasyMock Template Das EasyMock-Modul soll vor allem dabei helfen, Code-Duplikate zu verringern. Mock-Objekte sollen dabei die Aufgabe übernehmen, erwartete und tatsächliche

Werte von Variablen zu vergleichen (weitere Infos finden sich auf der Entwicklerseite[11]).

Fest wird im Gegensatz zu einigen der anderen vorgestellten Test-Umgebungen weiterentwickelt und hat eine aktive Community von Nutzern und Entwicklern. ([13], [10])

Marathon

Marathon[12] ist ein GUI Test Automatisierungs-Framework. Es wird in einer kommerziellen und einer Open Source Variante angeboten. Marathon ist eine Standalone-Applikation.

Kommerzielle Version Die kommerzielle Version (MarathonITE) ist auf das Testen von Java/Swing Applikationen ausgelegt und für Windows, OSX und Linux verfügbar. Es bietet die Möglichkeit zur Aufzeichnung von Testskripten. Es gibt zwei Aufzeichnungsmodi: Der Erste bietet die Möglichkeit, Rohdaten von Events zu erfassen (Klick-Koordinaten, Event-Details). Beim Zweiten werden semantische Informationen (z.B. Auswahl von GUI Elementen durch ihren Namen oder ein Label) aufgezeichnet. Der Benutzer kann im Betrieb frei zwischen den beiden Modi wählen. Ebenso existiert ein eingebauter Debugger zur detaillierten Fehlersuche. Außerdem ermöglicht Marathon während der Aufzeichnung, ein Kontextmenü auf GUI-Elementen aufzuklappen. Dadurch sind Status-Informationen über diese Elemente zugänglich. Als Skriptsprachen kommen JRuby und JPython zum Einsatz. Aus den Skripten heraus kann auch auf Java-Klassen zugegriffen werden bzw. nativer Java-Code ausgeführt werden. Eine herausragende Eigenschaft von Marathon zeigt sich bei der Objekt-Erkennung. Marathon bietet zwei Arten der Objekt-Erkennung: Einerseits kann ein einzigartiger Name vollautomatisch aus Objekt-Eigenschaften erstellt werden, andererseits können Elemente anhand ihrer Eigenschaften gefunden werden. Die Ergebnisse eines Testdurchlaufes zeigt Marathon in Form eines Reports an, wobei zusätzlich eine Bildschirmaufnahme im Fehlerfall verfügbar ist. Einen großen Vorteil bietet Marathon, indem es viele Möglichkeiten zum Starten von zu testenden Anwendungen bietet. So können etwa Java Applikationen über Eingabeaufforderung oder auch als ausführbare Dateien (erstellt mit launch4j[13] - ein Framework mit dem ausführbare Dateien für Windows aus JAR-Dateien erstellt werden können) gestartet werden. Auch der Start von Applets und Webstart-Applikationen ist möglich. Außerdem lassen sich *Data Driven Tests* umsetzen, bei denen Datensätze aus CSV-Files als Datenquelle für Test-Cases genutzt werden können. Schließlich können Test-Cases in verschiedene Module aufgeteilt werden, was die Übersichtlichkeit bei der Verwaltung von Test-Cases erhöht. [14]

Open Source Version Die Open Source Version von Marathon ist der Vorläufer der kommerziellen Version. Sie hat einen eingeschränkten Funktionsumfang und richtet sich an Entwickler von Kleinprojekten. Konkret gibt es folgende Funktionen in der Open Source Version nicht: Starten von ausführbaren Dateien sowie Webstarts bzw Applets, keine Aufteilung in Module, kein Data Driven Testing, eingeschränkter Support. Trotz

[11] http://docs.codehaus.org/display/FEST/EasyMock+Template+Module, Februar 2013
[12] http://marathontesting.com/, Februar 2013
[13] http://launch4j.sourceforge.net/, Februar 2013

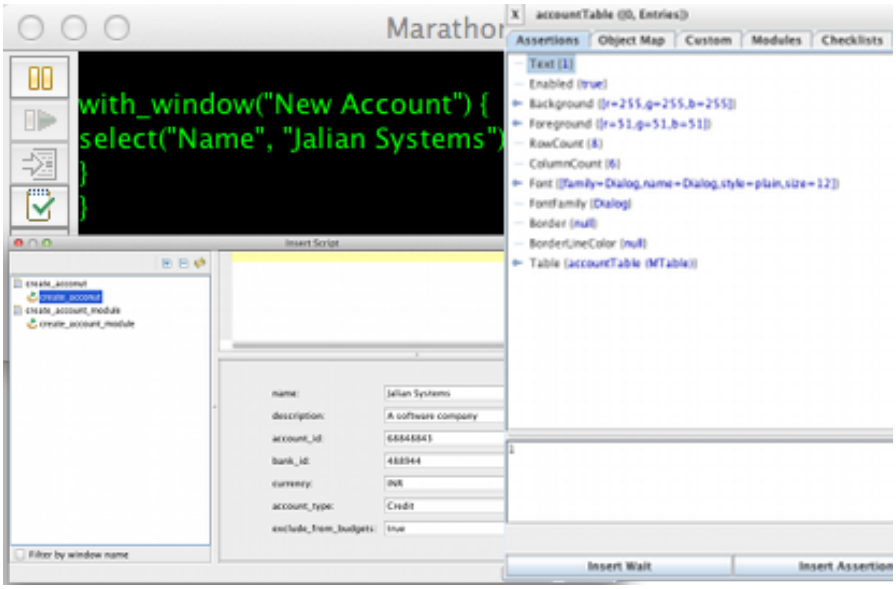

Abbildung 3.6.: Das Aufzeichnungs-Tool von Marathon (http://marathontesting.com/marathonite/, Februar 2013)

allem bietet es Catpure/Replay Funktionalität, sowie die Möglichkeit selbst Skripte zu schreiben. [14] Abbildung 3.6 zeigt das Aufzeichnungs-Tool von Marathon, Abbildung 3.7 die Anzeige nach erfolgter Ausführung von einer Test-Suite.

Ranorex Test Tools

Die Firma Ranorex[14] mit Firmensitz in Graz hat sich auf die Entwicklung von Tools zum automatisierten Testen von User Interfaces spezialisiert. Sie bietet ein Automatisierungs-Framework für Entwickler und Tester an. Im Unterschied zu den anderen vorgestellten Tools wird das Framework von Ranorex ausschließlich kommerziell vertrieben. Als Downlaod gibt es neben detaillierten Preisinformationen lediglich eine Testversion auf der Herstellerseite. Einen entscheidenden Vorteil, den Ranorex bietet, ist die Unterstützung vieler verschiedener Plattformen (.net, Java, Win32, Ios, Android, Ajax, Javascript, Adobe Flash...). Das Framework Ranorex Studio von Ranorex besteht aus den folgenden Komponenten, die beim Erstellen und Debuggen, sowie beim Management von Tests helfen([40]):

- **Ranorex Test-Suite** Mittels der Ranorex Test-Suite können Test-Cases verwaltet werden (sowohl aufgezeichnete, als auch geskriptete Tests). Weiters besteht die Möglichkeit, Test Daten zu verwalten und diese in Test-Cases einzubinden (SQL, CSV, EXCEL).

[14] www.ranorex.com, Februar 2013

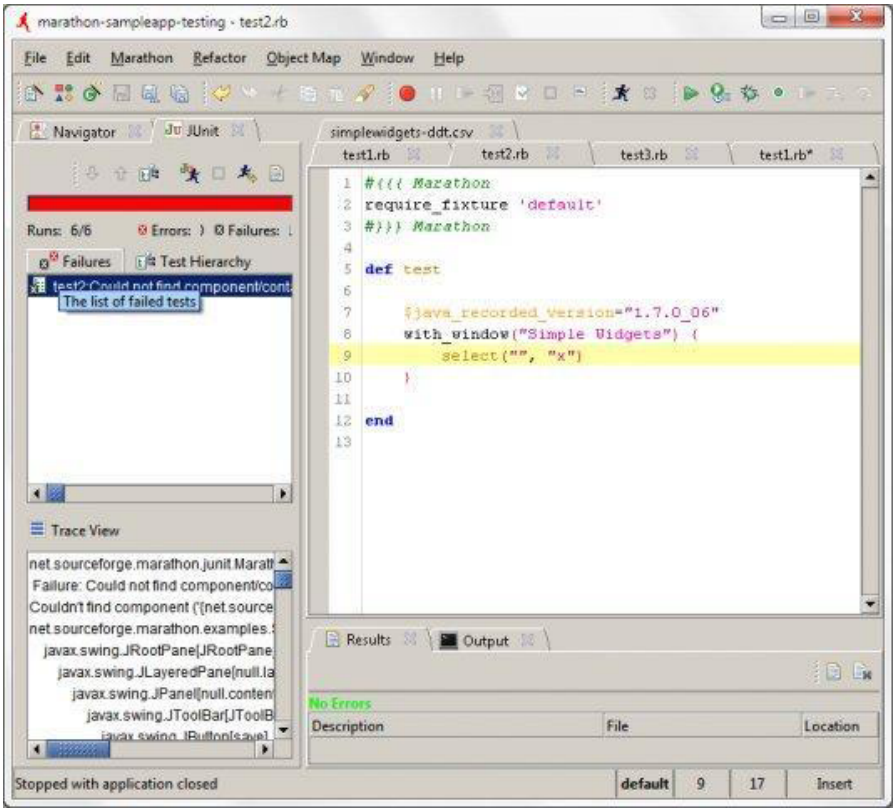

Abbildung 3.7.: Ansicht von Marathon (Open Source) nach Ausführung einer Test-Suite [http://sourceforge.net/projects/marathonman/?source=dlp, Februar 2013]

- **Ranorex Repository** Weiters ist Ranorex Repository auch ein Tool zum Verwalten und Ausführen von Tests.
- **Ranorex Recorder** Der Ranorex Recorder (siehe Abbildung 3.8) ist ein Capture/Replay Tool, mit welchem Aktionen, die ein Benutzer ausführt, aufgezeichnet werden können. Aufgezeichnete Daten werden in Action Tables visualisiert und können selbstverständlich auch wiedergegeben werden.
- **Ranorex Report** Das Tool Ranorex Report (siehe Abbildung 3.9) gibt einen Überblick über die Ausführung von Test-Cases. Fehler, die bei der Ausführung von Tests auftreten, werden entsprechend angezeigt und mit einer Momentaufnahme des Bildschirms versehen.

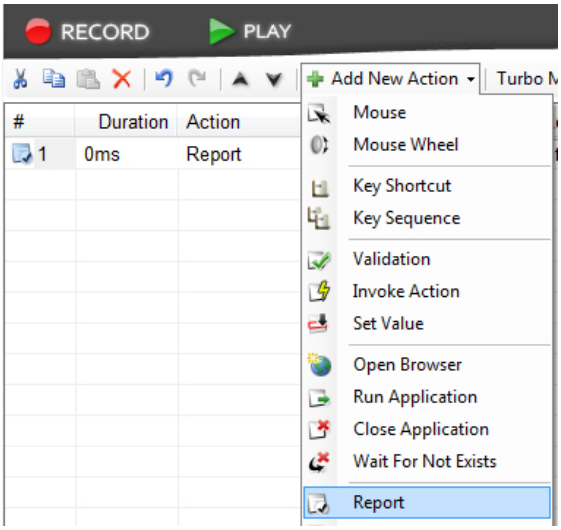

Abbildung 3.8.: Ansicht des Ranorex Recorders zum Aufzeichnen von Test-Cases [37]

3.3.3. Evaluierung und Vergleich der vorgestellten Tools

Wahl des passenden Tools

Wie schon in Abschnitt 3.3.1 erwähnt, ist die Wahl der richtigen Testmethodik eine große Herausforderung. Es gilt viele Faktoren zu beachten, die sich aus der Beschreibung der Tools ableiten:

- **Projektspezifische Faktoren** - Entscheidend für die Wahl des Tools ist einerseits die Größe und Dauer des Projekts, aber auch Implementierungsdetails, wie z.B. die verwendete Programmiersprache und Entwicklungsumgebung.
- **Personalspezifische Faktoren** - Die Entwickler und das Management müssen sich für eine Methodik entscheiden. Sowohl Erfahrung der beteiligten Personen, als auch ihr Informationsstand hat auf die Wahl des richtigen Tools große Auswirkungen.
- **Budget** Wie im Falle der Ranorex Test-Suite oder von Marathon, müssen finanzielle Mittel für das Test Framework aufgewendet werden, was für manche Projekte nicht möglich ist.
- **Features** Letztendlich sind die Funktionen, die ein Framework bietet entscheidend. Im folgenden Abschnitt werden die Features der vorgestellten Tools nochmals miteinander verglichen.

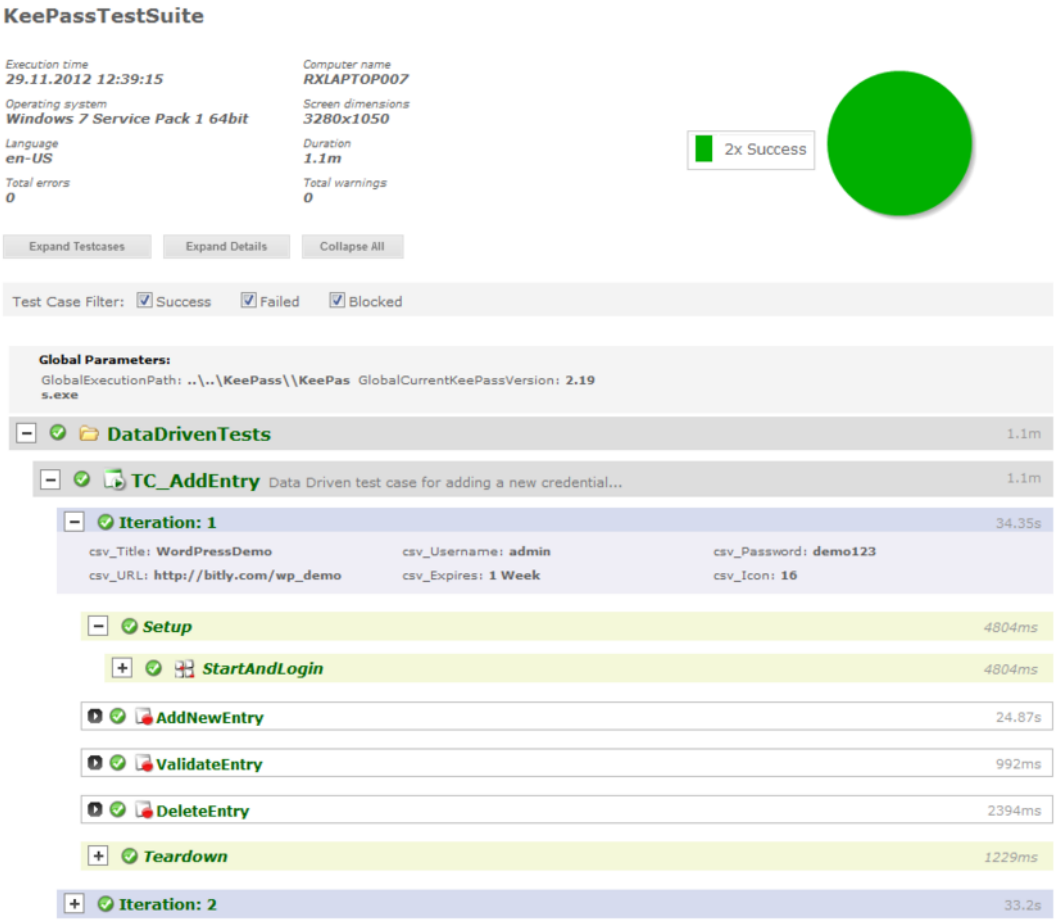

Abbildung 3.9.: Ansicht von Ranorex Report nach erfolgreichem Durchlaufen einer Test-Suite [37]

Feature-Vergleich

Die Tabelle 3.1 fasst die Funktionen und Features der vorgestellten Frameworks nochmals zusammen.

	UISpec4J	Pounder	jfcUnit	Abbot	FEST	MarathonITE	Marathon open	Ranorex
Capture/Replay	nein	ja	ja	ja	nein	ja	ja	ja
Test-Cases programmierbar	ja	nein	ja	ja	ja	ja	ja	ja
XML-Skripte	nein	ja	ja	ja	nein	nein, aber JRuby/Jython	nein, aber JRuby-/Jython	?
JUnit Tests möglich	ja	ja	ja	ja	ja	nein	nein	nein
Geeignet für Test First Dev	ja	nein	ja	ja	ja	ja	ja	nein
Visible Execution	nein	ja	nein	ja	nein	ja	ja	ja
Lizenz	Open Source	GNU GPL	Open Source	Open Source	Apache 2.0	kommerziell	Open Source	kommerziell
In Entwicklung	?	?	nein	?	ja	ja	ja	ja
Letzte Version von	12/2010	2002	12/2004	5/2012	02/2013	wird regelmäßig aktualisiert	12/2012	wird regelmäßig aktualisiert
Bildschirmfoto bei Fehler	nein	nein	nein	nein	ja	ja	?	ja
Nutzung	JAR einbinden	Start eines JAR Files; XML in JUnit aufrufen	JAR Files einbinden	Executable JAR bzw. Einbindung	JAR eindinden	Standalone JAR	Standalone JAR	Installer für Windwos
IDE Plugins	via JUnit	nein	ja	nein	nein	nein	nein	nein
Besonderheiten	gut kompatibel zu JUnit	-	gut kompatibel zu JUnit	-	aktive Community	-	up to date und offen	nur Windows; viele Test-Plattformen

Tabelle 3.1.: Vergleich der vorgestellten Test-Frameworks

4. Praxisprojekt

4.1. Anforderungsanalyse

Im Rahmen dieser Arbeit wird eine Applikation namens AIONAV LBS Editor entwickelt, die Bestandteil eines Anwendungs-Frameworks der Firma AIONAV[1] sein wird. Die Firma AIONAV entwickelte Positionierungs-Lösungen für den Indoor- und Outdoorbereich und entwickelt sich nun über Location Based Services zum Navigationsanbieter weiter. Die Entwicklung erfolgt in der Programmiersprache Java unter Einhaltung gängiger Vorgehensweisen der *agilen Softwareentwicklung*, wie sie in Abschnitt 2 beschrieben werden, sowie unter Verwendung von modernen Test-Methoden im Sinne des *Test Driven Developments*, wie sie in Abschnitt 3 beschrieben werden. In Abschnitt 5 findet eine Evaluierung des Projekts in Hinblick auf die Verwendung der vorgestellten Methoden statt.

4.1.1. Anforderungen an die zu entwickelnde Applikation

Ziel dieser Arbeit ist es, eine Desktop-Anwendung zu entwickeln, die auf das *AIONAV-Framework* aufbaut. Die Anwendung soll das Erstellen und Planen gewisser Navigations-Szenarien im Indoor und Outdoorbereich ermöglichen, welche von mobilen Anwendungen verwendet werden können. Szenario bedeutet in diesem Fall, dass auf bestehendem Kartenmaterial gewisse *Points of Interest (POI)*, also Orte von besonderem Interesse, und Routen definiert werden können. Im Grunde kann dadurch beliebiges Kartenmaterial wie etwa ein globalisiertes Bild (dies ist ein Bild, welches Daten über den geografischen Standort seines Inhaltes enthält), oder ein Gebäudeplan mit Zusatzinformationen und multimedialen Inhalten versehen werden. Ganz allgemein ist ein Point of Interest also ein Ort bzw. ein Bereich in einem Plan, der mit gewissen Informationen verknüpft wird. Die Information, welche in verschiedenen Formen vorliegen kann (Text, Audio, Video, Verlinkung zu anderen Inhalten etc.), wird in Verbindung mit dem Ort gespeichert und kann zu einem späteren Zeitpunkt (auch von anderen Anwendungen) wieder abgerufen werden. Eine Route ist ein Weg, der über mehrere Punkte führt. Es muss sich bei diesen Punkten nicht unbedingt um Points of Interest handeln. Eine Route kann jedoch den Weg zwischen verschiedenen Points of Interest beschreiben. Im AIONAV LBS Editor soll auch erfasst werden, was passieren soll, wenn sich ein Benutzer in einem gewissen Kontext (in der Nähe eines oder mehrerer Points of Interest) befindet und wie darauf reagiert werden soll. Um schließlich auch die autonome Navigation von Benutzern ermöglichen zu können, soll außerdem eine Funktionalität implementiert werden, mit der ein Wege-Netz auf einem vorhandenen Plan eingezeichnet werden kann. Dabei werden Weg-Punkte auf

[1] www.aionav.com, Februar 2013

dem Plan platziert, die sich schließlich zu sogenannten *Paths* verbinden. Jeder Path ist im späteren Sinne dann ein Bereich, auf dem sich ein Benutzer der mobilen App bewegen kann. Diese begehbaren Bereiche werden zur Navigation und zur Errechnung von Wegen genutzt, um auf schnellsten Wege zu einem Point of Interest zu gelangen (Routing der Benutzer).

Schließlich soll der AIONAV LBS Editor die Möglichkeit bieten, ein gesamtes Szenario zu simulieren. Das bedeutet, dass der Ersteller/Verwalter eines Szenarios, in einer Simulationsumgebung genau begutachten kann, was einem Benutzer passieren würde, der sich real entlang einer Route bewegt oder in die Nähe gewisser Points of Interest kommt.

Eine wichtige Anforderung an die Anwendung ist, dass sie intuitiv und leicht über ein *grafisches User Interface bedienbar* ist. Das einfache Erstellen, Editieren oder Löschen von Szenarien soll möglich sein. Außerdem soll es möglich sein, anhand von Templates (Vorlagen) wiederkehrende Points of Iterest mit ähnlichen Eigenschaften schnell anlegen zu können. Ein Teil dieser Arbeit dreht sich deshalb auch um das Testen von grafischen Benutzerinterfaces.

Der AIONAV LBS Editor soll für die verschiedensten Zwecke verwendbar sein. Einige Varianten, wie er verwendet werden kann, finden sich im Abschnitt über verschiedene Einsatzzwecke.

Wie schon erwähnt, erfolgt die Entwicklung im Sinne der *agilen Softwareentwicklung*, vor allem nach der Methode des *Extreme Programming*, welches eigentlich auf die Entwicklung in Teams ausgelegt ist. Die Entwicklung des AIONAV LBS Editors erfolgt im Kontext dieser Arbeit prinzipiell nur von einer Person, jedoch integriert in ein Team von Entwicklern von AIONAV-Komponenten. Deshalb stellt es auch eine Herausforderung dar, die agilen Konzepte auf die Entwicklung mit einer Person anzuwenden. Weitere Informationen dazu finden sich im Abschnitt 5.

4.1.2. Mobile Anwendung

Als mobiles Gegenstück zum AIONAV LBS Editor wird von AIONAV eine mobile Applikation entwickelt. Sie soll es potentiellen Nutzern ermöglichen, sich in den fertigen Szenarien zu bewegen und Informationen für entsprechende Points of Interest abzurufen. Benutzer der mobilen Anwendung können sich somit an den Orten bewegen, die durch oben genanntes Kartenmaterial abgebildet und "digital möbliert" wurden. Die mobile Applikation navigiert die Benutzer durch die Umgebung und versorgt sie an Points of Interest mit den nötigen Informationen. In Anhang A finden sich noch einige weitere Ausführungen sowie Screenshots vom Prototypen der mobilen Anwendung.

4.1.3. Geplante Einsatzzwecke

Im Folgenden werden mögliche Einsatzzwecke für den AIONAV LBS Editor beschrieben, welche auch als reale Projekte geplant sind oder bereits umgesetzt werden.

4.1.4. Wanderweg Ligist

Die Gemeinde Ligist in der Nähe von Graz plant den Einsatz des AIONAV LBS Editors zur Digitalisierung eines Wanderweges. Interessante Punkte entlang des sogenannten Keltenweges[2] sollen dementsprechend als Points of Interest markiert werden. Wanderer werden von der mobilen Anwendung entlang des Wanderweges geführt. Gelangen sie an einen Point of Interest, werden ihnen interessante Informationen zu den Orten gezeigt.

4.1.5. Shoppingcenter - Einkaufen der Zukunft

Ein großes, schweizerisches Handelsunternehmen[3] plant ebenso den Einsatz der AIONAV LBS-Lösungen, um eine Form des *Modernen Einkaufens der Zukunft* umzusetzen. Filialen des Unternehmens sollen mit dem AIONAV LBS Editor komplett digitalisiert und mit Produktinformationen, Aktionen und Angeboten versehen werden. In Einkaufszentren sollen darüber hinaus alle Geschäfte erfasst werden. Ebenso sollen benutzerspezifische, an das Einkaufsverhalten einzelner Kunden angepasste, Routen zur Verfügung stehen. Ein zuständiger Mitarbeiter kann somit in einem Plan einer Filiale den Standort von Produkten erfassen. Aktuelle Sonderangebote können ebenso markiert werden. Kunden, die mit der mobilen Anwendung ausgestattet sind, können sich somit ihren Wünschen entsprechend (z.B. laut einer digitalen Einkaufsliste) durch eine Filiale führen lassen. Sie können sich aber auch ohne Vorgabe einer genauen Route durch eine Filiale bewegen und werden z.B. in der Nähe von Sonderangeboten oder besonders interessanten Artikeln benachrichtigt.

4.1.6. Points of Interest

Die Definition und Umsetzung von Points of Interest ist ein erheblicher Teil dieser Arbeit. Points of Interest werden mit vielen verschiedenen Informationen versehen. Eine Einteilung der Points of Interest in mehrere Typen ist denkbar. So könnte es etwa eine Unterscheidung zwischen aktiven und passiven Points of Interest geben. Aktive Points of Interest informieren den Benutzer, wenn er sich in ihrer Nähe befindet. Passive Points of Interest rufen keine Meldungen hervor, können aber vom Benutzer angeklickt werden. Erst dann geben sie Informationen preis. Ebenso muss genau spezifiziert werden, was passiert, wenn ein Benutzer in die Nähe mehrerer Points of Interest kommt und in welcher Reihenfolge Informationen angezeigt werden. Für die Verwaltung der Daten ist eine Datenbank vorgesehen. Die Daten über Points of Interest müssen demnach in entsprechender Form vorliegen, um sie zweckdienlich in der Datenbank ablegen zu können. Das mobile Gegenstück zum AIONAV LBS Editor soll später genau diese Daten auslesen können, womit eine enge Zusammenarbeit zum restlichen AIONAV-Entwicklerteam notwendig ist.

[2] http://www.ligist.info/wandern-steiermark/de/tourismus/wandern/keltenweg-kurz/, August 2012
[3] www.migros.ch, August 2012

4.1.7. Datenstruktur eines Point of Interest

Ein Point of Interest muss einerseits Daten über seinen Standort und andererseits Daten über die Informationen, die für Benutzer zugänglich sein sollen, beinhalten. Weiters müssen Informationen darüber zur Verfügung stehen, was genau beim Erreichen eines Point of Interest passieren soll.

Punkt vs. Fläche

Aus der Bezeichnung *Point of Interest* (frei übersetzt: *Ort bzw. Punkt von Interesse*) geht eigentlich hervor, dass ein Point of Interest einen Punkt beschreibt. Im Sinne des AIONAV LBS Editors ist ein Point of Interest aber eine Fläche, bestehend aus einer beliebigen Anzahl von Punkten (Polygon).

Standortinformationen

Der AIONAV LBS Editor sollte die Möglichkeit bieten, Szenarien für die Indoor- *und* Outdoornavigation zu erstellen. Demnach sollten Standortinformationen in angemessener Art und Weise gespeichert werden. Zur Standortbestimmung gibt es verschiedene Koordinatensysteme, mit welchen jeder Punkt auf der Erde eindeutig über globale Koordinaten beschrieben werden kann. Im Outdoor-Bereich können diese Koordinaten auch zur Standortbestimmung via GPS für den AIONAV LBS Editor genutzt werden. Im Indoor-Bereich ist GPS im allgemeinen aber nicht verfügbar. AIONAV hat sich auf Indoor-Positionierung spezialisiert. Das AIONAV-Framework bietet Hilfsmittel an, um einerseits eine Standortbestimmung ohne GPS vorzunehmen und andererseits um Standorte auf lokalen Plänen auch in globale Koordinatensysteme zu konvertieren bzw. globale Koordinaten in ein lokales Koordinatensystem eines Lageplans umzuwandeln.

- **Globale Koordinaten:** Um Positionen auf der Erde eindeutig zu identifizieren gibt es viele verschiedene geographische Koordinatensysteme. Man kann beispielsweise geographische Koordinaten mit Längen- und Breitengrad nutzen oder auch ein UTM-Koordinatensystem[4], welches die Erde in mehrere Zonen aufteilt und diese jeweils mit einem kartesischen Koordinatensystem in Meter überzieht.
- **Lokale Koordinaten:** Vorliegendes Kartenmaterial kann mit einem lokalen Koordinatensystem überzogen werden. Beispielsweise durch Pixelkoordinaten: über eine Karte wird ein kartesisches Koordinatensystem gespannt, dessen x- und y-Koordinaten ein bestimmtes Pixel auf der Karte eindeutig identifizieren.
- **Mapping zwischen lokalem und globalem Koordinatensystem:** Das AIONAV-Framework stellt Funktionalitäten zur Verfügung, mit denen man ein Mapping zwischen lokalen und globalen Daten erzeugen kann. Dies ermöglicht es einem, die lokalen Daten einer vorliegenden Karte jederzeit mittels der Mapping-Funktionalität in globale Koordinaten umzuwandeln. Im Gegenzug ist auch die Umwandlung von globalen Koordinaten in das lokale Koordinatensystem möglich.

[4] Universal Transverse Mercator Projektion

Es wäre also möglich, Punkte auf einem lokalen Plan mittels globalen Koordinaten abzuspeichern.

Informationen für den Benutzer

An Points of Interest sollen natürlich Informationen für Benutzer vorliegen. Es gibt verschiedene Arten von Informationen. Diese sollen in beliebiger Kombination miteinander für den Benutzer bereit stehen.

- **Text:** Im einfachsten Fall liegt diese Information in Form von geschriebenem Text vor. Der Text beschreibt den Ort des Point of Interest und dessen Eigenschaften, die für einen Benutzer interessant sein können.
- **Hyperlink:** Eine weitere Möglichkeit ist die Anzeige eines Hyperlinks. Dieser kann mittels eines Web-Browsers geöffnet werden.
- **Symbol** Points of Interest können mit einem Symbol oder Bild versehen werden. Dieses wird dem Benutzer auf der Karte angezeigt.
- **Multimediainhalte:** Für Benutzer sollen erklärenden Bilder und Audio-Dateien zur Verfügung gestellt werden, welche in Form eines *Binary Large Objects* in einer Datenbank abgelegt werden können. Videos sind in der ersten Form der Anwendung noch nicht eingeplant, sollen aber eventuell zu einem späteren Zeitpunkt auch hinzugefügt werden können.
- **Social Media:** Im Zeitalter von sozialen Netzwerken soll auch die Möglichkeit zur sozialen Interaktion geboten werden. Ein mögliches Interaktions-Szenario kann zum Beispiel das *Liken* eines Point of Interest sein. Liken bedeutet im Chargon des sozialen Netzwerkes Facebook[5], dass man mittels Mausklick auf einen *Like-Button* den Gefallen an etwas (etwa einem Hyperlink oder dem Inhalt einer Website) ausdrückt. Auch im sozialen Netzwerk von Google - Google+[6] - besteht eine ähnliche Funktionalität.

Information über Interaktion mit dem Benutzer

Eine wichtige Eigenschaft eines Point of Interest ist schließlich, wie er sich dem Benutzer erkennbar machen soll. Vor allem aber ist es auch wichtig, genau zu definieren, in welchem Bereich (also an welchem Ort), sich ein Point of Interest erkennbar machen soll. Es ist nötig, genau zu definieren, was passiert, wenn ein Benutzer im Umkreis mehrerer Points of Interest ist.

- **Aktiv vs. Inaktiv** Points of Interest können inaktiv (unsichtbar) geschaltet werden, was bedeutet, dass sie bis zum Zeitpunkt der Reaktivierung, Benutzern nicht angezeigt werden.

[5] www.facebook.com, August 2012
[6] https://plus.google.com, August 2012

- **Aktiv vs. Passiv** Points of Interest können entweder aktiv oder passiv sein. Aktive Points of Interest geben dem Benutzer eine Meldung, wenn er sich in ihrer Nähe befindet. Passive Points of Interest interagieren nicht selbstständig mit dem Benutzer. Die Interaktion muss von ihm selbst (z.B. durch Klick auf einen Point of Interest) ausgelöst werden.

- **Aktivierungsfläche:** Beim Anlegen eines Point of Interest definiert der Benutzer eine bestimmte Fläche in der Karte, welche dann den Point of Interest repräsentiert. Es kann aber auch vorkommen, dass diese Fläche nicht mit dem Bereich übereinstimmt, an dem der Benutzer eine Meldung über einen Point of Interest erhalten soll. Aus diesem Grund soll es möglich sein, eine Aktivierungsfläche zu definieren, welche unabhängig vom eigentlichen Standort des Point of Interest liegt und eine Art Trigger-Fläche ist, die eine Popup-Meldung auslöst.

- **Priorität** Wenn sich mehrere Points of Interest überschneiden, geben sie je nach Priorität, die ihnen zugrunde liegt eine Meldung an den Benutzer ab. Zu diesem Zwecke gibt es verschiedene Prioritäts-Gruppen, die der Benutzer verwalten kann. Ein Point of Interest ist also genau einer Prioritäts-Gruppe zugeordnet. Höher priorisierte Points of Interest haben demnach Vorrang und werden dem Benutzer bevorzugt angezeigt. Erst wenn ein höher priorisierter Point of Interest vom Benutzer angesehen wurde, erscheinen Informationen niedriger priorisierter Points of Interest. Eine Liste der Points of Interest im Umkreis des Benutzer existiert aber ebenso, um zu verhindern, dass er einen Point of Interest mit niedrigerer Priorität übersieht.

Weitere Informationen

Einige weitere, wichtige Eigenschaften von Points of Interest sind

- **Gültigkeit:** Beschreibt die Anzeigedauer eines Point of Interest

- **Penetranz:** Wie oft erhält der Benutzer eine Popup-Meldung für einen Point of Interest

Kategorisierung

Mit der Option, Points of Interest gewissen Kategorien zuzuordnen, soll die Möglichkeit geschaffen werden, sie in der mobilen App gestaffelt anzuzeigen. D.h. ein Benutzer der mobilen App erhält die Info, dass es einen oder mehrere Points of Interest aus einer gewissen Kategorie in seiner Nähe gibt. Durch die Einführung von Kategorien ergibt sich demnach mehr Übersichtlichkeit.

Suche und Navigation

Im weiteren Projektverlauf stellte sich eine weitere wichtige Anforderungen heraus, die jedoch weitgehend das mobile Gegenstück zum AIONAV LBS Editor betrifft. Es soll

möglich sein, einen Point of Interest zu suchen. Im Anschluss soll der Benutzer der mobilen App genau zu diesem Point of Interest navigiert werden.

4.1.8. Usecase Diagramme

Im Folgenden werden die wichtigsten Usecases beschrieben. Der Teil über die mobile Anwendung wird im Rahmen dieser Arbeit zwar nicht entwickelt, aber zur Veranschaulichung ebenso angeführt.

Szenario erstellen

Grundsätzlich läuft das Erstellen eines Szenarios wie folgt ab:

- Zuerst muss der Benutzer einen Plan laden
- Der Benutzer kann schließlich einen Bereich auf dem Plan auswählen
  - Dieser Bereich kann nun als Point of Interest definiert werden
  - Points of Interest können auch aus Vorlagen erstellt werden
- Der Benutzer kann außerdem Routen in einen Plan einzeichnen
- Es können beliebig viele Points of Interest/Routen eingezeichnet werden
- Schließlich kann das Szenario abgespeichert werden
  - Späteres Wiederaufrufen und Editieren des Szenarios ist möglich
  - Das Szenario kann mobilen Anwendern zur Verfügung gestellt werden

Ein passendes Usecase Diagramm findet sich in Abbildung 4.1.

Point of Interest setzen

Das Setzen eines Point of Interest erfolgt auf folgende Weise: Der User wählt einen Bereich im Plan aus. Über einen entsprechenden Menüpunkt kann er an diesem Ort einen Point of Interest erzeugen. Ein Dialogfenster erfragt vom Benutzer die entsprechenden Daten für den Point of Interest. Es kann zuerst eine Vorlage gewählt werden, welche schon gewisse Voreinstellungen für einen Point of Interest enthält. Anschließend kann ein Beschreibungstext hinzugefügt werden. Außerdem können verschiedenste Arten von Mediendateien für den Point of Interest hinterlegt werden. Das Usecase Diagramm dazu befindet sich in Abbildung 4.2.

Route einzeichnen

Zum Einzeichnen von Routen bieten sich dem Benutzer mehrere Möglichkeiten an. Einerseits kann er mehrere Punkte definieren, an denen die Route entlang führen soll, andererseits kann er einen Startpunkt und einen Endpunkt definieren. Die kürzeste Route wird dann automatisch berechnet. Das Usecase Diagramm dazu befindet sich in Abbildung 4.3.

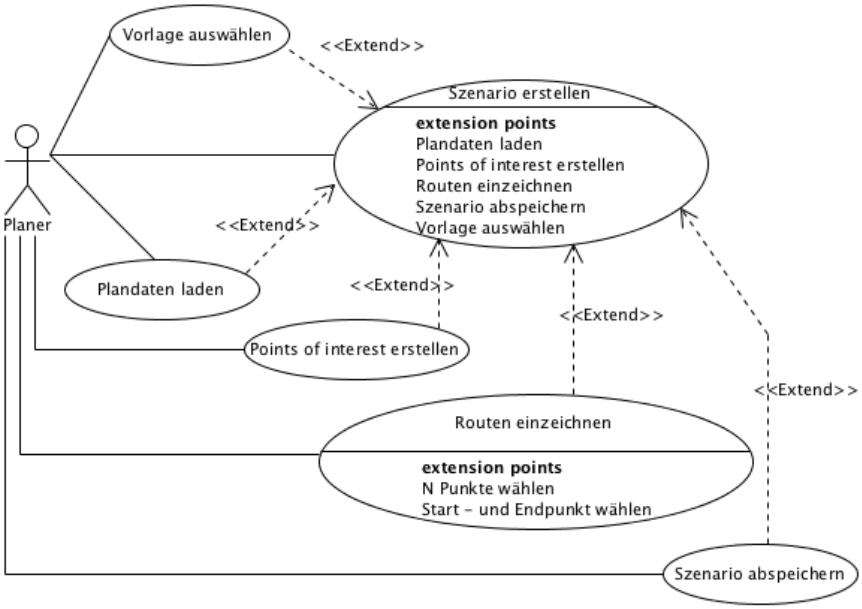

Abbildung 4.1.: Usecase Diagramm für das Erstellen eines Szenarios

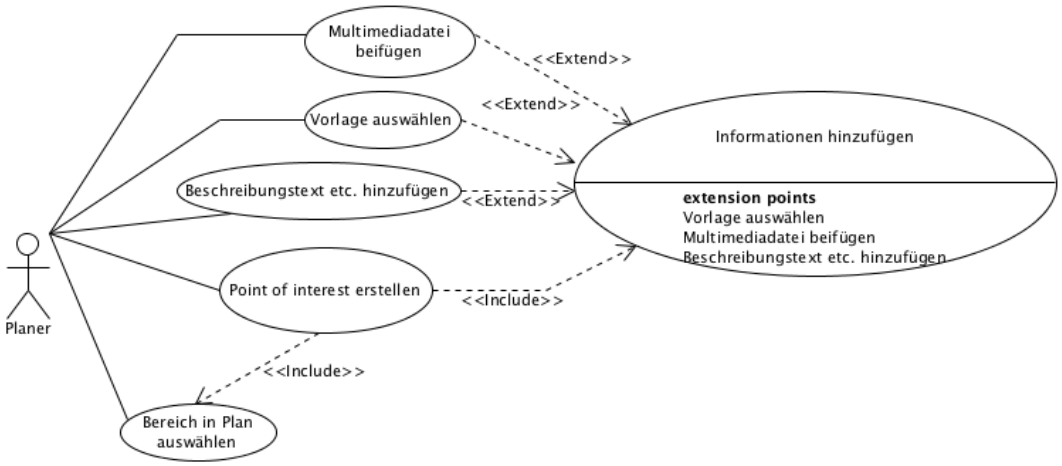

Abbildung 4.2.: Usecase Diagramm für das Erstellen eines Point of Interest

Simulation eines Szenarios

Nach dem Erstellen eines Szenarios kann man es simulieren. Das bedeutet, dass man es aus Sicht eines mobilen Nutzers betrachten kann. Ein Benutzer kann zu diesem Zwecke

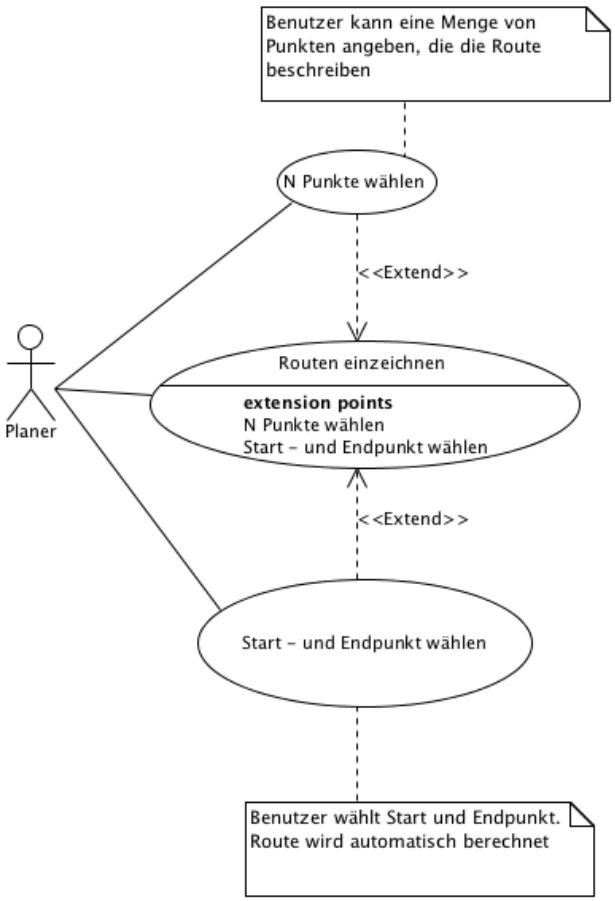

Abbildung 4.3.: Usecase Diagramm für das Erstellen einer Route

entweder eine Simulation entlang einer Route ansehen oder auf einen Punkt klicken, um zu sehen, welche Informationen einem Benutzer mit mobiler App dort gezeigt werden. Bei der Simulation der Route klickt der Benutzer auf Start. Eine virtuelle Figur bewegt sich entlang einer Route im Szenario. Sobald sich die Figur einem oder mehreren Points of Interest nähert, werden Informationen dazu sichtbar. Das passende Usecase Diagramm befindet sich in Abbildung 4.4

4.1.9. User mit mobiler Anwendung

Zur Veranschaulichung folgen nun noch Usecases, die beschreiben, wie sich die mobile Applikation[7] verhält.

[7] wird parallel zu dieser Arbeit von anderen Entwicklern erstellt

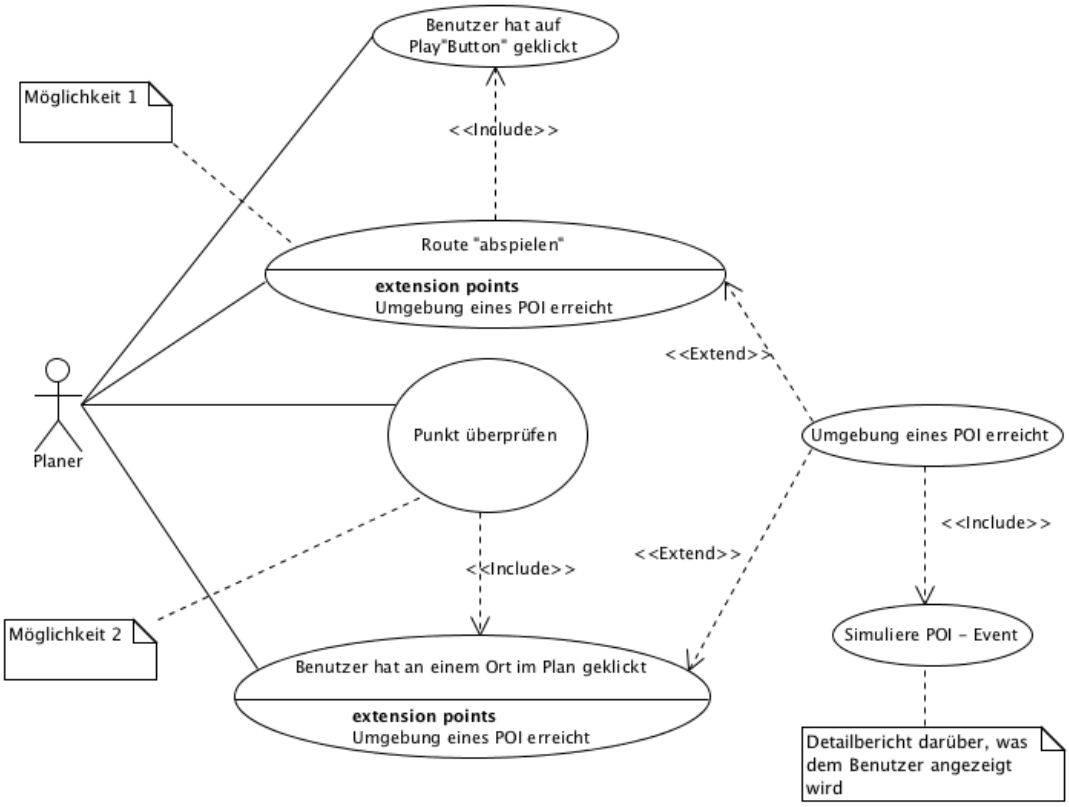

Abbildung 4.4.: Usecase Diagramm für die Szenariosimulation

Navigation entlang einer Route

Zur Navigation entlang einer Route wird dem Benutzer der mobilen App genau angezeigt, wo er sich zum jeweiligen Zeitpunkt auf dem Plan befindet. Außerdem erhält er laufend Informationen über Points of Interest in der Nähe seines Standortes. Wie sich die mobile App beim Erreichen eines Point of Interest verhält, kann dem Usecase über das Erreichen eines Point of Interest entnommen werden, wie er auch im nächsten Abschnitt beschrieben wird. Das entsprechende Usecase Diagramm für die Navigation befindet sich in Abbildung 4.5.

Erreichen eines Point of Interest

Der Benutzer kann sich entweder von der mobilen App an einer Route entlang führen lassen oder sich frei bewegen. In beiden Fällen kann er sich einem Point of Interest nähern bzw. in seinen näheren Umkreis gelangen. Der Benutzer erhält immer Hinweise auf Points of Interest in seiner Umgebung. Befindet er sich in unmittelbare Nähe eines Point of Interest, so erhält er eine Popup-Info. In Abbildung 4.6 wird dieser Usecase als

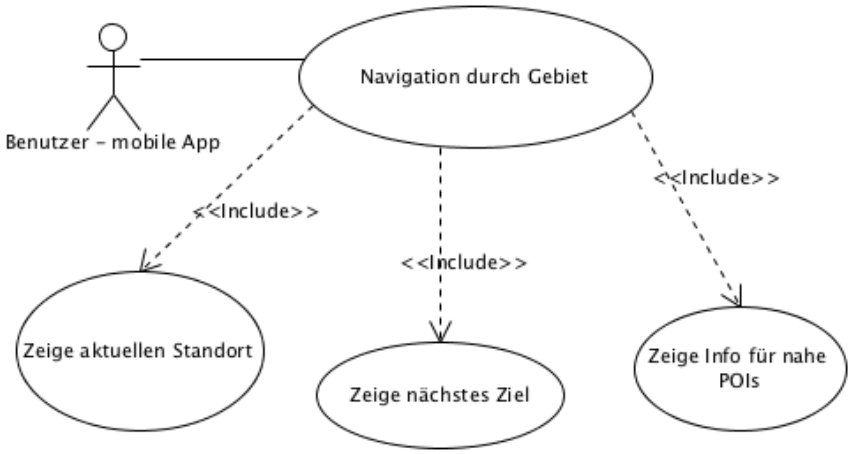

Abbildung 4.5.: Usecase Diagramm für die mobile App - *Routing*

Diagramm veranschaulicht.

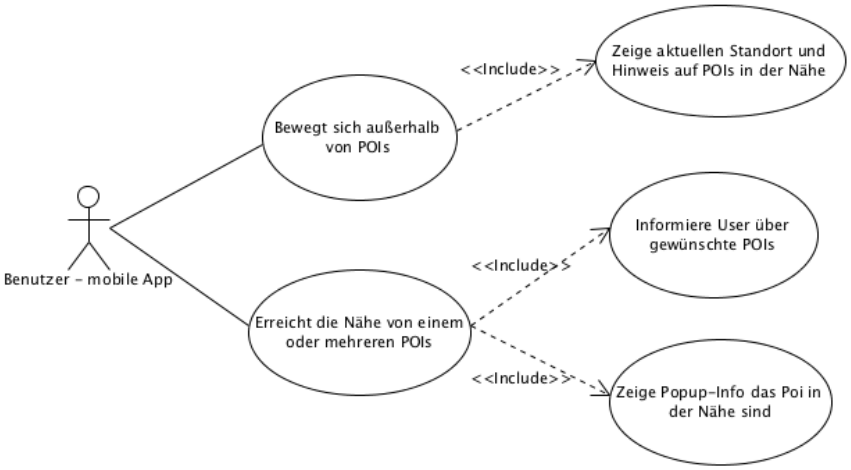

Abbildung 4.6.: Usecase Diagramm für die mobile App - *POI erreicht*

Suche und Navigation

Wie schon erwähnt, wurde als eine der wichtigsten Funktionen der mobilen App das Suchen von Points of Interest und die anschließende Navigation zu dessen Standort

identifiziert.

4.2. Umsetzung des Projektes

Relativ bald nach Projektbeginn stellte sich heraus, dass die erste reale Umsetzung des Projekts im Bereich *Einkaufen der Zukunft* (siehe auch Abschnitt 4.1.5) stattfinden sollte. Mit dem Handelskonzern Migros wurde vereinbart, bis Ende März 2013 Prototypen der mobilen Anwendung und des AIONAV LBS Editors auszuliefern, um das Navigieren von Kunden durch das Shoppingcenter Shoppyland Schönbühl[8] zu ermöglichen. Gegenstand dieser Arbeit ist die Entwicklung des AIONAV LBS Editors als Planungs-Applikation für die Marktleitung. Die mobile App wurde parallel dazu von AIONAV entwickelt. Obwohl der AIONAV LBS Editor nun für einen speziellen Anwendungsfall implementiert werden sollte, musste trotzdem darauf geachtet werden, die Anwendung nicht zu spezialisieren, da sie auch für spätere, weitere Anwendungsfälle geeignet sein sollte. Daraus ergab sich eine große Herausforderung bei der Planung und Implementierung. Während der Entwicklungszeit fielen in etwa im Monatsabstand mehrere Zwischen-Meetings mit Migros an, in welchen auch immer die aktuellsten Fortschritte von mobiler Applikation und AIONAV LBS Editor in Form von Prototypen oder zumindest als Screenshots bzw. Mockups präsentiert wurden.

4.2.1. Migros-spezifische Anforderungen

Da das Projekt *Einkaufen der Zukunft* die erste reale Umsetzung des AIONAV LBS Editors sowie der dazugehörigen mobilen App war, mussten auch gewisse Spezialanforderungen umgesetzt werden. Im Grunde sollte es möglich sein, alle Geschäfte, aber auch interessante Örtlichkeiten, wie Aufzüge, Toiletten, Parkplätze etc. im gesamten Shoppingcenter zu erfassen. Andererseits sollten innerhalb des Migros-Marktes, der eines der 80 Geschäfte im Shoppyland ist, möglich sein, einzelne Produkte zu erfassen. Sowohl Geschäfte, als auch Produkte sind deshalb im Sinne dieses Projektes Points of Interest.

4.2.2. Nutzung des AIONAV-Frameworks

Wie schon erwähnt, verfügt AIONAV über ein großes Applikations-Framework sowohl für den Desktop, als auch für den mobilen Bereich. Mit Hilfe des Frameworks können unter anderem Pläne visualisiert werden, Navigationsaufgaben und Positionierung durchgeführt werden und entsprechende Daten in einer Datenbank abgelegt werden. Wie auch schon erwähnt, werden Funktionalitäten zum Abbilden von realen Koordinaten auf Plandaten zur Verfügung gestellt. Eine wichtige Komponente zum Visualisieren und Manipulieren von Daten ist die Anzeigekomponente *LayeredView*. Sie wurde zur Umsetzung des Projekts häufig genutzt und wird im Folgenden beschrieben.

[8] http://www.shoppyland.ch/, Dezember 2012 - Das Shopyland ist ein großes Einkaufszentrum des Konzerns Migros mit über 80 Geschäften inklusive einem Migros-Markt

Anzeigekomponente - Layered View

Der LayeredView ist eine zentrale Anzeigekomponente des AIONAV-Frameworks. Es handelt sich dabei um die Erweiterung eines JComponents. Die Klasse JComponent ist Teil von Java[9]. Sie ist die Basis für alle Swing-Komponenten (sowohl Standard-Komponenten als Teil von Java, als auch selbst-erstellter Komponenten). JComponent enthält sowohl Funktionalitäten zum Behandeln von Events, als auch solche zum Zeichnen auf der Komponente. Einem LayeredView können beliebig viele *Layer* hinzugefügt werden. Ein Layer ist dabei eine Java-Komponente, auf welcher beliebige geometrische Elemente - Angefangen von Linien, Polygonen bis über BufferdImages - gezeichnet werden können. Dies geschieht unter Zuhilfenahme eines Graphics2D-Objektes, welches wiederum Bestandteil von Java ist[10]. Die Klasse Layer ist hierbei nur eine generische Basisklasse. Das Aionav Framework enthält Implementierungen von Subklassen von Layer, mit denen gewisse Komponenten gezeichnet werden können. Es gibt zum Beispiel die Klassen *TextLayer*, zum Anzeigen von Texten, die Klasse *GeoImageLayer*, zum Anzeigen von globalisierten Bildern, die Klasse *LayerOfShapes* zum Zeichnen von geometrischen Figuren, oder auch die Klasse *AreaLayer* zum Anzeigen gewisser Pläne, wie sie in Abschnitt 4.2.3 beschrieben werden. Der LayeredView kann beliebig viele solcher Layer anzeigen. In welcher Reihenfolge sie (übereinander) angezeigt werden, kann ebenfalls geregelt werden. Die Klasse LayeredView übernimmt außerdem gewisse Anzeige-Operationen auf den verschiedenen Layern, sodass z.B. zoomen oder verschieben der Layer mittels Maus- oder Tastaturgesten möglich ist.

Datenbank-Anbindung

Weiteres verfügt das AIONAV-Framework über Klassen, die sich um die Anbindung und Synchronisation einer lokalen Datenbank kümmern. D.h. um Daten aus der Datenbank zu holen oder hineinzuschreiben, stehen gewisse Klassen mit den jeweiligen Funktionalitäten zur Verfügung. Eine Besonderheit ist, dass die Datenbank lokal auf dem jeweiligen Gerät liegt und über Netzwerk synchronisiert werden kann. Dieser Mechanismus ist in vorherigen Projekten entstanden und bildet seither die Basis aller AIONAV-Anwendungen. Somit können Kunden der mobilen App die aktuellsten Daten leicht auf ihr Gerät laden und anschließend *offline* mit der App im Kaufhaus navigieren. Die Datenbank wurde außerdem wie in Abschnitt 4.2.4 angepasst bzw. erweitert.

4.2.3. Pläne einlesen und verarbeiten

Eine sehr wichtige Aufgabe bei der Entwicklung der Applikation war das korrekte Anzeigen von Plänen, die von Mirgos für das Shoppingcenter Schönbühl zur Verfügung gestellt wurden. Die Pläne sind einerseits Basis für das Einzeichnen von Points of Interest und Routen, andererseits sind sie Grundlage für die Navigation von Benutzern, da sie praktisch ein lokales Koordinatensystem über den Bereich, der für die Anwendung

[9] http://docs.oracle.com/javase/6/docs/api/javax/swing/JComponent.html, Februar 2013
[10] http://docs.oracle.com/javase/6/docs/api/java/awt/Graphics2D.html, Februar 2013

relevant ist, spannen. Von Migros wurden entsprechende Plandaten im XML-Format zur Verfügung gestellt. Pro Stockwerk eines Gebäudes gab es zwei Dateien, die anschließend beschrieben werden:

XML-Hauptdatei

Die erste XML-Datei, das Hauptfile des Planes, enthielt den Grundriss und einige elementare Gebäudeelemente (z.B. Grundmauern oder Infrastrukturinformationen). Der Aufbau des Dokuments ähnelt dem Plandaten-Format der Firma Speedikon[11], welche unter anderem in den Bereichen Flächen- und Gebäudemanagement tätig ist. Im AIONAV-Framework existierte bereits eine rudimentäre Implementierung zum Parsen und Verarbeiten dieser Plandaten, an der nur wenige Änderungen durchgeführt werden mussten, um die Grunddaten einlesen zu können. Das Endprodukt des Einlese-Vorganges ist ein virtuelles Gebäudemodell, welches sich in die folgenden Hauptkomponenten aufteilt:

- Region
- Building
- Floor
- Area

Diese Komponenten sind hierarchisch inklusive ihrer Standortdaten im XML-Dokument aufgeführt. Im AIONAV-Framework besitzen diese Komponenten nach erfolgtem Einlesen aus dem XML-Dokument eigene, spezifische Funktionen und ergeben zusammen ein Gebäudemodell. Eine Region beschreibt einen Bereich, in dem sich mehrere Buildings befinden können. Ein Building, welches eine Gebäudes beschreibt, kann dabei mehrere Floors (Stockwerke) enthalten, in welchen sich Areas befinden. Dies sind die kleinsten Elemente im Gebäude, und bilden meist Räume bzw. die Umrisse von Böden ab. Dieses System ermöglicht es, eines oder mehrere komplette Gebäude zu modellieren und die nötigen Grundfunktionen zum Navigieren darin zur Verfügung zu stellen. Schließlich kann ein eingelesenes Gebäudemodell mit Hilfe der Anzeigekomponente LayeredView angezeigt werden (siehe Abbildung 4.7). Dieses Gebäudemodell ist dadurch auch die Basis für den AIONAV LBS Editor, da es mit Hilfe des AIONAV-Frameworks auch möglich ist, den eingelesenen Plan zu globalisieren. Dies bedeutet, dass der Plan mit Geokoordinaten versehen wird. Somit kann für jeden Punkt im Plan seine genaue Position nach einem geographischen Koordinatensystem der Erde bestimmt werden. Diese Daten sind sowohl beim Einzeichnen von Points of Interest und Routen, als auch beim Navigieren auf dem Plan nötig. Weiters können Plandaten in der beschriebenen Form mit Hilfe des AIONAV-Frameworks in eine Datenbank gespeichert und später wieder ausgelesen werden. Pro Stockwerk wurde von Migros je eine XML-Hauptdatei zur Verfügung gestellt.

[11] http://www.speedikonfm.com/, Februar 2013

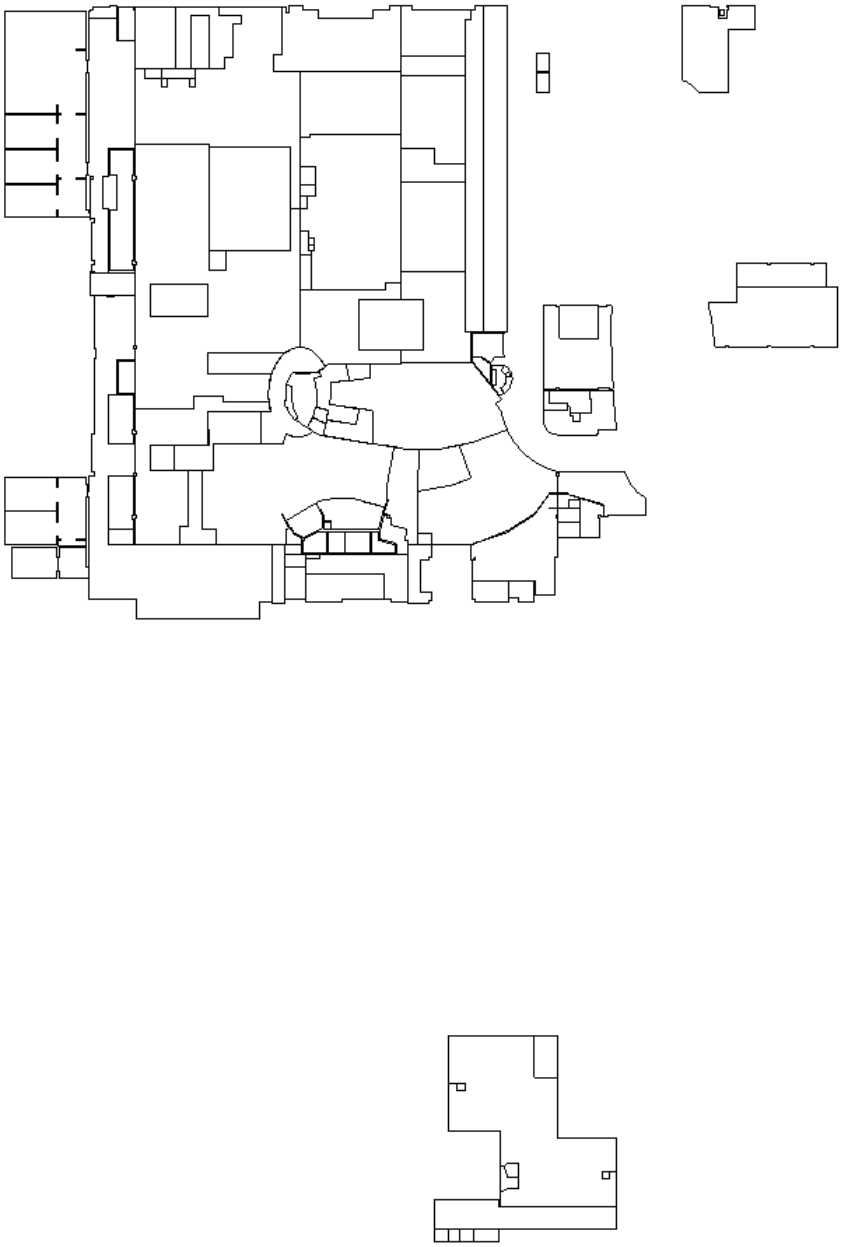

Abbildung 4.7.: Ansicht von eingelesenen Gebäudeinformationen aus einer von Migros zur Verfügung gestellten XML-Datei

B-Files

Zusätzlich zu jeder XML-Hauptdatei wurde von Migros eine weitere XML-Datei zur Verfügung gestellt. Aufgrund des Namensschemas, nach welchen diese Zusatzdateien im-

mer den selben Datei-Namen wie die oben beschriebenen Hauptdateien haben, aber mit der zusätzliche Endung "_b" versehen sind, werden diese Files im Folgenden *B-Files* genannt. Ein B-File enthält Informationen über geometrische Elemente (Linien, Polygone, Ellipsen etc.), welche Details eines Stockwerkes modellieren. Dabei handelt es sich unter anderem um Grundrissdaten eines Gebäudes, die noch nicht in der XML-Hauptdatei enthalten waren, wie etwa um ganze Räume, aber auch Durchgänge und Türen, oder um Rolltreppen, Aufzüge, Treppen oder auch Parkplätze. Außerdem wurden für den Migros-eigenen Markt im Shoppyland Schönbühl Regale, Stellagen und Kassenbereiche abgebildet. In einem B-File befinden sich somit viele Informationen zu Gegenständen und zur Gebäudeinfrastruktur. Die Informationen sind allerdings ohne jeglicher Semantik hinterlegt. Nach dem Einlesen dieser Daten ist deshalb noch erheblicher Aufwand nötig, um sie im Kontext dieses Projektes (z.B. als Point of Interest) auch nutzen zu können. Das Einlesen von B-Files wurde in Form eines eigenen Parsers umgesetzt, welcher die geometrischen Formen aus einem B-File ausliest und in eine Form umwandelt, in welcher sie in weiterer Folge als eigener Layer auf dem LayeredView angezeigt werden können. Bei der Umsetzung der Anzeige gab es kritische Performance-Probleme, da ein B-File mehrere Hunderttausend einzelne Formen enthält. Sowohl die interne Verarbeitung, als auch die Anzeige und damit zusammengehörige Aufgaben wie Zoomen und Verschieben, waren sehr rechenintensiv und mussten deshalb auf langwierigem Wege optimiert werden. Weiters wurde die Funktionalität geschaffen, B-Files in Dateien zu speichern bzw. sie wieder auszulesen. Abbildung 4.8 zeigt das gesamte Shoopyland-Areal, wie es durch die geometrischen Formen eines B-Files dargestellt wird. Abbildung 4.9 zeigt die Detailansicht in der Migros-Filiale. Besonders wichtig für die spätere Eingabe von Points of Interest sind die dort dargestellten Regale.

4.2.4. Datenbank planen und umsetzen

Als besonders wichtig in der Entwicklung des AIONAV LBS Editors erwies sich der Entwurf einer Datenbank, sowie die Implementierung der Zugriffslogik. Zum Speichern von Plänen, georeferenzierter Bildern und Positionsdaten bestand im AIONAV-Framework bereits eine Datenbank und die nötige Zugriffslogik. Diese Datenbank wurde nun erweitert. Ziel war nicht nur die Entwicklung im Sinne des Projektes *Shooping der Zukunft* in Zusammenarbeit mit Migros, sondern die Schaffung eines allgemeinen und variabel einsetzbaren Datenbankschemas. Deshalb wurden viele Konzepte auch sehr generisch umgesetzt und nicht spezifisch für dieses Projekt. Wie zum Beispiel die Einteilung von Points of Interest, die im Migros-Projekt Produkten, aber auch Geschäften entsprechen. Migros hat intern eine gewisse Vorgehensweise, Produkte in verschiedene Kategorien einzuteilen. Es existiert eine fixe Hierarchie von Produktkategorien. Diese Hierarchie besteht aus folgenden vier Stufen: Bedarfswelt, Bedarfsbereich, Verkaufssektor und Artikel. Bedarfswelten sind ausgezeichnete Flächen innerhalb eines Migros-Marktes. Bedarfsbereiche und Verkaufssektoren sind zumeist Flächen, die innerhalb von Bedarfswelten liegen können oder auch nur abstrakte Hierarchie-Stufen, die keine eigene Fläche haben, aber zur Visualisierung als Zwischenkategorien bestehen. Es sollte also möglich sein, Produkte im Migros-Markt, aber auch Geschäfte im gesamten Shoppyland, sowie andere wichti-

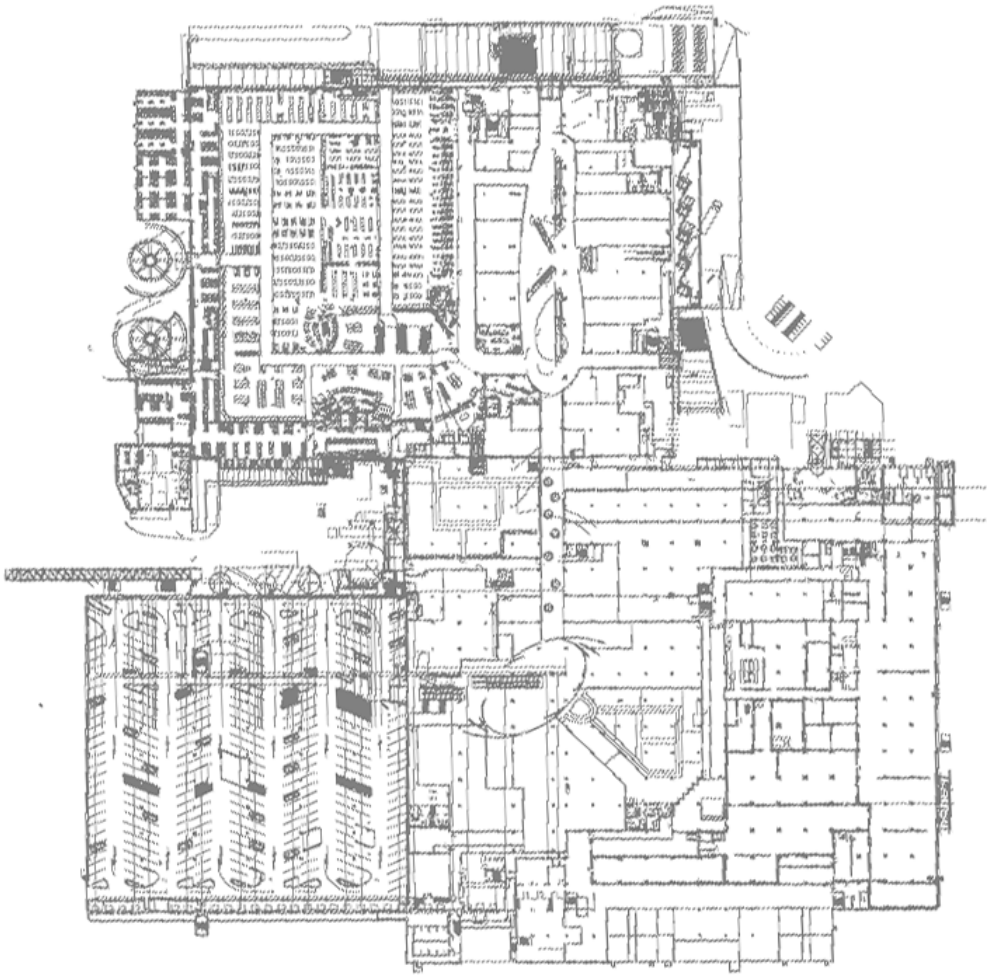

Abbildung 4.8.: Anzeige eines eingelesenen B-Files (Gesamtes Shoppyland-Areal)

ge Punkte zu erfassen. Darum wurde die Datenbank so erweitert, dass es beliebig viele Kategorien geben kann, wobei eine Kategorie immer genau eine Parent-Kategorie und beliebig viele Child-Kategorien haben kann. In einer Kategorie können wiederum beliebig viele Points of Interest liegen. Produkte, Geschäfte und andere Orte wurden schließlich in verschiedenen Kategorien abgebildet. Dadurch blieb die Anwendung weitestgehend generisch und kann für andere Einsatzzwecke weiterverwendet werden. Für das Projekt mit Migros ist der Root-Knoten der Kategorie-Hierarchie das Shopping Center selbst. Darunter hängen die Geschäfte in Form von Kategorien und zugehörigen Point of Interest. Im Migros-eigenen Markt hängen weitere Unterkategorien, die die Produkthierarchie wie oben beschrieben abbilden. Neben der Kategorisierung wurden für die Speicherung von Points of Interest die nötigen Funktionalitäten umgesetzt, um Eigenschaften, wie in Abschnitt 4.1.6 beschrieben, mit einem Point of Interest zu verknüpfen. Die wichtigsten

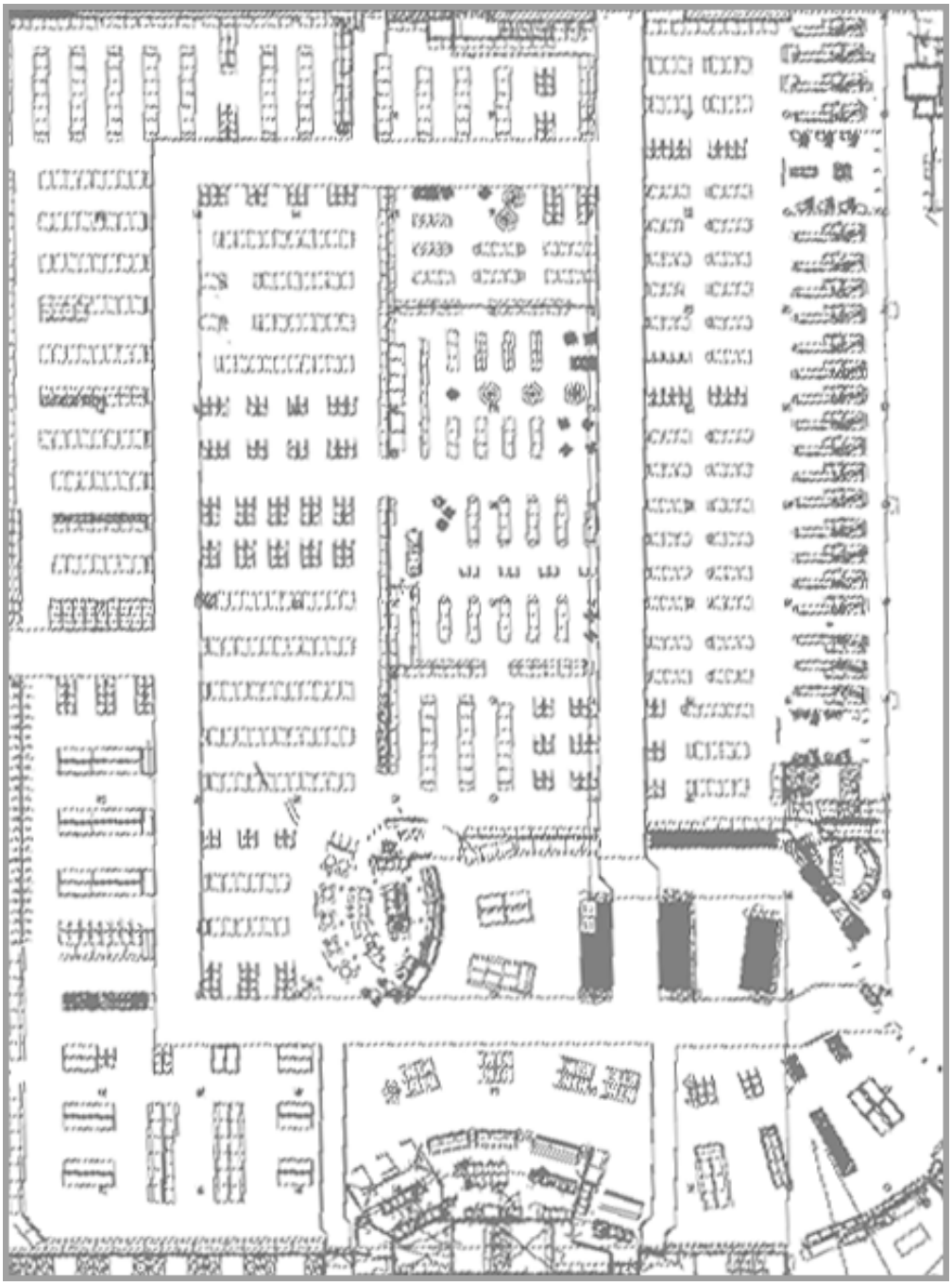

Abbildung 4.9.: Anzeige eines eingelesenen B-Files (nur Migros-Filiale mit Regalen und Kassenbereich)

neu entstandenen Datenbank-Tabellen sind:

POI

Die Tabelle *POI* ist die wichtigste neu hinzugefügte Datenbanktabelle. Sie enthält einerseits Einträge auf die wichtigsten Eigenschaften eines Point of Interest, wie etwa Beschreibungstext und Label, aber auch Verlinkungen (Foreign-Key-Relation) auf andere wichtige Tabellen, nämlich die POI-Kategorie, Priorität, Position und Aktivitätsfläche.

POI_CATEGORY

Zwischen der Tabelle *POI* und der Tabelle *POI_CATEGORY* gibt es eine 1-zu-1-Beziehung. Dies bedeutet, dass ein Point of Interest immer nur einer Kategorie zugeordnet werden kann. Ein Kategorie hat die Eigenschaften Label und Beschreibungstext. Außerdem kann eine Kategorie über die Intersection-Tabelle *POI_CATEGORY_INTERSECT* genau einer Parent-Kategorie zugeordnet werden, sowie beliebig viele Kind-Kategorien erhalten. Dadurch kann eine Hierarchie von Kategorien aufgebaut werden.

AREA_INFO und AREA_POSITION

Die Tabelle POI enthält zwei Referenzen auf die Tabelle *AREA_INFO*. Einmal für die POI-Position und einmal für die Aktivierungsfläche. Einträge in der Tabelle *AREA_POSITIONS* sind wiederum mit der Tabelle *AREA_INFO* verknüpft. Somit ist jedem Point of Interest sowohl eine POI-Position als auch eine Aktivierungsfläche jeweils als Fläche (*AREA_INFO*) in der Datenbank zugeordnet, wobei eine Fläche beliebig viele Punkte (*AREA_POSITIONS*) haben kann.

POI_PRIORITY, POI_ACTIONS, POI_STATES

Die Tabelle *POI* enthält weiters Referenzen auf die Tabelle *POI_PRIORITY*, mittels welcher einem Point of Interest eine gewisse Priorität zugeordnet werden kann. Eine Priorität hat ein Label, einen Beschreibungstext und eine Zahl zugeordnet, die zur Sortierung von Prioritäten dient. Beutzer können die Prioritäten selbst erstellen und administrieren. Weiters gibt es die Tabellen *POI_ACTIONS* und *POI_STATES*, die über die Intersection-Tabellen *POI_ACTION_TIMETABLE* und *POI_STATE_TIMETABLE* mit der Tabelle POI verbunden sind. Einem Point of Interest können somit verschiedene Aktionen und Status zugeordnet werden, wobei über die Timetable-Tabellen die Gültigkeit der jeweiligen Eigenschaften zeitlich begrenzt werden kann.

FILE_STORE

Zum Speichern und Verknüpfen von Dateien mit einem Point of Interest gibt es die Tabelle *FILE_STORE*. In ihr kann ein beliebiges *Binary Large Object* abgelegt werden. Zu jedem File existiert weiters ein Eintrag in der Tabelle *MIME_TYPE*, um es bei einem späteren Ladevorgang in den richtigen Datei-Typen umwandeln zu können. Ein Point of

Interest kann mit beliebig vielen Files verknüpft werden. Die Verbindung geschieht über die Tabelle *POI_FILE_STORE_INTERSECT*. In dieser Intersectiontabelle ist für jedes File außerdem ein Verweis auf einen Eintrag der Tabelle *POI_FILESTORE_FUNCTION*. Die File-Store-Function gibt an, wie ein File im Kontext einer Applikation verwendet werden kann. So kann beispielsweise ein Bild im File Store hinterlegt werden und zusätzlich bei der Verknüpfung die Funktion "Produktfoto" eingetragen werden. Somit kann die Applikation aus mehreren Files das Produktfoto herausfiltern und entsprechend anzeigen. Weitere Möglichkeiten wären die Hinterlegung eines PDF-Files als Flyer eines gewissen Geschäfts oder eines Produktvideos. Hier wurde bewusst generisch entwickelt, um die verschiedensten, zukünftigen Anforderungen abdecken zu können.

ADDITIONAL_INFO

Ein sehr generisches und dennoch mächtiges Konzept wurde außerdem mit der Tabelle *ADDITIONAL_INFO* eingeführt. Ein Point of Interest kann über die Intersection-Tabelle *POI _ ADDITIONAL _ INFO _ INTERSECT* mit beliebig vielen Additional Infos verknüpft werden. Ein Eintrag in der Tabelle Additional Info hat lediglich ein Label und einen Beschreibungstext. Somit können einem Point of Interest beliebig viele neue Eigenschaften in Textform zur Laufzeit hinzugefügt werden, ohne das dahinterliegende Datenschema zu verändern. Darüber hinaus ist in der Tabelle *POI _ ADDITIONAL _I NFO _ INTERSECT* eine Referenz auf die Tabelle *ADDITIONAL_INFO_FUNCTION* hinterlegt. Ähnlich wie bei der File Store-Funktion kann somit jedem Additional-Info-Eintrag eine Funktion zugeordnet werden. Das folgende Beispiel soll das Konzept nochmals veranschaulichen. Will man zum Beispiel einem Point of Interest eine Webadresse zuordnen, so muss nicht in der Datenbank ein neuer Eintrag dafür geschaffen werden, sondern lediglich die entsprechende Additional-Info-Funktion angelegt werden. Danach kann ein Additional-Info-Eintrag mit der Webadresse erstellt und mit dem Point of Interest verknüpft werden, wobei die neu erstellte Additional-Info-Funktion ebenso eingetragen wird. Um das Konzept noch flexibler zu machen, können Additional Infos auch optional zeitlich begrenzt werden. In der Tabelle *ADD_INFO_TIMETABLE* können Start- und Endzeitpunkt für die Gültigkeit eines Additional-Info-Eintrages gesetzt werden. Das Konzept der Additional Infos wurde im Laufe der Implementierung häufig verwendet und zeigte durch seine Flexibilität viele Vorteile. Unter anderem auch, weil von Seiten der Stakeholder häufig neue Anforderungen für Eigenschaften von Points of Interest kamen, wie zum Beispiel der Wunsch nach farbigen POI-Flächen. Um dies umzusetzen, musste lediglich ein Additional-Info-Eintrag mit der Funktion "Color" erstellt werden.

GRAPH_INFO

Die Tabelle *GRAPH_INFO* wurde entwickelt, um ein Wegenetz auf Plänen speichern zu können. Auf diesem Wegenetz soll es möglich sein, die Navigation von Kunden der mobilen App durchzuführen. Ein Eintrag in der Tabelle GRAPH_INFO steht stellvertretend für ein Linien-Netz, welches sich über einen Plan zieht. Mit *GRAPH_INFO* sind beliebig viele Einträge der Tabelle *GRAPH_EDGES* verknüpft, welche jeweils aus zwei

Einträgen (Start-und Endpunkt) in der Tabelle *GRAPH_POINTS* bestehen. D.h. ein Graph besteht aus beliebig vielen Graph-Edges mit Start- und Endpunkt. Eine weitere Anforderung, die jedoch in dieser Version des AIONAV LBS Editors noch nicht umgesetzt wurde, ist das Versehen von Graph-Edges mit Richtungs-Informationen. Dies ist für eine spätere Programmversion jedoch geplant.

PATH

Um vordefinierte Routen in der Applikation einzugeben, gibt es außerdem die Tabellen *PATH* und *PATH_POINTS*. Ein Path besteht dabei aus beliebig vielen Punkten der Tabelle *PATH_POINTS*. Eine Route kann zum Beispiel eine vom Marktleiter vordefinierte Einkaufsroute zu einer gewissen Thematik sein. Beispielsweise kann die Route mit dem Namen "Grillsaison" erstellt werden, welche an allen interessanten Orten in einem Plan entlang führt, die das Thema "Grillen" betreffen.

Die neu entstandenen Datenbanktabellen sind in Abbildung 4.10 visualisiert. Die Datenbank hat weitere Tabellen, die schon vor diesem Projekt bestanden. Unter anderem gibt es die zentrale Tabelle *LOCATION*. Die Location steht stellvertretend für ein ganzes Szenario. Ihr sind sowohl mehrere Pläne und georeferenzierte Bilder, wie auch alle Points of Interest und die damit zusammenhängenden Daten zugeordnet. Durch Auswahl einer gewissen Location kann somit ein Datensatz mit Plan und Points of Interest inklusive Zusatzinfo geladen werden.

4.2.5. Grafische Oberfläche

Wie bei vielen Applikationen war auch im Rahmen dieses Projektes die Umsetzung eines intuitiven und übersichtlichen grafischen User Interfaces erforderlich. Zur Gestaltung der Oberfläche wurden sowohl Java-Swing-Komponenten als auch Anzeigekomponenten aus dem AIONAV-Framework (siehe auch Beschreibung des *LayeredView* auf Seite 63) verwendet. Folgende Funktionen sollten vom User Interface zur Verfügung gestellt werden:

- Plandaten anzeigen
- Kategorien verwalten
  - Kategorie erstellen
  - Kategorie bearbeiten
  - Kategorie löschen
  - Kategorie-Hierarchie aufbauen bzw. verändern
- Point of Interest verwalten
  - Point of Interest erstellen
    - Eigenschaften eingeben
    - Position/Fläche einzeichnen

  - Eigenschaften und Position/Fläche bearbeiten
  - Point of Interest einer Kategorie zuordnen
- Routen einzeichnen
- Wegenetz einzeichnen
- Simulation eines Szenarios durchführen

Nicht alle Anforderungen konnten im Rahmen dieser Masterarbeit umgesetzt werden. Die Arbeit am Projekt wird jedoch darüber hinaus fortgesetzt. Die wichtigsten Grundfunktionalitäten, welche im Folgenden dokumentiert werden, wurden jedoch im Laufe der Arbeit implementiert.

Plan anzeigen

Um Pläne anzuzeigen, wurden die Plandaten, wie in Abschnitt 4.2.3 beschrieben, eingelesen und schließlich so aufbereitet, dass sie mit der LayeredView-Komponente des AIONAV-Frameworks angezeigt werden können. Abbildung 4.11 zeigt den Hauptbildschirm der Anwendung mit geladenen Plandaten und Kategoriebaum. (Anmerkung: In Anhang A zeigt die Abbildung A.1 den selben Inhalt wie Abbildung 4.11 nochmals aus Sicht er der mobilen Applikation).

Kategorien verwalten

Um Kategorien anzuzeigen und zu verwalten, wurde eine Baum-Anzeige implementiert, die in der App mehrmals verwendet wird (beim Erstellen und Editieren von Kategorien, beim Zuweisen eines Point of Interest zu einer Kategorie oder in der Hauptanzeige, wo eine Übersicht über alle Kategorien angezeigt wird). Abbildung 4.12 zeigt den Kategorie-Management-Dialog.

Point of Interest verwalten

Abbildung 4.13 zeigt den Dialog zum Erstellen und Editieren eines Point of Interest. Die relevanten Informationen können dort eingegeben und editiert werden.

Wegenetz verwalten

Um das Wegenetz für die Navigation zu verwalten, wurde eine eigene Ansicht entwickelt, in der beliebig viele Punkt-zu-Punkt-Verbindungen eingezeichnet werden können. Beim Einzeichnen von sich-überkreuzenden Verbindungen erkennt dies die Applikation automatisch und erstellt einen neuen Kreuzungspunkt. Abbildung 4.14 zeigt das fertig eingegebene Liniennetz für eine Präsentation der Applikation im Jänner 2013.

Anzeige bestehender Daten

Nach der Eingabe von Kategorien und Points of Interest können die Informationen natürlich auch entsprechend dargestellt werden. Abbildung 4.15 zeigt das Hauptfenster der Anwendung mit realen Testdaten.

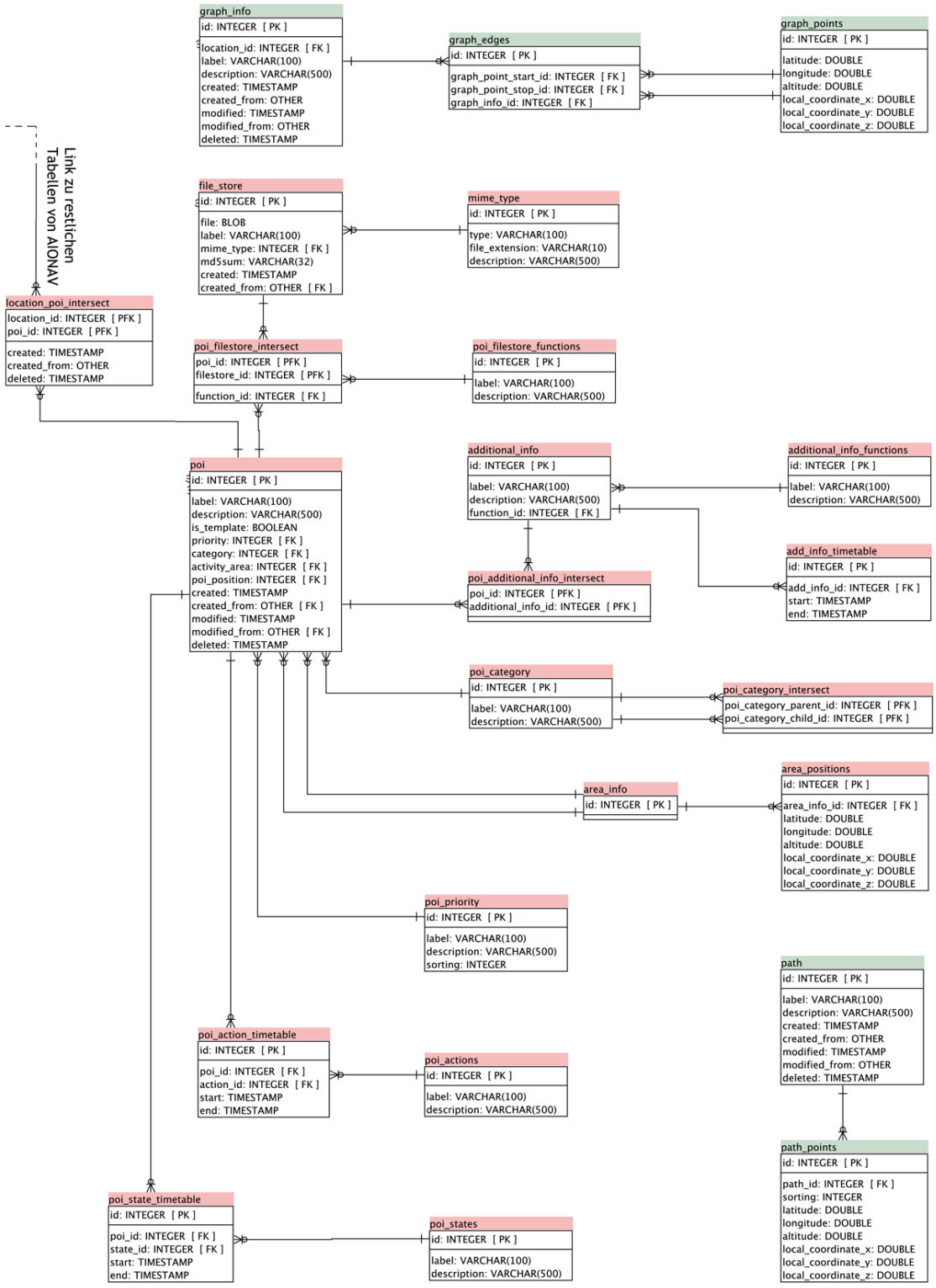

Abbildung 4.10.: Neue Tabellen im Datenbankschema

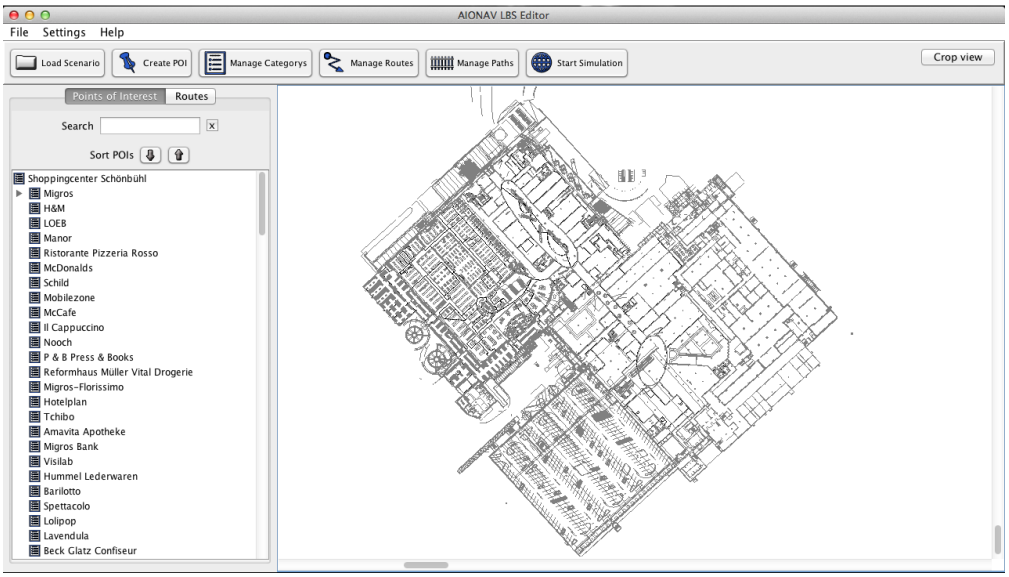

Abbildung 4.11.: Hauptbildschirm der Anwendung mit Plan von Migros Schönbühl (Mitte) und Kategorie-Baum (Links)

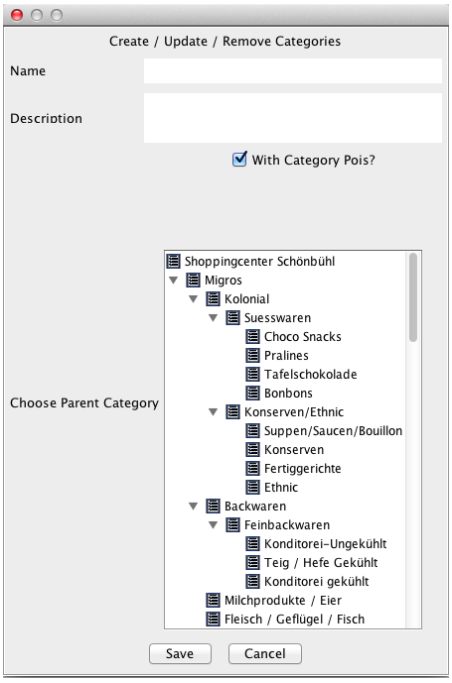

Abbildung 4.12.: Der Kategorie-Management-Dialog zum Eingeben und Editieren von Kategorien

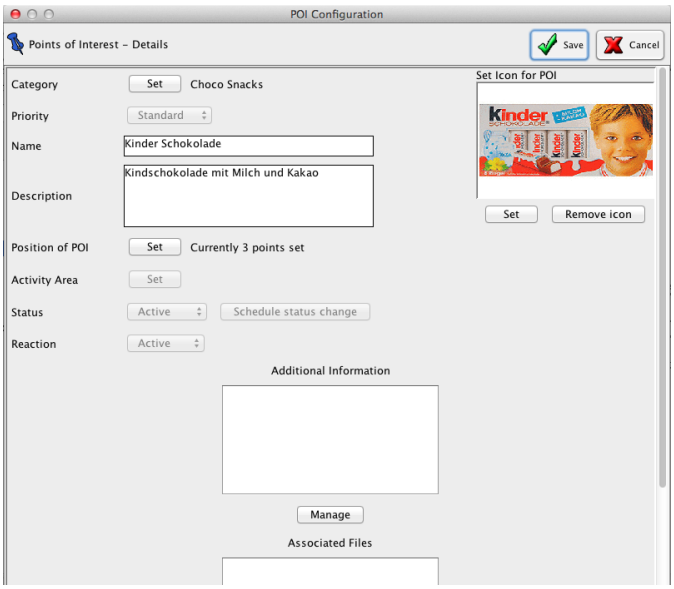

Abbildung 4.13.: Der Dialog zum Eingeben und Editieren von POI-Eigenschaften

Abbildung 4.14.: Die Ansicht zum Eingeben, Editieren und Löschen des Wegenetzes

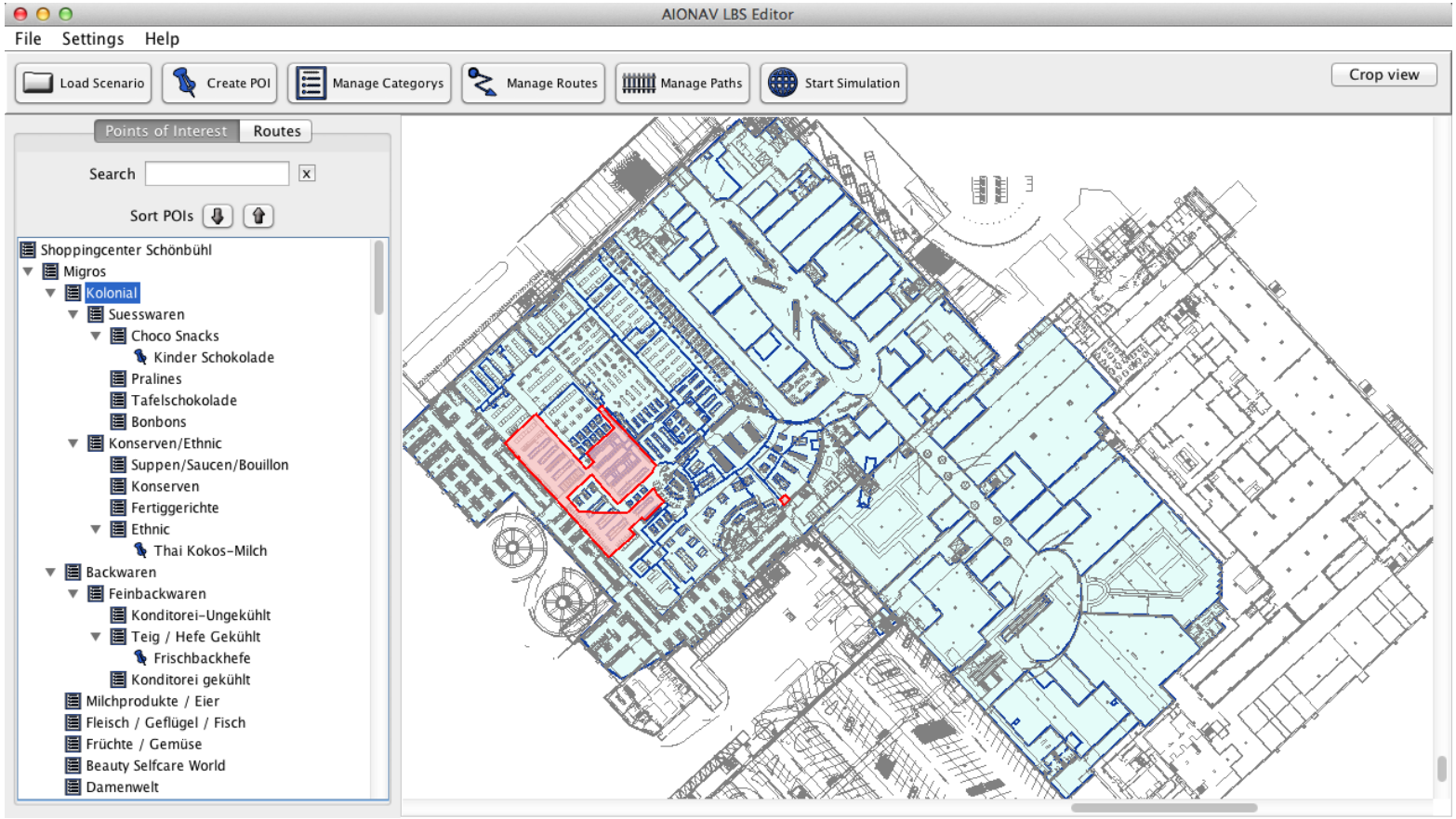

Abbildung 4.15.: Die Hauptansicht der Applikation. Es wurden alle Geschäfte des Erdgeschosses des Standortes in Schönbühl sowie exemplarisch einige Produkte im Migros-Laden eingegeben

5. Evaluierung des Einsatzes der vorgestellten agilen Methoden

Es folgt eine Evaluierung der in vorangegangenen Abschnitten vorgestellten Methoden der Softwareentwicklung. Probleme und Herausforderungen bei der Umsetzung werden aufgezeigt und die gefundenen Lösungsstrategien beschrieben.

5.1. Einsatz agiler Methoden

Das in Abschnitt 4 beschriebene Projekt wurde durch den Einsatz agiler Methoden entwickelt. Viele Vorgehensweisen des Extreme Programming wurden verwendet. Im folgenden Abschnitt werden die verwendeten Methoden, das Umfeld des Projektes und die beteiligten Personen beschrieben, sowie Erfahrungen mit den Vorgehensweisen und Problemstellungen erläutert.

5.1.1. Vorgehensweisen und Beschreibung des Arbeitsplatzes

Am Projekt beteiligte Personen

Der Prototyp des AIONAV LBS Editors wurde von einer Person im Rahmen dieser Masterarbeit entwickelt. Am gesamten Projekt waren jedoch einige weitere Personen beteiligt, die im Folgenden (siehe Tabelle 5.1.1) beschrieben werden. Die Projektleitung hatte Univ.-Prof. Dr.techn. Dipl.-Bauing. Ulrich Walder[1] über. Wie schon in Abschnitt 2.3.3 erwähnt, gibt es im Extreme Programming unterschiedliche Rollen innerhalb eines Entwicklerteams, welche großteils auch auf die beteiligten Personen dieses Projektes abbildbar sind.

Planning Game und Story Cards

Zu Projektbeginn wurden zusammen mit CEO und Chefentwickler in einem initialen Meeting die grundsätzlichen Anforderungen an die Anwendung, wie sie in Abschnitt 4.1 beschrieben sind, abgesteckt. Von vornherein war klar, dass die Anforderungen noch sehr veränderlich sind. Zudem war zu diesem Zeitpunkt noch gar nicht fixiert, dass ein Prototyp für Migros in naher Zukunft folgen sollte. Das Projekt war also von Anfang

[1] https://online.tugraz.at/tug_online/visitenkarte.show_vcard?pPersonenId=8EFCC24E04A3B791&pPersonenGruppe=3, Februar 2013

Rolle im Sinne des XP	**Rolle im Projekt**	**Beschreibung**
Big Boss	CEO AIONAV	Der Leiter des Projektes und zugleich CEO von AIONAV, Ulrich Walder, nahm im Sinne des XP die Rolle des Big Boss ein, da er als Hauptkoordinator des Projektes fungierte.
Kunde	Auftraggeber, AIONAV-Management, Chefentwickler	Das Projekt wurde ursprünglich von der Firma AIONAV initiiert, wodurch die Grundanforderungen an die Applikation geschaffen wurden. Darüber hinaus gab es mehrere Interessenten an einer spezifischen Umsetzung des Projektes. Schließlich wurde der Handelskonzern Migros erster richtiger Auftraggeber und Kunde für dieses Projekt. Kontakt zwischen AIONAV und Migros bestand vor allem durch eine Person aus dem Management von AIONAV, sowie dem CEO von AIONAV. Sowohl CEO als auch Management traten im Sinne des XP als Kunden im Projekt auf, da sie vergleichbar mit dem Konzept des Customer-Proxies (vergleiche Abschnitt 2.4.3), als Vermittlungsmänner zwischen Entwicklungsteam und realen Kunden standen.
Coach und Tracker	Chefentwickler AIONAV	Der Chefentwickler der Firma AIONAV kam in gewisser Weise als Coach (bzw. auch als Tracker) im Sinne des XP zum Einsatz. Er war einerseits Mentor und Mittelsmann zum Programmierer des Projektes, andererseits auch bei Meetings mit Migros anwesend und war somit die Person mit dem umfassendsten Projektverständnis.
Programmierer	Entwickler des AIONAV LBS Editors und weitere Entwickler von AIONAV	Die Programmierung des AIONAV LBS Editors wurde während dieser Arbeit von einer Person durchgeführt. Es gab jedoch auch weitere Entwickler der Firma AIONAV, die sich mit Parallel-Projekten beschäftigten (Datenbank-Entwicklung, Entwicklung der mobilen App). Trotz der Beschäftigung mit unterschiedlichen Projekten, gab es eine gemeinsame Basis, weshalb es viele Meetings aller Entwickler zusammen mit dem Chefentwickler gab.

Tabelle 5.1.: Beteiligte Personen des Projektes in verschiedenen Rollen des XP

an ideal geeignet für den Einsatz agiler Methoden, da sehr stark variierende Anforderungen zu erwarten waren. Der genaue Funktionsumfang des Endproduktes sollte sich im Laufe des Projektes herausstellen. Die Situation war gewissermaßen vergleichbar mit der der Projekte, die in Abschnitt 2.4.2 über die Auswirkungen von agilen Methoden auf Forschungssoftware nach *Sletholt et al.* [36] beschrieben wurden. Demnach konnten im diesem Zusammenhang durchaus positive Ergebnisse erwartet werden. Auf Basis der Daten der Anforderungsanalyse konnte schließlich ein erstes Planning Game durchgeführt werden. Dabei wurden initiale Story Cards erstellt, Prioritäten zugewiesen und der Aufwand der einzelnen Stories abgeschätzt. Die ersten Story Cards waren (sortiert nach ihrer Priorität):

1. Die Eigenschaften eines POI genau definieren
2. GUI planen und designen
3. Pläne ins Programm laden
4. Datenbank designen und entsprechende Hintergrund-Logik implementieren
5. Kategorien erstellen, ändern, löschen
6. POI erstellen, ändern, löschen
7. Route planen, ändern, löschen
8. Szenario abspeichern
9. Szenario laden
10. Szenario-Simulation

Die meisten Storycards mussten zur Umsetzung weiter in kleinere Aufgaben aufgespalten werden, da die Umsetzung einer Storycard ansonsten zu viel Zeit auf einmal in Anspruch genommen hätte. Die Story Card zum Einlesen von Plänen (für Details siehe Abschnitt 4.2.3) musste beispielsweise in die Aufgaben "Hauptfile laden", "B-File laden", "Hautpfile anzeigen"und "B-File anzeigen"aufgespalten werden. Die Aufgabe "B-File anzeigen"musste außerdem aus Performanz-Gründen viele Male refactored werden.

Mit fortschreitender Projektlaufzeit kamen außerdem weitere Anforderungen zum Projekt hinzu, die entsprechend als Stories in das Storyboard einsortiert wurden. Einige dieser zusätzlichen Anforderungen waren:

- Ein Wegenetz einzeichnen, um die Navigation von Benutzern zu ermöglichen
- Abbildung der Migros-eigenen Produkthierarchie
- Points of Interest erhalten eine größere Spannweite (Geschäfte, aber auch Produkte werden zu Points of Interest)

- Zusätzliche Eigenschaften von Points of Interest
- Standortdaten von Regalen aus B-Files auslesen
- Punkte aus B-Files klickbar machen, um die Eingabe der Positionen von neuen Points of Interest zu erleichtern

Die Storys wurden auf einem Storyboard, welches sich direkt am Arbeitsplatz des Entwicklers befand, aufgehängt (siehe Abbildung 5.1). Damit wurde der gängigen Methode im Extreme Programming, den Arbeitsplatz informativ zu gestalten, Folge geleistet. Wie die Vorgehensweisen des Extreme Programming in Rahmen dieses Projektes an die Situation als Einzelentwickler angepasst wurde, wird schließlich im Abschnitt 5.2 beschrieben.

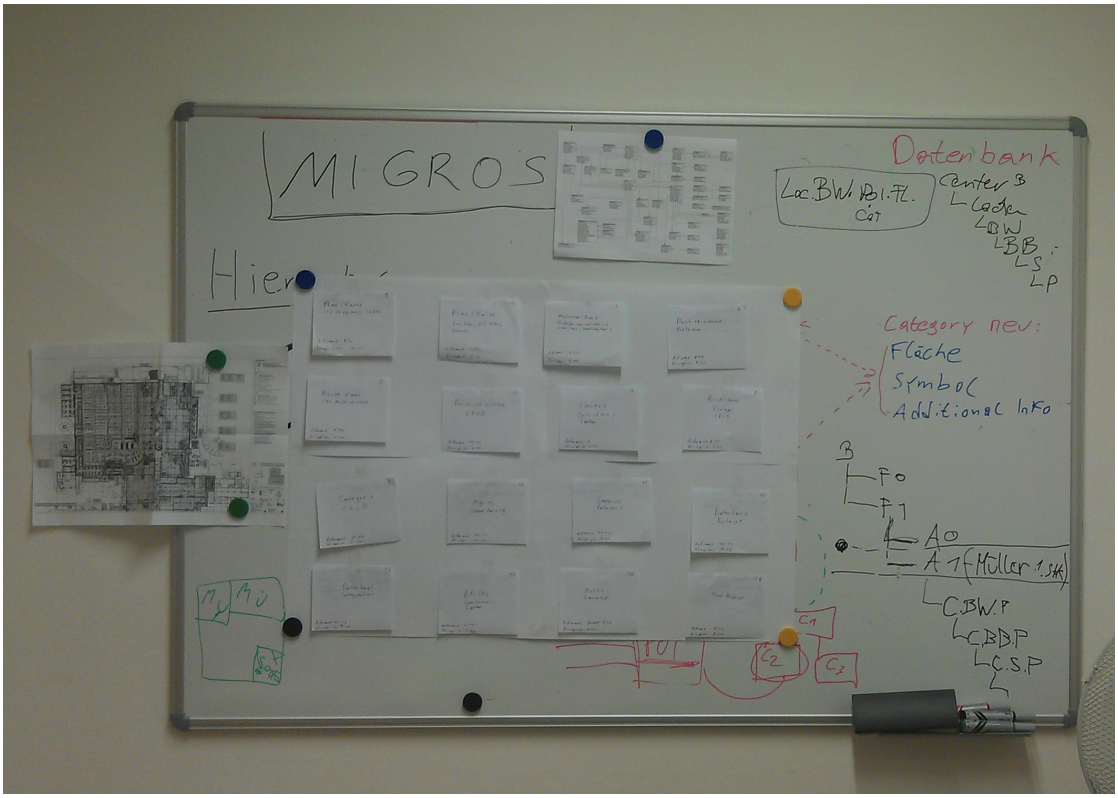

Abbildung 5.1.: Storyboard am Arbeitsplatz

Informationsfluss im Projekt

Ein wichtiges Erfordernis bei der Entwicklung im Sinne des Extreme Programming ist, dass der Kunde dauerhaft am Projekt beteiligt ist und regelmäßig Feedback zu aktuellen Releases abgibt. Dies war bei der Umsetzung dieses Projektes allerdings nicht

möglich. Die Auftraggeber hatten ihren Firmensitz in der Schweiz, die Entwicklung der Applikation fand jedoch in Graz statt. Dementsprechend wurden Informationen, die von Kundenseite kamen, an das Management mitgeteilt, über den Chefentwickler von AIONAV gefiltert und für die Entwickler des AIONAV LBS Editors und der mobilen App aufbereitet. Den Kunden wurden in regelmäßigen Abständen (4-6 Wochen) die Fortschritte des Projektes bzw. auch aktuelle Releases in Form von Prototypen präsentiert. Dies machte einige Anpassungen an die Vorgehensweise bei der Entwicklung nötig, da beim Extreme Programming letztendlich von einem Kunden innerhalb des Entwicklungsteams ausgegangen wird. Die genauen Anpassungen werden in Abschnitt 5.1.2 detailliert beschrieben.

Weitere verwendete Methoden

Da im Stile des Extreme Programming entwickelt wurde, wurden auch noch weitere entsprechende Methoden verwendet:

- **Test Driven Develoment:** Die Entwicklung des AIONAV LBS Editors erfolgte *testgetrieben*. Zur Unterstützung wurde das in Abschnitt 3.2.3 beschriebene JUnit-Framework eingesetzt. Die Umsetzung war für Hintergrund-Logik großteils problemlos möglich. Auf einige Probleme, besonders im Bezug auf das Testen von grafischen User Interfaces, wird in Abschnitt 5.3.1 nochmals detailliert eingegangen.

- **Zentraler Arbeitsplatz:** Entwickelt wurde an einem XP-geeigneten Arbeitsplatz. Da AIONAV als Spinoff aus dem Institut für Bauinformatik der TU Graz hervorgegangen ist, sind ein Großteil der Mitarbeiter auch Angestellte der TU Graz. Somit standen die Institutsräume für Entwicklungszwecke zur Verfügung. Die am Projekt beteiligten Personen konnten ohne große räumliche Trennung regelmäßig miteinander kommunizieren und kollaborieren.

- **Teamarbeit:** Auch wenn die Programmieraufgaben für den AIONAV LBS Editor weitgehend von einer Person ausgeführt wurden, so wurden doch gemeinsam mit Mitarbeitern der Firma AIONAV Planungs- und Designtätigkeiten ausgeführt. Die Anzahl der Teammitglieder variierte dabei immer zwischen 1 und 4 Personen.

- **Zyklen:** Die Einhaltung gewisser Zyklen (wöchentlich, monatlich) war ebenso gegeben. Da ohnehin etwa im Monatstakt Meetings mit dem Endkunden Migros abgehalten wurden, waren auch die Projekt-internen Abläufe darauf ausgelegt.

- **Continuous Integration:** Das Prinzip der Continuous Integration konnte während der Entwicklung des AIONAV LBS Editors problemlos eingehalten werden, da bei AIONAV ein Sourcecode-Versionierungstool im Einsatz ist, welches von allen Entwicklern der Firma AIONAV genutzt wird. Sobald neue Funktionen implementiert (und getestet) waren, wurden diese sofort in das Hauptsystem integriert.

5.1.2. Probleme im Zusammenhang mit agilen Methoden

Storycard-übergreifende Anforderungen

Durch die agile Vorgehensweise konnten neue Anforderungen durchgehend problemlos in das Projekt eingepflegt werden. Es zeigte sich allerdings auch, dass es Anforderungen gibt, die nicht ohne weiteres als Storycard abbildbar sind. Wie schon erwähnt, machte die Anzeige von Plänen (um genau zu sein die Anzeige von B-Files) Probleme. Besonders betroffen war die Gesamtperformanz des Systems, sobald eine große Anzahl von Grafikelementen angezeigt werden musste. Ein Problem, dass daraus resultierte, war die Zuordnung der Problematik zu einer gewissen Story. Im Grunde betraf das Problem eine gewisse Storycard, nämlich die zum Anzeigen von B-Files. Allerdings gab es an das System nun auch die neue Anforderung der Performanz-Optimierung. Da mit dem Performanz-Problemen auch Mängel in der Usability des Programmes entstanden (lange Reaktionszeiten auf Eingaben), zeigte sich, dass diese Problematik einen großen Einfluss auf das Gesamtprojekt und somit auf mehrere Stories hatte. Neben dem genannten Problem durch die große Menge an geometrischen Figuren, die anzuzeigen waren, kam ein weiterer Bug zum tragen, welcher sich ebenso stark auf die Performanz und Usability der Applikation auswirkte. Der Auslöser dieses Bugs war jedoch nicht direkt im Einflussbereich des Entwicklers des AIONAV LBS Editors , sondern in einem anderen Teil des AIONAV-Frameworks (im Bereich der Datenbanklogik) zu finden. So ergab sich die Situation, dass Performanz und Usability - zwei weitreichende Eigenschaften der Applikation - mehrere Storycards beeinflussten. Nach mehrmaligen Refactoring der entsprechenden Storycards und nach Auffinden des Bugs in der Datenbanklogik, konnte das Problem letztendlich gelöst werden. Aus dieser komplexen Situation heraus ergab sich schließlich die Erkenntnis, dass es gewisse Anforderungen gibt, die sich auch Storycard-übergreifend auswirken können und dass mit diesen entsprechend umzugehen ist. Konkret sollten also solch übergreifende Anforderungen wie Systemperformanz, Usability, Einheitlichkeit etc. bei der Umsetzung jeder Storycard beachtet werden.

Kein Kunde in unmittelbarer Nähe

Aufgrund der räumlichen Trennung der Kunden vom Entwicklerteam, war die Möglichkeit nicht gegeben, diese für ständige Rückmeldungen vor Ort zu haben. Dadurch ergaben sich im Sinne des Extreme Programmings natürlich Nachteile. Das Ausliefern von kleinen Releases in kurzen Zyklen war nicht möglich, das damit verbundene, schnelle Feedback blieb aus. Um dem entgegenzuwirken, wurden mehrere Maßnahmen gesetzt, wie sie auch schon im Abschnitt 2.4.3 (*nach Hoda et al.* [21]) erwähnt wurden: Einerseits fungierte der Chefentwickler von AIONAV, der zugleich als Coach im Sinne des XP tätig war, als *Ersatz-Kunde*, andererseits waren CEO und Management-Personen gewissermaßen in der Rolle der Customer-Proxies im Einsatz. Der Chefentwickler in der Aufgabe des Kunden, übernahm in diesem Sinne die Tätigkeit des Priorisierens von Storycards. Außerdem wurden ihm in regelmäßigen Abständen die neuesten Releases zur Evaluierung vorgelegt. Somit wurde auch in regelmäßigen Abständen Feedback an den Entwickler gegeben. Zusätzlich zu diesem Feedback fungierten auch der CEO und das Management

- beide in regelmäßigem Kontakt mit den realen Stakeholdern - als Kunden im Sinne des Extreme Programmings. Sie gaben ebenso Feedback, allerdings nicht in so regelmäßigen Abständen, da durch die räumliche Trennung zu den Auftraggebern und durch die zeitlichen Abstände zwischen Meetings, eine gewisse Verzögerung zustande kam. Hinzu kam die Anforderung, dass die Anwendung ohnehin generisch gehalten werden sollte, um sie auch für Einsatzzwecke, die über das Projekt mit dem Handelskonzern Migros hinausgehen, nutzbar zu machen. Diese Anforderung war unabhängig von externen Kunden und konnte somit intern vom Chef-Entwickler betreut werden.

Zeitmanagement, falsche Abschätzungen

Aufgrund der erwähnten Entfernung zwischen Kunden und Entwicklern, kam es vor der Präsentation eines Releases zu einem vorübergehenden, zeitlichen Engpass. Einige Anforderungen mussten unter Verletzung des Prinzips des *ausgeruhten Arbeitens* (Stichwort 40-Stunden-Woche) bis zum Zeitpunkt der Präsentation noch jedenfalls refactored werden. Die Anforderungen waren einerseits erst relativ knapp vor dem Präsentations-Zeitpunkt klar geworden, andererseits waren einige Funktionen vom Ausmaß her nicht richtig abgeschätzt worden, wodurch ihre Implementierung länger als erwartet dauerte (Gründe dafür finden sich im Abschnitt 5.2 über den Einzelentwickler-Ansatz). Um den Erfolg des Projektes und von Folgeprojekten nicht zu gefährden, wurde diese Übertretung des XP-Prinzips einmal bewusst durchgeführt. Es wurden jedoch auch Schlüsse aus der Situation für darauffolgende Zyklen gezogen, um derartige Situationen zu vermeiden.

Unterschiedliche Arbeitszeiten der Entwickler und Collective Ownership

Bei der Umsetzung des Projektes wurde mit mehreren Entwicklern der Firma AIONAV kollaboriert. Es gab regelmäßige Team-Meetings, wobei ein Entwickler für das AIONAV-Framework zuständig war, einer für die Umsetzung der mobilen App und ein anderer für die Datenbankschnittstelle und Zugriffslogik. Auch wenn regelmäßig Meetings stattfanden, stellte es sich als problematisch heraus, dass der zuständige Entwickler für Datenbanklogik während der Umsetzung des Projektes nicht Vollzeit, sondern nur für 15 Stunden pro Woche beschäftigt war. Da er für Teile der Sourcecodes alleine zuständig war, ergaben sich im Laufe des Projektes Probleme im Bezug auf Collective Ownership. Wie schon in Abschnitt 5.1.2 erwähnt, gab es mehrmals Bugs in der Datenbanklogik, die sich auf die Performanz und Funktionalität des AIONAV LBS Editors auswirkten. Da der zuständige Entwickler teilweise nicht vor Ort war, war es schwer, diese Bugs zu lokalisieren bzw. zu beheben. Aufgrund dieser Erfahrungen zeigte sich, welche Vorteile das Konzept der Collective Ownership gegenüber Systemen mit aufgesplitteten Zuständigkeitsbereichen hat. Darüber hinaus wurde auch der Schluss gezogen, dass eine Abstimmung der Arbeitszeiten von verschiedenen Teammitgliedern eine äußerst wichtige Aufgabe ist.

5.2. Einzelentwickler-Ansatz mit Extreme Programming

Dass der Ansatz des Extreme Programming nicht gänzlich ohne Probleme von Einzelentwicklern angewendet werden kann, wurde schon in Abschnitt 2.3.4 auf Basis des *Personal Extreme Programmings* nach *Agarwal et al.* [2] erläutert. Zwar konnten viele Methoden, wie Test Driven Development, Planning Game, Erstellen von Storycards oder Continuous Integration problemlos übernommen werden, doch ergaben sich auch bei einigen Methoden, wie etwa Pair Programming und Collective Ownership einige Probleme. Im folgenden Abschnitt wird darauf eingegangen, welche spezifischen Probleme es bei der Umsetzung von Extreme Programming als Einzelentwickler im Rahmen dieser Arbeit gab und welche Lösungen (in Anlehnung an *Agarwal et al.* [2]) gefunden wurden, um entstandene Nachteile auszugleichen.

5.2.1. Rollenverteilung und Teamarbeit

Wie schon in Abschnitt 5.1.1 erläutert, übernahmen Mitarbeiter der Firma AIONAV im Rahmen dieses Projektes die Rollen von Personen, wie sie bei der Entwicklung im Sinne des Extreme Programming üblich sind (Big Boss, Coach, Kunde, Programmierer). Dadurch und durch die Zusammenarbeit mit anderen Entwicklern der Firma, konnte eine dem Extreme Programming sehr ähnliche Team-Situation geschaffen werden. Einzig die Programmierarbeit wurde von einer Person alleine übernommen.

5.2.2. Pair Programming

Offenkundig ist es nicht möglich, als Einzelperson Pair Programming durchzuführen. Zu Zwecken des Code-Reviews oder auch zur Vorbesprechung von geplanten Tasks standen jedoch jederzeit Entwickler der Firma AIONAV zur Verfügung. Darüber hinaus erfolgten auch kurzzeitige Pair Programming - Sessions zusammen mit dem Chefentwickler von AIONAV, vor allem bei herausfordernden Aufgaben während der Verwendung von Komponenten des AIONAV-Frameworks.

5.2.3. Collective Ownership

Durch regelmäßige Code-Reviews sowie Vorbesprechungen mit dem Chefentwickler von AIONAV wurde gesichert, dass das Wissen über Implementierungsdetails jeder Zeit über mehrere Personen verteilt war.

5.2.4. Aufwandsabschätzung

Besondere Probleme ergaben sich bei der Aufwandsabschätzung von gewissen Storycards. Wie schon in Abschnitt 5.1.2 erläutert, gab es einen einmaligen zeitlichen Engpass, der unter anderem auf fehlerhafte Aufwandsabschätzung zurückzuführen war. Dieses Problem entstand letztendlich, weil die Aufwandsabschätzung nur von einer Person alleine getroffen wurde. Hier zeigte sich der mutmaßlich größte Schwachpunkt beim Einzelentwickler-Ansatz. Hätte es in diesem Fall mehrere Entwickler für den AIONAV

LBS Editor gegeben, wäre der Aufwand möglicherweise richtiger abgeschätzt worden. Außerdem wären bei größerer Entwickleranzahl im beschriebenen Fall der Zeitknappheit mehrere Programmierer zur Verfügung gestanden, um die ausständigen Anforderungen umzusetzen (es handelte sich um mehrere voneinander unabhängige Storycards). Zu bedenken ist auch, dass das gleichzeitige Bearbeiten von mehreren Storycards durch eine Person eine große Angriffsfläche für Fehler bietet, beispielsweise durch den häufigen Wechseln des Fokus' zwischen verschiedenen Aufgaben.

5.3. Testen Driven Development

Im Rahmen der Entwicklung des AIONAV LBS Editors wurde die Methodik des Test Driven Developments angewandt. Vor dem Schreiben von eigentlichem Programmcode wurden die nötigen Test-Cases unter Zuhilfenahme des JUnit-Test-Frameworks (siehe 3.2.3) angefertigt. Für neuentstandenen Code (im Bereich der Hintergrundlogik) konnte dadurch eine relativ hohe Testabdeckung bezüglich des Testkriteriums *Statement Coverage* erreicht werden. Nahezu jede neugeschriebene Methode hatte einen zugehörigen Test (einige wenige Ausnahmen bildeten etwa die Main-Methode der Applikation oder gewisse GUI-Erzeuger-Methoden, wie sie in Abschnitt 5.3 noch erläutert werden). Probleme ergaben sich allerdings bei der Erweiterung und Nutzung von externen Komponenten bzw. von Komponenten aus dem AIONAV-Framework.

Ungetestete Komponenten

Für die Umsetzung des AIONAV LBS Editors mussten auch gewisse Komponenten des AIONAV-Frameworks genutzt werden. Das AIONAV-Framework ist in jahrelanger Arbeit entstanden und nicht mit agilen Methoden entwickelt worden. Demnach existieren auch einige ungetestete bzw. nur teilweise getestete Komponenten. Da der Aufwand, alle verwendeten Komponenten des Frameworks im Nachhinein komplett mit Tests zu versehen, zu hoch gewesen wäre, wurden teilweise Black-Box-Tests geschrieben, mit denen Komponenten nur auf ihre grundsätzliche Funktionsweise getestet wurden. So bestand der Parser zum Einlesen von Plandaten (siehe 4.2.3) bereits teilweise. Die entsprechenden Klassen mussten nur noch um einige Funktionalitäten erweitert werden. Um die bestehenden Teile zu testen, wurden Test-Cases geschrieben, die den Parser mit vordefinierten Test-Plänen aufrufen. Der Output des Parsers wurde schließlich mit bereits bekannten, validierten Ergebnissen verglichen. Es wurden also nicht die einzelnen Unterfunktionen des Parsers getestet, sondern nur noch der Parser als Ganzes. Dies soll exemplarisch den Umgang mit derartigen Situationen zeigen.

Datenbank

Parallel zur Entwicklung des AIONAV LBS Editors wurde auch die entsprechende Datenbank designed und entwickelt (siehe Abschnitt 4.2.4). Die Implementierung der Zugriffslogik wurde jedoch zu einem bestimmten Teil von einem Entwickler der Firma AIONAV außerhalb des Rahmens dieser Arbeit vorgenommen. Dies geschah jedoch nicht unter

Verwendung agiler Methoden. Ebenso lagen keine Unit-Tests für die Datenbanklogik vor. Auch in diesem Fall wäre eine nachträgliche Implementierung von Unit-Tests mit erheblichem Aufwand verbunden gewesen und hätte den Rahmen dieses Projektes gesprengt. Zudem ist das Testen einer Datenbank und der damit verbundenen Zugriffslogik oft eine große Herausforderung, weil die Datenbank selbst meist vom Sourcecode entkoppelt ist. Eine Testklasse, welche durch exemplarische Aufrufe der Datenbanklogik die Funktionalität testet, lag jedoch vor. Wie sich im Laufe des Projektes herausstellte, gab es aus diesem Grund auch immer wieder kleinere Probleme mit Fehlern in der Datenbank-Zugriffslogik. Auch die in Abschnitt 5.1.2 erläuterten Probleme hängen mit dieser Problematik zusammen. In diesem Zusammenhang zeigten sich besonders die Vorteile von durchgehend getestetem Code.

Testen von GUI-Erzeugern

Im Rahmen dieses Projektes zeigte sich, dass es oft schwierig war Methoden zu testen, welche sich nur um das Erstellen und Initialisieren von GUI-Elementen kümmern. Diese Methoden haben oftmals keine direkt sichtbaren Ergebnisse, vor allem wenn GUI-Elemente erstellt werden, welche erst zu einem späteren Zeitpunkt sichtbar werden. Wenn möglich, wurde im Nachhinein zumindest überprüft, ob die GUI-Elemente existierten und nicht dem Wert *null* entsprachen.

5.3.1. Testen des grafischen User Interfaces

In Abschnitt 3 wurde unter anderem auf das Testen von User Interfaces und damit verbundenen Problemen und Herausforderungen eingegangen. Weiters wurden verschiedene Test-Frameworks vorgestellt und miteinander verglichen. Im folgenden Abschnitt wird nun aufgezeigt, wie die theoretischen Inhalte, die in Anschnitt 3 erarbeitet wurden, im Laufe dieser Arbeit zum Einsatz kamen. Weiters wird auf Projekt-spezifische Probleme und Herausforderungen beim Umgang ausgewählter Test-Frameworks eingegangen.

Wahl eines geeigneten Test-Frameworks

Um ein geeignetes Tool zur Durchführung von sinnvollen GUI-Tests zu finden, mussten einerseits zuerst die Anforderungen an solch ein Tool geklärt werden, so wie bestehende Tools analysiert und miteinander verglichen werden. Diese Schritte wurden in den Abschnitten 3.3.1 und 3.3.2 dokumentiert. Die Anforderungen an ein geeignetes Test-Framework waren im Rahmen dieses Projektes die Folgenden:

- **Kompatibilität zu agilen Methoden, vor allem Test Driven Development**
- **Unentgeltliche Nutzbarkeit**
- **Nach Möglichkeit Kompatibilität zu JUnit**

Dadurch schränkte sich die Auswahl der verfügbaren Test-Frameworks auf UISpec4J, jfcUnit und FEST ein. Nach weiterer Filterung fiel auch jfcUnit aus der Auswahl, da die

Entwicklung schon im Jahr 2004 eingestellt wurde und somit Kompatibilitätsprobleme mit der verwendeten Java-Version auftraten. Ebenso zu Problemen kam es bei ersten Tests mit dem FEST-Framework. Das AIONAV-Framework und somit auch der AIONAV LBS Editor erstrecken sich über mehrere Java-Projekte, was dem Fest-Framework schon beim Start der Applikation Probleme bereitete. Ebenso waren gewisse Funktionen des FEST-Frameworks nicht mit der Anzeigekomponente LayerdView kompatibel. (Anmerkung: auch Pounder und Abbot wurden auf die Verwendbarkeit hin getestet, hatten aber genauso Probleme mit der Verarbeitung des LayeredViews). Also kristallisierte sich UISpec4J als ideales Framework für GUI-Tests im Rahmen der Entwicklung des AIONAV LBS Editors heraus.

Wahl geeigneter Testkriterien und der Testabdeckung

Wie schon in Abschnitt 3 erwähnt, gibt es die verschiedensten Herangehensweisen, um die nötige Testabdeckung anhand gewisser Testkriterien zu erreichen. Bei der Implementierung im Rahmen dieser Arbeit, stellte sich bald heraus, dass viele Test-Cases sich nur darauf bezogen, dass durch das Auslösen eines GUI-Events, gewisse andere GUI-Elemente aktiviert oder deaktiviert bzw. aus- und eingeblendet wurden. Diese Eigenschaften konnten aber genauso gut über bestehende Zugriffsmethoden des Swing-Frameworks in herkömmlichen Unit Tests überprüft werden. Nach weiterer Analyse kristallisierten sich letztendlich einige zentrale Test-Cases heraus, welche die Möglichkeiten des entsprechenden GUI-Test-Frameworks UISpec4J ausnützen:

- **Erstellen, Ändern und Löschen von POIs**
- **Erstellen, Ändern und Löschen von Kategorien**
- **Erstellen, Ändern und Löschen eines Wegenetzes**

In allen drei Fällen werden Daten im User-Interface eingegeben, die anschließend in der zugehörigen Datenbank abgelegt werden. Es geht also hauptsächlich um die korrekte Funktionsfähigkeit von Dialogen zur Dateneingabe. Der korrekte Programmablauf kann schließlich getestet werden, indem überprüft wird, ob eingegebene Daten später auch tatsächlich entsprechend in der Datenbank abgebildet wurden. Im Folgenden (siehe Abbildung 5.2) wird exemplarisch gezeigt, wie ein Test für das einfache Erstellen eines POIs aussieht. Dabei wird in der Methode *testCreateSimplePoi()* erst mittels *getMainWindow()* das Hauptfenster der Anwendung abgefangen. Schließlich wird ein entsprechendes Event auf dem Button zum Öffnen des *POI-Configuration-Dialoges* ausgelöst. Sobald dieser Dialog geöffnet ist, werden dort eine Kategorie ("NONE") und der Name des POIs ("Auto POI") eingetragen. Schließlich wird der Klick auf den Button zum Speichern ("Save") ausgelöst. Abschließend wird mit dem Aufruf der Hilfsmethode *Util.TestHelper.databaseContains()* überprüft, ob der POI auch tatsächlich in die Datenbank gespeichert wurde.

Auf diese Art und Weise wurden GUI-Tests für die wichtigsten GUI-Komponenten (im allgemeinen wichtige Dialoge im Programm) implementiert. Diese Tests wurden

```
@Test
public void testCreateSimplePoi() {

final Window mainWindow = getMainWindow();

WindowInterceptor.init(mainWindow.getButton("Create Poi").triggerClick())
    .process(new WindowHandler() {
            public Trigger process(Window dialog) {
                assertTrue(dialog.titleEquals("POI Configuration"));

                WindowInterceptor.init(dialog.getButton("btnChangeCategory").triggerClick())
                .process(new WindowHandler() {
                    public Trigger process(Window dialog2) {
                        assertTrue(dialog2.titleEquals("Category Configuration"));
                        dialog2.getTree().select("NONE");

                        return dialog2.getButton("Ok").triggerClick();
                    }
                }).run();

            assertTrue(dialog.getTextBox("lblSelectedCat").getText().equals("NONE"));
            dialog.getTextBox("txtFieldLabel").setText("Auto POI");
            return dialog.getButton("Save").triggerClick();
    }
}).run();

assertTrue(Util.TestHelper.databaseContains("Auto POI", "NONE"));

}
```

Abbildung 5.2.: Ein GUI-Test-Case, welcher einen neuen POI anlegt und auf seine Existenz in der Datenbank abprüft

in weiterer Folge wie herkömmliche Unit Tests behandelt und konnten somit in die bestehende Test Suite für den AIONAV LBS Editor eingebunden werden.

Herausforderungen und Probleme

In diesem Abschnitt wird auf Herausforderungen und Probleme eingegangen, die bei der Umsetzung dieses Projektes identifiziert wurden.

Überforderung der Test-Frameworks Wie schon erwähnt, gab es zumindest bei den nicht-kommerziellen Frameworks teilweise Probleme beim Starten des komplexen AIONAV-Frameworks bzw. auch bei der Verwendung der AIONAV-eigenen Anzeigekomponente LayeredView. Gründe dafür sind, dass einige der Frameworks schlicht keine Möglichkeiten boten, komplexere Java-Projekte (z.B. mit Virtual-Machine-Parametern) zu starten oder die dafür benötigten Vorgehensweisen wenig bis gar nicht dokumentiert waren. Vor allem Frameworks mit Visible Execution und Capture/Replay-Funktionalität gerieten außerdem mit von AIONAV personalisierten Anzeigekomponente an ihre Grenzen (so standen z.B. gewisse Event-Informationen, wie Klick-Koordinaten, nicht zur Verfügung). UISpec4J hingegen konnten ohne weitere Probleme genutzt werden.

Test Driven Development Die Umsetzung von Test Driven Development ließ sich auch im Bezug auf GUI-Tests überraschend einfach umsetzten. UISpec4J bot alle nötigen Funktionen, um Test-Cases für User Interfaces schon im Vorhinein zu schreiben. Allerdings zeigte sich im Rahmen der Umsetzung, dass für das Schreiben von GUI-Test-Cases zumindest grobe Pläne über das Aussehen und Design eines Userinterfaces vorliegen müssen. Außerdem mussten die Namen für gewisse GUI-Elemente schon beim Erstellen der Tests vergeben werden.

Sinnvolle Tests finden Im Kontext dieses Projektes zeigte sich, dass sich viele Tests nur auf das korrekte Setzen des Status' bestimmter GUI-Elemente bezogen. Nicht selten hatte in solchen Fällen der Test mit allen nötigen Aufrufen mehr Codezeilen, als die eigentlichen Codeabschnitte. Es stellt eine gewisse Herausforderung dar, relevante Test-Cases zu identifizieren, die über Aufrufe des bestehenden Swing-Frameworks hinausgehen.

Reaktion auf Redesign Ein weiteres Problem, das sich zeigte, war, dass sich viele GUI-Tests nur unter erheblichen Aufwand an ein Redesign des eigentlichen User Interfaces anpassen ließen. Dies ist vor allem der Fall, wenn nicht nur Layout- sondern auch Workflow-Änderungen vorgenommen werden. Ein Beispiel wäre, die Änderung eines User Interfaces dahingehend, dass für die Erfüllung einer Aufgabe plötzlich eine komplett andere Klick-Abfolge nötig ist, welche sich möglicherweise auch über viele verschiedene Dialoge erstreckt. Es ist außerdem vorstellbar, dass die Notwendigkeit von Testanpassungen einen Unwillen zu solchen Implementations-Änderungen bewirkt.

5.4. Fazit

Es wurde ein Praxisprojekt unter realen Bedingungen umgesetzt. Der AIONAV LBS Editor war als Tool konzipiert, welches innerhalb eines Frameworks von Applikationen eine wichtige Aufgabe übernehmen sollte. Die Anforderungen an die Anwendung waren teilweise sehr variabel oder entstanden erst während der Umsetzung und aufgrund des Feedbacks von Kunden. Die Anwendung von agilen Methoden der Softwareentwicklung bot sich gerade deshalb an, weil sie es ermöglichen, auf veränderliche Anforderungen zu reagieren. So wurde versucht, die Stärken der agilen Methodik des Extreme Programming auszunutzen um eine stabile, gut wartbare und vor allem ihren Anforderungen entsprechende Anwendung zu entwickeln. Nicht alle Bedingungen waren perfekt. Die Entwicklung wurde von einer Person durchgeführt. Es gab weitere Teammitglieder, deren Aufgabenbereiche sich jedoch nur teilweise mit der Umsetzung des hier beschriebenen Projektes überschnitten. Auch die Umsetzung des Test Driven Developments war erschwert, da einige ungetestete Framework-Komponenten genutzt werden mussten. Ebenso musste ein User Interface ansprechend designed, umgesetzt und auch getestet werden. Es gab unzählige **Problemstellungen**, **Anforderungen** sowie **Herausforderungen**. Durch die **Lösung** von Problemen, durch die **Umsetzung** von Anforderungen und durch das **Meistern** von Herausforderungen, wurden aber auch viele **Erfahrungen** gesammelt

und dokumentiert. Agile Methoden wurden erarbeitet, verstanden und umgesetzt. Sie wurden aber auch kritisch hinterfragt und durchleuchtet. In einigen Fällen mussten Anpassungen an die realen Gegebenheiten vorgenommen werden. Nicht immer konnte die Theorie problemlos in der Praxis angewendet werden. In gewisser Weise musste *agil* gearbeitet werden, um die *agilen Methoden* auch in der Praxis umsetzen zu können. Letztendlich zeigte sich aber doch, welche enormen Vorteile agile Methoden der Softwareentwicklung haben, wenn sie richtig angewendet werden. Der Verlauf des Projekts war ein großer Erfolg. Ein Prototyp des AIONAV LBS Editors wird zusammen mit dem mobilen Gegenstück dazu Ende März 2013 von AIONAV ausgeliefert und das Projekt wird über den Rahmen dieser Arbeit hinaus fortgesetzt. Dank des Einsatzes von agilen Methoden konnte ein gutes Stück Software geschaffen werden.

Da sich die Anwendung agiler Methoden während der Entwicklung des AIONAV LBS Editors durchgehend positive auf das gesamte Projekt ausgewirkt hatte, entschied man sich bei AIONAV bewusst für eine weiterführende Anwendung agiler Methoden. So wurden sie etwa bei der Weiterentwicklung der bestehenden mobilen App, die parallel zum AIONAV LBS Editor entstand, angewandt. Außerdem wurden Probleme erkennbar, wie etwa schlechte Wissensverteilung über gewisse Programmkomponenten, ausgelöst durch alleinige Zuständigkeit von einzelnen Personen für abgetrennte Bereiche oder Fehleranfälligkeit durch fehlende Tests. Agile Methoden der Softwareentwicklung sind bestens für die Lösung solcher Probleme geeignet (Collective Ownership, Code Review etc.) und werden deshalb in Zukunft intensiver genutzt, um von den, mit dieser Arbeit gezeigten, Vorteilen der umgesetzten Methodik profitieren zu können.

A. Mobile Anwendung

Parallel zum AIONAV LBS Editor wurde ein Prototyp einer mobilen App fur Enduser entwickelt (nicht im Rahmen dieser Arbeit). Die Entwicklungsplattform war Android. Als gemeinsame Softwarebasis mit dem AIONAV LBS Editor diente das Framework von AIONAV, welches als Java-Projekt vorlag. Funktionalitäten, wie z.B. der Datenbankzugriff sind in diesem Framework integriert. Zur Veranschaulichung der App folgen einige Screenshots mit Erklärungen.

Abbildung A.1.: Ansicht des Shoppylands Schönbühl mit Geschäften

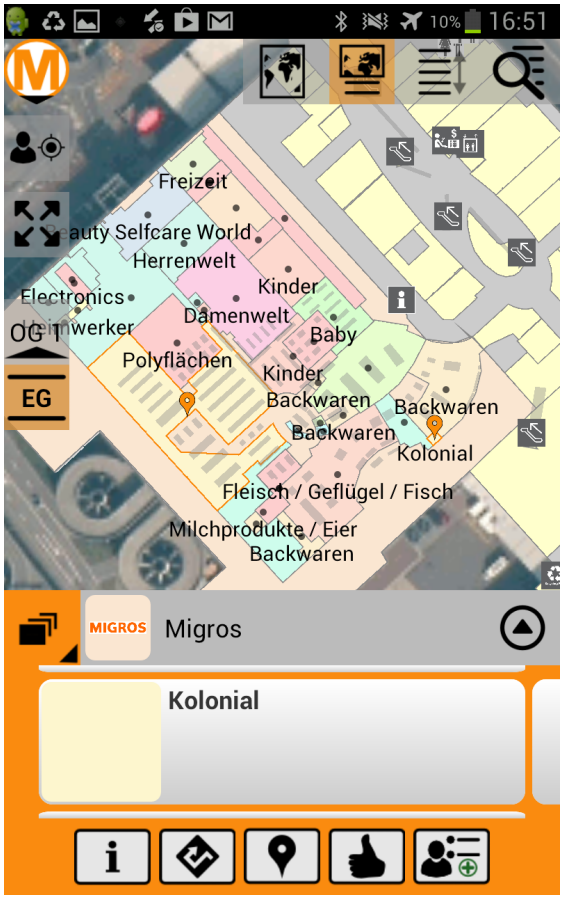

Abbildung A.2.: Detailansicht des Migros-Marktes. Anzeige von Bedarfswelten (verschiedene Abteilungen innerhalb eines Geschäfts)

Abbildung A.3.: Detailansicht des Migros-Marktes. Anzeige eines Produktes.

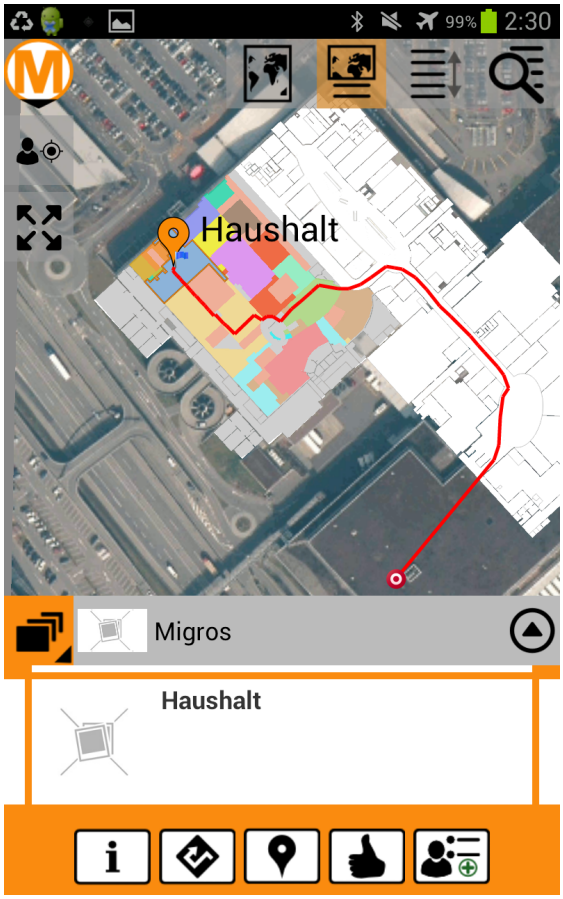

Abbildung A.4.: Ergebnis eines Routings. Der Bildschirm zeigt genau den Pfad, den ein Benutzer der mobilen Anwendung gehen soll, um zu einem POI zu gelangen (Anmerkung: die Live-Positionierung wird innerhalb eines Gebäudes ohne GPS, z.B. mit Hilfe eines Sensors am Fuß des Benutzers durchgeführt).

Literaturverzeichnis

[1] Pekka Abrahamsson, Outi Salo, Jussi Ronkainen, and Juhani Warsta. Agile software development methods. Technical report, VTT Publications, 2002.

[2] Ravikant Agarwal and David Umphress. Extreme programming for a single person team. In *Proceedings of the 46th Annual Southeast Regional Conference on XX*, ACM-SE 46, pages 82–87, New York, NY, USA, 2008. ACM.

[3] Scott W. Ambler. Lessons in agility from internet-based development. *IEEE Softw.*, 19(2):66–73, March 2002.

[4] Beck. *Test Driven Development: By Example.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[5] Kent Beck. *Extreme Programming – Das Manifest.* Addison-Wesley, München, 2004.

[6] Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change (2nd Edition).* Addison-Wesley Professional, 2004.

[7] Judy Bowen and Steve Reeves. Ui-driven test-first development of interactive systems. In *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems*, EICS '11, pages 165–174, New York, NY, USA, 2011. ACM.

[8] Alistair Cockburn. *Agile software development.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[9] Jangwoo Lee Danny Miller. The people make the process: commitment to employees, decision making, and performance. *Journal of Management*, 2001.

[10] Dominik Dary. Functional tests with the fest framework. *te-testing experience; The Magazine for Professional Testers*, (8):82–84, December 2009. can be found at http://www.de.capgemini.com/insights/publikationen/functional-tests-with-the-fest-framework/?d=83EA4B43-C8EF-9431-E33B-2BFC5E3BA2DA.

[11] E. Dijkstra. Classics in software engineering. chapter The humble programmer, pages 111–125. Yourdon Press, Upper Saddle River, NJ, USA, 1979.

[12] Abbot Entwickler-Website. http://abbot.sourceforge.net/. besucht am 18.02.2013.

[13] FEST Entwickler-Website. http://fest.easytesting.org/. besucht am 18.02.2013.

[14] Marathon Entwickler-Website. http://marathontesting.com/. besucht am 18.02.2013.

[15] Pounder Entwickler-Website. http://pounder.sourceforge.net/. besucht am 18.02.2013.

[16] UISpec4J Entwickler-Website. http://www.uispec4j.org/. besucht am 18.02.2013.

[17] John Favaro. Managing requirements for business value. *IEEE Software*, 19:15–17, 2002.

[18] John B. Goodenough and Susan L. Gerhart. Toward a theory of test data selection. *SIGPLAN Not.*, 10(6):493–510, April 1975.

[19] Jim Highsmith and Alistair Cockburn. Agile software development: The business of innovation. *Computer*, 34:120–122, 2001.

[20] Elke Hochmüller and Roland T. Mittermeir. Agile process myths. In *Proceedings of the 2008 international workshop on Scrutinizing agile practices or shoot-out at the agile corral*, APOS '08, pages 5–8, New York, NY, USA, 2008. ACM.

[21] Rashina Hoda, Philippe Kruchten, James Noble, and Stuart Marshall. Agility in context. *SIGPLAN Not.*, 45(10):74–88, October 2010.

[22] Agiles Manifesto: http://agilemanifesto.org/iso/de/.

[23] jfcUnit Entwickler-Website. http://jfcunit.sourceforge.net/. besucht am 18.02.2013.

[24] J. Langr. *Agile Java^{TM}: Crafting Code with Test-Driven Development.* Pearson Education, 2005.

[25] Johannes Link. *Unit Testing in Java - How tests drive the code.* dpunkt.verlag GmbH, Heidelber, Germany, 2003.

[26] Wes Masri and Marwa El-Ghali. Test case filtering and prioritization based on coverage of combinations of program elements. In *Proceedings of the Seventh International Workshop on Dynamic Analysis*, WODA '09, pages 29–34, New York, NY, USA, 2009. ACM.

[27] Atif Memon, Mary Soffa, and Mary Pollack and. Automated test oracles for guis. *SIGSOFT Softw. Eng. Notes*, 2000.

[28] Atif M. Memon. Gui testing: Pitfalls and process. *Computer*, 35(8):87–88, August 2002.

[29] Atif M. Memon and Mary Lou Soffa. Regression testing of guis. *SIGSOFT Softw. Eng. Notes*, 28(5):118–127, September 2003.

[30] Atif M. Memon, Mary Lou Soffa, and Martha E. Pollack. Coverage criteria for gui testing. In *Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering*, ESEC/FSE-9, pages 256–267, New York, NY, USA, 2001. ACM.

[31] Brad A. Myers. User interface software tools. *ACM Trans. Comput.-Hum. Interact.*, 2(1):64–103, 1995.

[32] Sridhar Nerur, RadhaKanta Mahapatra, and George Mangalaraj. Challenges of migrating to agile methodologies. *Commun. ACM*, 48(5):72–78, May 2005.

[33] Duc Hoai Nguyen, Paul Strooper, and Jörn Guy Süß. Automated functionality testing through guis. In *Proceedings of the Thirty-Third Australasian Conferenc on Computer Science - Volume 102*, ACSC '10, pages 153–162, Darlinghurst, Australia, Australia, 2010. Australian Computer Society, Inc.

[34] Michael Olan. Unit testing: test early, test often. *J. Comput. Sci. Coll.*, 19(2):319–328, December 2003.

[35] Ken Schwaber and Mike Beedle. *Agile Software Development with Scrum*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2001.

[36] Magnus Thorstein Sletholt, Jo Hannay, Dietmar Pfahl, Hans Christian Benestad, and Hans Petter Langtangen. A literature review of agile practices and their effects in scientific software development. In *Proceedings of the 4th International Workshop on Software Engineering for Computational Science and Engineering*, SECSE '11, pages 1–9, New York, NY, USA, 2011. ACM.

[37] Ranorex Tutorial von der Herstellerseite. http://www.ranorex.com/Documentation/Ranorex-Tutorial.pdf. besucht am 18.02.2013.

[38] Qing Xie and Atif M. Memon. Designing and comparing automated test oracles for gui-based software applications. *ACM Trans. Softw. Eng. Methodol.*, 16(1), February 2007.

[39] Hong Zhu, Patrick A. V. Hall, and John H. R. May. Software unit test coverage and adequacy. *ACM Comput. Surv.*, 29(4):366–427, December 1997.

[40] Broschüre über Ranorex von der Herstellerseite. http://www.ranorex.com/fileadmin/Brochures/Ranorex-Brochure.pdf. besucht am 18.02.2013.