Entwicklung eines Programms mit Extreme Programming

Lösung von Traveling Salesman Problems mit dem Ant Colony Optimization Algorithmus

Arne Krawielitzki, Alexander Landmann, Maurice von Loesch, Engin Yilmaz, Jan Zimmer

Keywords—Agile Softwareentwicklung; Ant Colony Optimization; Extreme Programming; Traveling Salesman Problem

I. EINFÜHRUNG

Die Softwareentwicklung hat über Jahre auf Entwicklungsprozessen wie dem Wasserfallmodell aufgebaut. Modelle wie dieses sind jedoch sehr statisch und können nicht flexibel auf neue Anforderungen reagieren. Diese Inflexibilität führt bei vielen Projekten dazu, dass diese nicht zum vereinbarten Termin fertiggestellt werden können oder sogar scheitern. Eine Alternative dazu bietet die Agile Softwareentwicklung. Extreme Programming, kurz XP, ist eines von mehreren Vorgehensmodellen der Agilen Software-entwicklung. Mit der Einführung von XP wurde eine Vielzahl an Praktiken definiert. Mit der Einhaltung der Praktiken und Werte von XP soll somit die Wahrscheinlichkeit erhöht werden, dass ein Projekt erfolgreich beendet werden kann. Im Zuge eines Projektes sollen diese Praktiken angewandt und vertieft werden. Dazu soll ein Programm entwickelt werden welches Traveling Salesman Problems mittels des Ant Colony Optimiziation Algorithmus löst. Dieses Paper beschreibt die Erfahrungen des Projektteams mit den XP Techniken während der Implementierung.

II. VORBEREITUNGEN ZUR ENTWICKLUNG

Zunächst hat sich das Projektteam hauptsächlich aufgrund der gemeinsamen Abneigung gegenüber der Programmiersprache Java zusammengefunden. Dadurch fanden sich die Teammitglieder Arne Krawielitzki, Alexander Landmann, Maurice von Loesch, Engin Yilmaz und Jan Zimmer zusammen. Gemeinsam wurde eine Programmiersprache festgelegt in der dhttp://code.google.com/p/uni-tsp-ant/as Programm kodiert werden soll, hierbei fiel die Wahl auf C#. Vorteile hier waren, dass einige Mitglieder des Teams bereits Erfahrungen in dieser Programmiersprache hatten und das .NET-Framework, ähnlich wie das Java-Framework, einen großen Funktionsumfang zur Verfügung stellt. Damit wurde die Suche nach einer Entwicklungsumgebung sehr beschleunigt, da es für C# nur wenige Möglichkeiten gibt. Die Wahl fiel hier auf Microsoft Visual Studio 2010.

Bei der Auswahl des Versionskontrollsystems standen Git und SVN im Raum. Wobei die Auswahl auf SVN fiel, da im Projektteam mehr Erfahrungswerte mit SVN vorhanden waren und bei der testweisen Verwendung von Git massive Probleme auftraten. Damit während der Entwicklung des Quellcode leicht mit dem Repository synchronisiert werden kann wurde das Programm TortoiseSVN [1] ausgewählt. Dieses bietet neben einer grafischen Benutzeroberfläche auch ein Plug-In für die verwendete IDE Microsoft Visual Studio an. Damit wird die Handhabung des Quellcodes in Verbindung mit dem Repository nochmalig vereinfacht. Als Hosting-Plattform für den Quellcode wurde Google Code [2] ausgewählt. Es bietet die Möglichkeit Projekte kostenlos zu speichern und unter anderem auch die Möglichkeit ein Repository per SVN zu verwalten. Zudem bietet Google Code die Möglichkeit sowohl ein Wiki zu dem Projekt als auch ein Ticketsystem zu erstellen. Wobei nur letzteres verwendet wurde, um den Entwicklungsprozess zu dokumentieren und zu steuern.

Im Zuge der Vorbereitung wurde zunächst getestet ob die Anforderungen mit dem .NET-Framework überhaupt umgesetzt werden können. Als schwierig zeigte sich schnell die Umsetzung der Anforderung der grafischen Ausgabe, da das .NET-Framework dafür keine Komponenten bereitstellt. Daher musste dafür nach Alternativen gesucht werden. Eine Lösung wurde in der Verwendung des TAO-Frameworks [3] gefunden, welches unter anderem eine Komponente zur Verfügung stellt mit der ein eingebettetes Fenster mit OpenGL-Kontext auf der Oberfläche platziert werden kann. OpenGL ist eine Grafik-API zur Ausgabe von Grafiken und Modellen in 2D und 3D. Mit den Erfahrungen von den Teammitgliedern Arne Krawielitzki und Engin Yilmaz in der Verwendung von OpenGL war dies eine naheliegende Entscheidung.

III. ROLLENVERTEILUNG

Die Rollen der Teammitglieder wurden diesen aufgrund ihrer Erfahrungen in der entsprechenden Rolle und den jeweiligen Wünschen zugewiesen. Die Rolle der Technischen Leitung ist erst während der Entwicklung entstanden, da ein entsprechender Bedarf entstanden ist und nicht von Beginn des Projektes bedacht wurde. Tabelle 1 zeigt die Aufgabenverteilung der Projektmitglieder bei der Entwicklung des Programms.

A. Rolle des Projektmanagements

	Arne Krawielitzki	Alexander Landmann	Maurice von Loesch	Engin Yilmaz	Jan Zimmer
Projekt-management	✓	✓			
Technische Leitung	✓				
GUI Design		✓			
Entwicklung	✓		✓	✓	✓
Tester		✓			

Tabelle 1 – Aufgabenverteilung im Projektteam

Die Rolle des Projektmanagements wurde von den Projektmitgliedern Arne Krawielitzki und Alexander Landmann umgesetzt. Das Erfassen der Anforderungen des Projektes und umwandeln in Storycards waren ein Aspekt des Projektmanagements. Dazu wurden die Anforderungen des Kunden so umformuliert, sodass daraus Funktionen des Programms aus Kundensicht beschrieben wurden, sowie Akzeptanzkriterien zum späteren prüfen ob eine Story vollständig implementiert wurde. Die erstellten Storycards wurden in das Ticketsystem eingefügt und dort verwaltet. Da Storycards nur eine Funktion beschreiben aber nicht vorgaben was alles getan werden musste bis diese vollständig implementiert sind, wurden Taskcards vom Projektmanagement formuliert und wie die Storycards in das Ticketsystem aufgenommen. Diese beschreiben explizit Aufgaben die von der Entwicklung implementiert werden müssen. Dabei musste auch das Technische Design des Programms beachtet werden, wodurch ein ständiger Kontakt zur Technischen Leitung entstand.

Mit den erstellten Taskcards war das Projektmanagement in der Lage den Entwicklungsprozess zu leiten. Dazu wurden die Taskcards gemäß ihrer Priorität bewertet und eine Reihenfolge festgelegt in der Funktionalitäten von der Entwicklung umgesetzt wurden.

Nach Abschluss aller Kodierungsarbeiten und Tests war es Aufgabe des Projektmanagements eine Dokumentation des Entwicklungsprozesses in Form eines Papers zu beschreiben.

B. Rolle der Technischen Leitung

Die Rolle Technische Leitung wurde vom Projektteilnehmer Arne Krawielitzki durchgeführt. Die Technische Leitung beschäftigte sich mit dem Design der Software und stellte dementsprechend Klassendiagramme zur Verfügung. Damit alle Entwickler wussten wie das Klassendiagramm zu verstehen ist und wie Datensätze entsprechend dem Design verarbeitet werden konnten, wurde ein gesondertes Meeting durch die Technische Leitung durchgeführt, um das Design vorzustellen und eventuelle Fragen zu klären.

Die Festlegung des Coding-Standards und das ständige durchführen von Codereviews um die Einhaltung dieses zu prüfen, waren die Kernaufgaben während der Zeit der Entwicklung. Des Weiteren stand die Technische Leitung den Entwicklern stets bei Fragen oder bei Problemen beratend zur Seite.

C. Rolle des GUI Design

Die Rolle des GUI Design wurde von Alexander Landmann durchgeführt. Der Aufgabenbereich umfasste die Planung des GUI Design, das Erstellen des Entwurfes der GUI, sowie das Erstellen der GUI für das Programm selbst. Dafür stand ihm ein Entwurfstool von Microsoft Visual Studio zur Verfügung. Die platzierten Komponenten mit Funktionalitäten zu hinterlegen lag nicht im Aufgabenbereich des GUI Designs.

D. Rolle der Entwicklung

Die Entwicklung wurde von allen Projektteilnehmern mit guten Programmier-kenntnissen übernommen. Dies waren Arne Krawielitzki, Maurice von Loesch, Engin Yilmaz und Jan Zimmer. Dabei war es die Aufgabe der Entwicklung die im Ticketsystem beschriebenen Taskcards programmiertechnisch umzusetzen und die vom GUI Design erstellte Oberfläche mit Funktionen zu hinterlegen.

Um die Fehlerwahrscheinlichkeit im Code zu reduzieren, war es auch Aufgabe den geschrieben Code in Automatisierten Tests zu prüfen. Daher war es Aufgabe der Entwickler Automatisierte Tests zu schreiben und so eine möglichst hohe Testabdeckung zu erreichen.

E. Rolle des Testers

Die Rolle des Testers wurde von Alexander Landmann übernommen. Zu der Kernaufgabe zählte das Durchführen von Tests, um zu prüfen ob die von Projektmanagement definierten Akzeptanzkriterien der Storycards erfüllt wurden und so zu garantieren, dass das Programm möglichst fehlerfrei dem Kunden übergeben werden konnte. Dazu mussten entsprechende Akzeptanztest formuliert und durchgeführt werden. Fehler wurden dokumentiert und in das Ticketsystem eingetragen, damit diese vom Projektmanagement bewertet werden konnten und bei Bedarf von der Entwicklung behoben wurden.

IV. Implementierung

A. Grafische Benutzeroberfläche

Damit dem Kunden schnell Ergebnisse gezeigt werden konnte, wurde eine Windows Forms Application erstellt. Diese bietet unter anderem die Möglichkeit, eine grafische Benutzeroberfläche mit einem Editor zu entwerfen. Damit konnten schnell Ergebnisse erreicht werden und die GUI hätte ggf. schnell geändert werden können. Einzig für die Ausgabe der Traveling Salesman Problems auf der Benutzeroberfläche musste ein gewisser Aufwand betrieben werden, da keine .NET-Standard-komponente die gewünschte Funktionalität verwirklichen konnte. Dazu musste auf ein externes Framework zurückgegriffen werden, das TAO-Framework. Dieses bot eine .NET-Komponente welche einen OpenGL-Kontext beinhaltete und somit der grafischen Ausgabe diente.

B. Ant Colony Optimization Algorithmus

Bei der Implementierung des Ant Colony Optimization (ACO) Algorithmus gab es verschiedene Möglichkeiten. Das Projektteam hat sich dabei an die Vorgaben aus der Projektvorstellung von der Dozentin Prof. Dr. Dagmar Monett-Diaz gehalten.

In der aktuellen Implementierung wird pro Iteration die Wegfindung für die vom Benutzer angegebene Anzahl an Ameisen durchgeführt. Nachdem alle Ameisen alle Punkte des TSP besucht haben, wird das Pheromon-Update (1) durchgeführt. Bei der Berechnung des Pheromon-Wertes, der auf den aktuellen Wert aufaddiert wird, hat sich das Projektteam für das „Ant quantitiy model“ entschieden (2). Dieses bezieht die Länge einer Strecke ein und macht damit kürzere Strecken für den Algorithmus attraktiver. So sollen bessere Ergebnisse erzielt werden.

$$\tau_{ij}(t+n) = \rho * \tau_{ij}(t) + \Delta\,\tau_{ij} \quad (1)$$

$$\Delta\,\tau_{ij} = \sum_{k=1}^{m} \frac{Q}{length_{ij}} \quad (2)$$

C. Optimierung

Da es bei der Implementierung einige Mängel an der Performance des ACO Algorithmus gab, mussten Wege gesucht werden diesen zu beschleunigen. In der ursprünglichen Form wurde der Algorithmus in einem Thread ausgeführt der nacheinander für jede Ameise einen Pfad bestimmt hat. Abschließend hat der Thread das Pheromon-Update durchgeführt und Statistiken aktualisiert. Da der Algorithmus die meiste Zeit bei der Suche des Pfades für die Ameisen verbrachte und dabei lediglich 40-50% der CPU-Auslastung verbrauchte, war dies der Flaschenhals. Um diesen zu lösen wurde die Wegsuche der einzelnen Ameisen in eigene Threads ausgelagert. Dadurch wurde der Prozessor dann vollständig ausgelastet und wie Performancemessungen zeigten, wurde ein Performancezugewinn von über 20% erreicht.

D. Auslieferung

Das Programm wird sowohl als 32-Bit Version als auch als 64-Bit Version auf der Downloadseite des Projektes bei Google Code [4] angeboten. Das Programm muss nicht installiert werden und kann direkt über die „*.exe“-Datei ausgeführt werden. Voraussetzung hierfür ist dass das .NET-Framework 1.0 und das .NET-Framework 4.0 installiert sind. Ist dies nicht der Fall wird der Benutzer mit einer Meldung darauf hingewiesen.

V. Angewandte XP-Praktiken

Bei der Umsetzung des Projektes wurde eine Vielzahl an XP-Praktiken verwendet. In den folgenden Abschnitten wird auf jede einzeln eingegangen.

A. User-Storys

Mit der Verwendung des Ticketsystems von Google Code stand dem Projektmanagement ein effizientes Mittel zur Verfügung, um User-Storys zu verwalten. Mit eigens angelegten Templates wurden die Anforderungen zunächst in Storycards umgewandelt.

Aus den Storycards konnten anschließend Taskcards formuliert werden, um die zu erledigenden Arbeiten zu beschreiben. Einer Task wurde eine Priorität und ein Status zugewiesen. Durch die Priorität wurde angegeben in welcher Reihenfolge die Tasks abgearbeitet werden sollen. Der Status gab an ob eine Aufgabe noch neu ist, bereits angefangen oder bereits implementiert wurde. Dadurch konnte das Projektmanagement stets schnell einen Überblick bekommen über was und wie viele Aufgaben noch zu implementieren sind. Damit wurde es dem Projektmanagement sehr erleichtert das Projekt zu steuern und den Entwicklern vorzugeben was priorisiert entwickelt werden sollte. Des Weiteren halfen gut durchdachte und möglichst umfassende Beschreibungen der Taskcards den Entwicklern bei der Implementierung. Allerdings war es ein sehr hoher Aufwand alle Aufgaben einzeln zu beschreiben. Zudem wurden zu Beginn nicht alle Möglichkeiten des Ticketsystems genutzt, etwa Verweise zwischen den Tickets oder zu Revisionen.

B. Planung

Bereits während der Formulierung der Tasks musste das spätere Design beachtet werden. Damit sollte eine gewisse Codebasis erstellt werden, die während der Implementierung an eventuelle neue Anforderungen angepasst werden konnte. Dazu wurde zunächst ein Technisches Design erstellt und in Form eines Klassendiagramms den Entwicklern zu Verfügung gestellt. Damit wurde sichergestellt dass alle Entwickler das gleiche Ziel während der Entwicklung haben und jeder weiß wie und wo Daten gespeichert sind und wie auf diese zugegriffen werden kann. Zusätzlich ermöglichte das frühe Design, dass nach der Kodierung der Kernklassen die Entwicklung der weiteren Komponenten unabhängig voneinander durchgeführt werden konnte, etwa dem auslesen der TSP-Informationen aus einer Datei und das Durchführen des Ant Colony Optimization Algorithmus.

Neben der Technischen Planung fand auch eine Planung auf Projektmanagement-Ebene statt. Dabei wurde festgelegt in welchen Iterationen welche Funktionen des Programms priorisiert entwickelt werden. So wurde festgehalten, dass in der ersten Iteration die Entwicklung der grafischen Benutzeroberfläche priorisiert und eine Vorbereitung des ACO-Algorithmus mit niedriger Priorität durchgeführt

werden sollte. In der zweiten Iteration wurde der Fokus auf die Entwicklung des ACO Algorithmus verschoben.

C. Stand-Up Meetings

Stand-Up Meetings wurden vor jedem Treffen durchgeführt bei dem alle Teammitglieder anwesend waren. Dabei haben sich die Projektmitglieder mitgeteilt was diese zuletzt gemacht haben, was diese als nächstes machen werden und ob sie Probleme haben bzw. hatten. Somit wusste jedes Teammitglied was die jeweils anderen als nächstes machen und konnten ggf. bei Problemen in einem gesonderten Meeting helfen. Die Meetings waren besonders wichtig für die Übersicht im Team, da nicht durchgehend zusammen gearbeitet worden ist und Programmierteams sich gelegentlich gesondert getroffen haben.

D. Coding-Standards

Bei der Verwendung von Coding-Standards wurde von der Technischen Leitung festgelegt, wie Klassen, Variablen, Konstanten, etc. zu benennen sind. Damit wurde sichergestellt, dass der Code ein einheitliches Aussehen hat und das Lesen von fremdem Code vereinfacht wird. Trotz des bewusst leicht gewählten Standards, haben sich nicht alle Projektteilnehmer konsequent an diesen gehalten. Dadurch entstand ein hoher Refrakturierungsaufwand. Dieser sollte dadurch verhindert werden, dass der Pair-Programming-Partner stets darauf achtete, dass dieser eingehalten wird. Da dies nicht ausreichte, wurden Codereviews von der Technischen Leitung durchgeführt und ggf. gleich refrakturiert.

E. Pair-Programming

Bei der Programmierung wurde besonders die XP-Praktik des Pair-Programming angewandt. Bei dieser Praktik wird der Code von zwei Entwicklern entwickelt.

Grundsätzlich wurden zwei Entwicklungsteams gebildet. Die Zuordnung zu den Teams geschah anhand der Vorkenntnisse und Erfahrung. Eines der Teams war primär mit der Entwicklung der grafischen Benutzeroberfläche und später der Fehlerbehebung beschäftigt. während das zweite Team für die Entwicklung des ACO Algorithmus verantwortlich war. Die Teammitglieder blieben die meiste Zeit festgelegt und wurden nur selten geändert. Lediglich wenn Teammitglieder nicht an einem Treffen teilnehmen konnten, wurden die Teams angepasst. Das Pair-Programming hat vor allem geholfen die Zahl der Fehler im Quellcode zu reduzieren und die Qualität des Codes massiv zu steigern. Des Weiteren führte das intensive arbeiten miteinander zu einem starken Wissensaustausch, von dem alle Teammitglieder profitierten.

F. Automatisierte Tests

Für die Verwendung der Automatisierten Tests wurde das Testframework von Microsoft Visual Studio eingesetzt. Dieses bietet die Möglichkeit Unit-Tests für ein Programm zu schreiben, um so die Funktionalitäten zu testen. Ziel war es eine möglichst hohe Codeabdeckung für alle funktionalen Klassen zu erreichen. Aufgrund anfänglicher Schwierigkeiten, die Testbiliothek bei allem Teammitgliedern lauffähig zu machen, verzögerte sich die Einführung der Tests. Da es zusätzlich sehr viel Erfahrung erfordert gute Tests zu schreiben und ein sehr hoher zusätzlicher Aufwand zur eigentlichen Programmierung ist, war die Akzeptanz gegenüber der Automatisierten Tests nicht groß. Daher wurde lediglich eine sehr geringe Testabdeckung erreicht und nur die Kernklassen durch Tests abgedeckt. Positiv zu bemerken ist jedoch, dass für die Klassen welche mit Tests versehen wurden, bedeutend weniger Fehler enthielten als die nicht getesteten Klassen. Zudem konnte während der Entwicklung sehr schnell überprüft werden ob trotz Änderungen am Quellcode Funktionen noch so funktionieren.

G. Continuous Integration

Für die Umsetzung von Continuous Integration (CI) wurde vereinbart stets Commits in das Repository zu machen nachdem eine abgeschlossene Funktionalität implementiert wurde, um so den Änderungsgehalt pro Commit gering zu halten. Damit konnten Fehler schnell gefunden und bei Bedarf genauso schnell wieder rückgängig gemacht werden. Neben den kontinuierlichen Änderungen sollte vor jedem Commit die Automatisierten Tests durchgeführt werden. Da dem Projektteam leider kein CI-Server zur Verfügung stand, der diese automatisch durchgeführt und dem Team Bescheid über das Ergebnis gegeben hätte, wurden automatisierte Tests lokal auf dem PC des Entwicklers und auf einem neutralen PC nach dem Commit durchgeführt und somit der gleiche Effekt mit ein wenig mehr Aufwand erreicht. Aufgrund der mangelnden Testabdeckung, beschrieben in F. *Automatisierte Tests*, konnten jedoch nicht so viele Fehler gefunden werden wie erhofft.

VI. ZUSAMMENFASSUNG

Die Verwendung der XP-Praktiken und deren Einhaltung war stellenweise nicht leicht. Teils aufgrund mangelnder Erfahrung, teils aufgrund fehlender Akzeptanz. Die XP-Praktiken bieten gute Ansätze, die, wenn man denn alle korrekt befolgt, helfen ein Projekt erfolgreich abzuschließen. Mit einem bereits aufeinander eingespielten Team und einem größeren Projekt kommen die Vorteile sicher besser hervor. Trotzdem hat das Projektteam bei der Umsetzung viele wertvolle Erfahrungen gesammelt. Vor allem dadurch dass viele Fehler begangen wurden die beim nächsten Projekt sicher nicht wieder auftreten würden oder man vorher nach Wegen suchen würde, wie man z.B. die Akzeptanz von Automatisierten Tests im Projektteam steigern könnte. Viele Techniken wie z.B. die Verwendung von User-Storys, trugen massiv zum Erfolg des Projektes bei und wurden durchweg positiv wahr genommen.

[1] http://tortoisesvn.net/ Abgerufen am 05.11.2012

[2] https://code.google.com/p/uni-tsp-ant/ Abgerufen am 05.11.2012

[3] http://www.mono-project.com/Tao Abgerufen am 05.11.2012

[4] https://code.google.com/p/uni-tsp-ant/downloads/list Abgrufen am 05.11.2012