Softwareentwicklung in der Wissenschaft

Ein Dialog zwischen Theorie und Praxis einiger Softwaretechnikelemente anhand eines Interviews

Seminararbeit

eingereicht bei

Dr. Hermann Lenhart

Arbeitsbereich Wissenschaftliches Rechnen

MIN Fakultät

Universität Hamburg

Betreuer:

Dr. Hermann Lenhart

Von:

Carlos Fernando Oliveira Santos

Inhaltsverzeichnis

1. Motivation und Einführung

Die Verbindung zwischen Praxis und Theorie ist beim heutigen Universitätssystem, in dem rapide vierzehn oder sechzehn Vorlesungen vorgetragen werden und in dem zusätzlich einige Übungen zur Vertiefung von Lesestoff dienen sollten, ist so gut wie nie vorhanden. Die Gelegenheit, sich theoretische Konzepte in der alltäglichen Routine eines Softwareentwicklers in einer Firma oder in einer Institution anzusehen, ist leider meistens während des Studiums nicht möglich. Viele der Studenten werden erst mit der Theorie konfrontiert, wenn sie einen Auftrag entgegen nehmen müssen und sich nicht mehr genau daran erinnern können, in wie weit die Theorie eine Unterstützung sein könnte.

Mit diesen Gedanken als Ausgangspunkt beschäftigte mich die Idee, in diesem Seminar genau diese Beziehung zwischen Theorie und Praxis näher betrachten zu können. Es musste dennoch eine Auswahl an Themen getroffen werden, damit eine präzise Beobachtung erfolgen konnte, ohne die Thematik des Seminars „Softwareentwicklung in der Wissenschaft“ aus den Augen zu verlieren. In dieser Hinsicht habe ich mich für die Softwaretechnik entschieden, denn man kann schnell theoretische Aspekte der Technik für die Softwareentwicklung auswählen und festlegen, um sie in der entwickelten Software zu verifizieren.

Der Fokus dieses Vorgehens war also zu identifizieren, welche der ausgewählten theoretischen Aspekte der Softwaretechnik sich in einer reellen Softwareentwicklung widerspiegeln würden, um zu analysieren, ob der feste rote Faden, der an der Universität vorgestellt wird, auch in der Entwicklung der Software für den akademischen Bereich, also für die Wissenschaft, verfolgt wird. Das Ziel war also feststellen zu können, ob es große Abweichungen zwischen der Theorie, die im Bereich der Softwaretechnik vermittelt wird, und der Praxis einer spezifischen Softwareentwicklung geben würde.

2. Interviewpartner

Der gefundene Interviewpartner ist der Dr. Felix Seifert. Er ist ein wissenschaftlicher Mitarbeiter der Universität Hamburg im Bereich der Entwicklungsbiologie. Er ist sowohl der Programmierer als auch der Endanwender seiner Software, die entwickelt worden ist, um hybridische Kreuzungen von Mais-Pflanzen durch das Vorhersagen zu optimieren.

Dr. Seifert hat im Hauptstudium Biologie mit Bioinformatik als Nebenfach studiert. Seine Programmierkenntnisse wurden im Nebenfach erweitert, obwohl er sich bereits aus seiner Jugendzeit damit beschäftigt hat.

3. Die Softwareentwicklung

3.1 Die Entwicklung und die Struktur der Software

Die Software wurde Ende 2012 konzipiert. Die gesamte Entwicklung hat Dr. Seifert allein übernommen. Die Softwareformulierung wurde anhand der Methode „Scratch“ durchgeführt. Das heißt, die gesamte Implementation des Programms wurde ohne weitere Hilfsmethoden getätigt. Der Aufwand für die gesamte Arbeit der Formulierung bis zur Einsetzung der Endversion der Software dauerte circa sechs Monate. Der Endbenutzer ist allein Dr. Seifert.

Die Programmiersprache, die benutzt wurde, ist Java. Als Zusatzsoftware wird noch die MySQL-Verbindung zu einer Datenbank genutzt, damit die Daten für die Softwareanalyse zur Verfügung stehen. Das ganze Programm wurde mit dem Entwicklungswerkzeug Netbeans programmiert. Sowohl Netbeans als auch die MySQL Datenbank sind Opensource Softwares.

Die gesamte Größe der Software beträgt circa 2 MB. Intern ist das Programm in 15 Klassen aufgeteilt, je mit einem Durchschnitt von zwei Hundert Zeilen. Eine Strukturierung durch Java Packages ist auch zum Teil vorhanden. Clustering durch ähnliche Klassen wurde benutzt, um die Packages zu formulieren. Kommentare im Code sind zum Teil vorhanden. Das Programm besitzt weder eine GUI (Graphical User Interface) noch eine Test-Klasse.

Zur Softwarebenutzung werden circa 10 Gigabytes an Informationen über die Pflanzenlinien in die Software eingespielt. Nach der Softwareanalyse wird eine CSV-Datei erzeugt, die dem Benutzer erlaubt, Muster zu erkennen. Weitere Softwares können dann mit der Datei arbeiten, um weitere Aspekte zu analysieren. Zur Zeit des Interviews besaß die Datenbank über 150 Gigabytes an Informationen über die Pflanzen, die bereits analysiert worden sind.

3.2 Anwendungsbereich der Software

Die Problematik seiner Arbeit besteht darin, die besten Pflanzen für eine mögliche Kreuzung auszuwählen, ohne jedoch tatsächlich alle kreuzen zu müssen. Durch eine Methode der RNA-Markierung ist es ihm möglich, diese Pflanzen zu identifizieren, und durch die Software wird eine Vorhersage durchgeführt, sodass die Probekreuzung entfällt. Im unteren Abschnitt ist die Beschreibung der Arbeit von Dr. Seifert, die er selbst formuliert hat:

„Die Pflanzenzucht der meisten wirtschaftlich relevanten Pflanzen erfolgt über Hybridzucht bei der zwei unterschiedliche reinerbige Eltern gekreuzt werden um den dabei auftretenden Heterosiseffekt, welcher eine den Eltern in vielen Merkmalen überlegene Pflanze bewirkt, sowie die Uniformität dieser ersten Hybridgeneration zu nutzen. Da jedes Jahr tausende neue Inzuchtlinien erzeugt werden ist die Anzahl der Kreuzungskombinationen nicht in Feldversuchen zu testen, die optimalen Kreuzungspartner werden daher über markerbasierte Vorhersagemethoden bestimmt.

In unserer Arbeit werden kurze RNA (sRNA), deren Profile in Keimlingen von Inzuchtlinien über Tiefensequenzierungen bestimmt werden mit Hybridmerkmalen assoziiert. Hierfür wurde ein Programm geschrieben, welches die Signifikanz der Assoziation für jede einzelne sRNA bestimmt und diese Wahrscheinlichkeit korrigiert. Dieses Programm erhält sowohl die sRNA-Expressionsprofile als auch die bekannten Hybridmerkmalswerte der möglichen Inzuchtlinienkombinationen und liefert als Ergebnis eine Auflistung der assoziierten sRNA sowie deren Wahrscheinlichkeit. Diese Information wird zum einen für weitergehende Analysen zur Charakterisierung dieser Sequenzen zur Identifikation des Beitrags von sRNA zur Entstehung von Heterosis, als auch für eine markerbasierte Vorhersage von Hybridmerkmalen herangezogen und das Verfahren patentiert. In dem gegenwärtigen Projekt wird dieses Programm nach Bestimmung der optimalen Parameter einmalig zur Assoziation und anschließend für eine Bewertung der Vorhersagegenauigkeit über eine Kreuzvalidierung genutzt.“

4. Ausgewählte Konzepte der Softwaretechnik

Zwischen den vielen Teilgebieten der Softwaretechnik, wie zum Beispiel Projektmanagement, Anforderungserhebung, Softwaretest und Wartung, wurden nur einige Punkte hervorgehoben, da eine solche Analyse zu umfangreich für diese Art von Arbeit wäre.

Die Auswahl der folgenden Unterpunkte Use Cases, Aktivitätsdiagramm, Mock-Ups, Testen und Code Review wurde nach der Vorlesung „Softwaretechnik“ an der Universität Hamburg im Bereich des Bachelors System-Software-Entwicklung 2015 getroffen. Die Auswahl ist relevant, denn die Vorlesungsinhalte könnten/konnten sofort mit der Praxis konfrontiert werden.

4.1 Use Cases

Ein Use Case ist eine informelle Beschreibung von einem Szenario oder mehreren Szenarien, die einen ähnlichen oder gleichen Ablauf aufweisen. Ein Szenario kann für diesen Kontext als eine Tätigkeit verstanden werden, die eine Routine darstellt. Ein Beispiel dafür ist eine Bestellung in einem Online-Shop: Ein Anwender A interagiert mit dem Bestellsystem und nach einigen ausgewählten Artikeln geht er zur Kasse. Für die Kassen-Interaktion ist es notwendig, dass der Anwender sich einloggt. Das Einloggen benötigt die Registrierung, falls der Anwender A noch nicht registriert ist. Abbildung A zeigt das Use Case zum beschriebenen Fall.

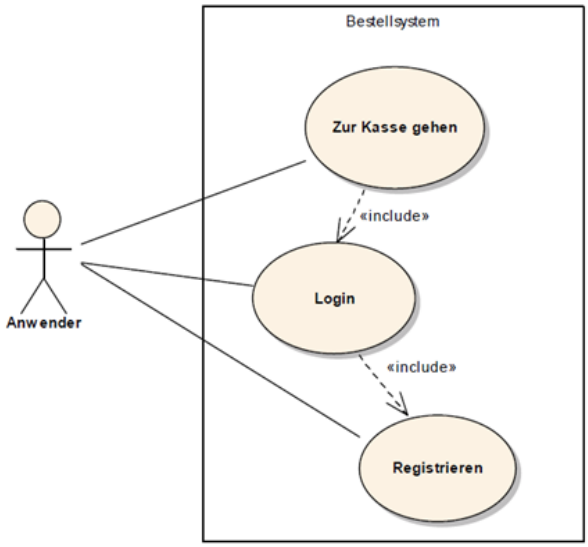

Abb. A: Ablauf einer Online-Bestellung

4.2 Aktivitätsdiagramm

Das Aktivitätsdiagramm beschreibt Abläufe. Dabei sind verschiedene graphische Elemente vorhanden, die dem Beschreibenden eine präzise Vorgehensweise eines Anwenders

mit dem zu entwickelnden System ermöglichen. In Abbildung B wird die Aktivität „Anschrift eingeben“ mit Hilfe des Aktivitätsdiagramms demonstriert. Der schwarze Punkt am Anfang stellt den Anfang der Aktivität dar; der schwarze eingekreiste Punkt am Ende symbolisiert das Ende. Zwischen den beiden Punkten

befinden sich mehrere Ovale mit entsprechender Beschreibung des aktuellen Schrittes der Aktivität. Bei dieser Darstellung ist der nächste Schritt erst dann dran, wenn der vorherige Schritt vervollständigt wurde. Ein Anwender A kann beispielsweise keinen Vornamen eingeben, wenn er noch keine Anrede gewählt hat usw. Erst wenn alle Etappen korrekt ausgefüllt wurden, ist die Aktivität zu Ende. Damit lässt sich dieses Vorgehen im Diagramm auf einer Softwareentwicklung übertragen, damit das System strukturiert aufgebaut wird.

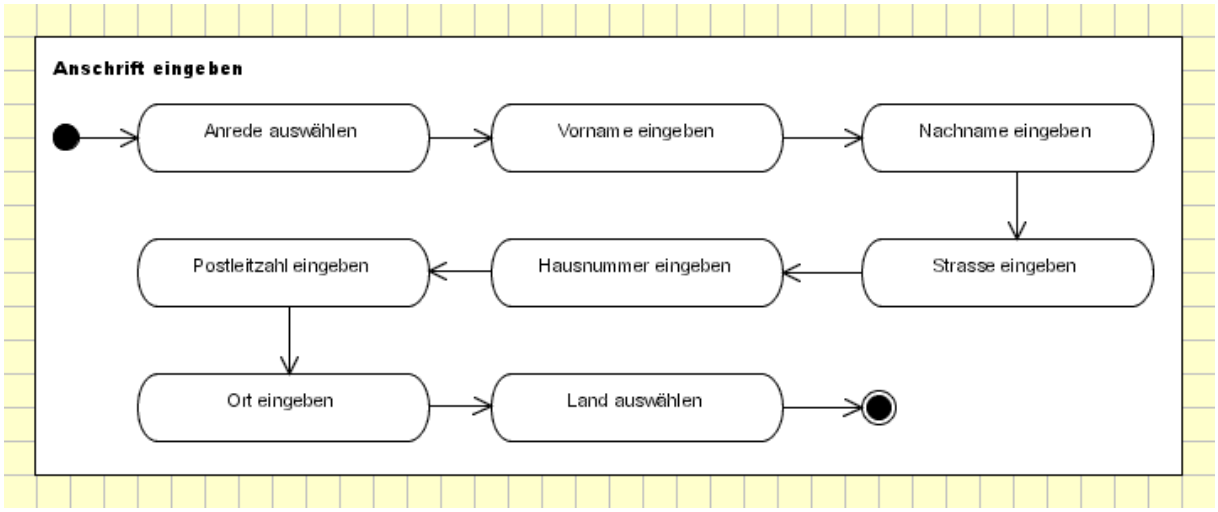

Abb. B: Aktivitätsdiagramm für die Eingabe einer Anschrift

4.3 Mock-Ups

Ein Mock-Up ist eine Skizze einer graphischen Oberfläche. Mit Hilfe dieser Skizze ist es möglich, eine Interaktion zwischen dem zukünftigen User und dem System zu veranschaulichen. Ein Mock-Up kann aus verschiedenen Materialien erstellt werden. Das Hauptziel ist, kostengünstig den User-Umgang mit dem System zu beobachten und Vorschläge für mögliche Veränderungen zu sammeln.

Abbildung C zeigt ein Beispiel für ein Mock-Up.

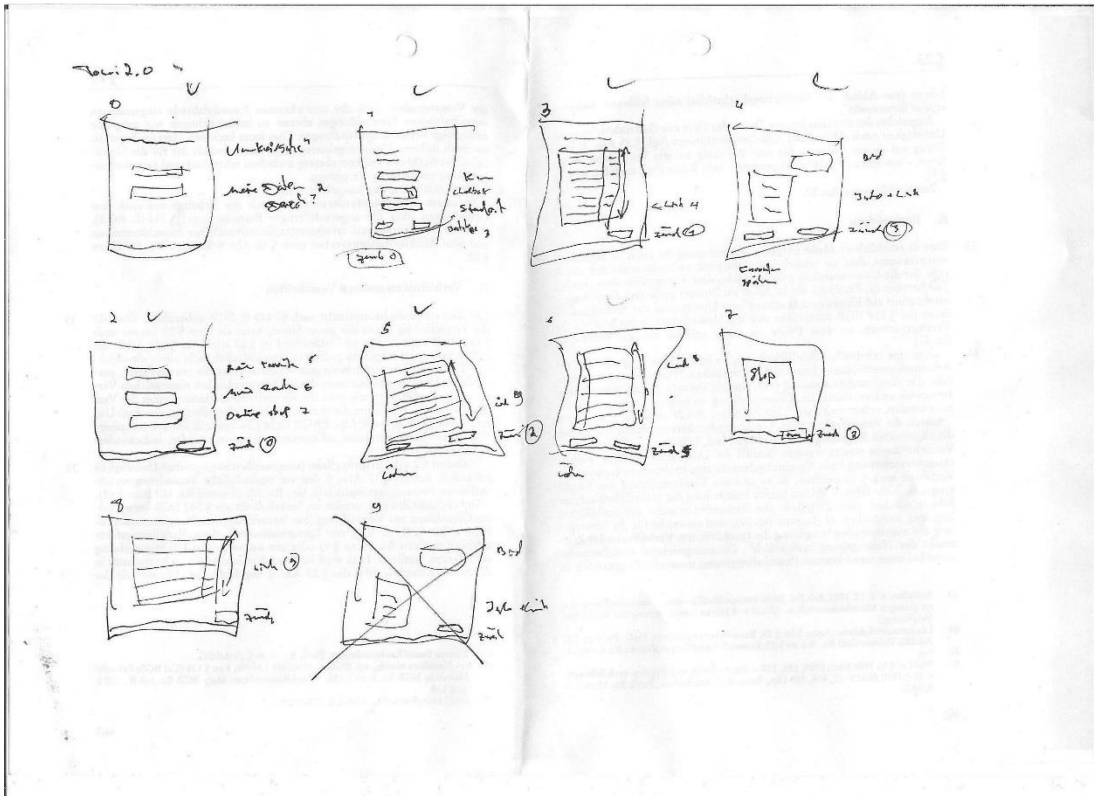

Abb. C: Skizzierung von Fensterdarstellungen eines Programms

4.4 Testen

Ein Softwaretest ist unter anderem eine Prozedur zur Prüfung und Bewertung einer Software in Beziehung ihrer Erfüllung von Anforderungen. Unter den unterschiedlichen Typen von Softwaretests wurden für diese Arbeit der Positive Test und der Negative Test gewählt. Der Gedanke dahinter war, dass diese beiden Typen von Tests das Minimum sein sollten, um eine Software ordnungsgemäß einsetzen zu können.

Der positive Test prüft, ob die Software die erwarteten Ergebnisse liefert, wenn die korrekten Parameter eingegeben werden. Der negative Test prüft im Gegenteil, ob kein Fehler in der Ausführung einer Software passiert, wenn falsche Parameter eingegeben werden. Die Abbildungen D und E erläutern dieses Vorgehen. Angenommen es gibt ein Feld, in dem ein User sein Alter eingeben soll. Das Alter eines Menschen kann nur als Zahl eingetragen werden, so die Aussage „Enter only numbers“. Ein positiver Test

wird testen, ob kein Fehler vorkommt und die Software weiter rechnet, wenn das Alter korrekterweise mit Zahlen eingegeben wird. Die Abbildung D zeigt ein Feld, in dem das Alter eingegeben werden soll.

Abb. D: Feld für die Eingabe des Alters, korrekterweise mit Zahlen befüllt

Ein negativer Test wird prüfen, ob die Software nicht abbricht, wenn das Programm nicht die erwarteten Parameter erhält. Im unteren Beispiel soll das Feld „Age“ nur Zahlen erhalten. Ein negativer Test würde dies überprüfen, indem z.B. Buchstaben eingegeben werden, um den möglichen Fehler abzusichern.

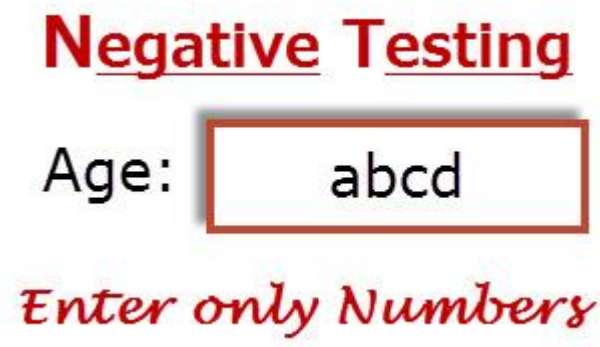

Abb. E: Feld für die Eingabe des Alters, fälschlicherweise mit Buchstaben befüllt

4.5 Code Review

Der Code Review ist eine Methode, in der der Quellcode und die gesamte Dokumentation eines Programms manuell nach Fehlern oder möglichen Lücken für Erweiterungen geprüft werden. Das Ziel eines Code Reviews ist hauptsächlich, Fehler zu vermeiden. Abbildung F zeigt einen Ausschnitt von Kriterien, die für das Code-Reviewing von Relevanz sind.

Qualitätsziele	Wartbarkeit					
Qualitätsmerkmale	Analysierbarkeit	Änderbarkeit	Stabilität	Testbarkeit	Konformität (W.)	
Checkliste						
SE2-Konventionen						
2.1 Code-Reihenfolge						
2.2 Einrückungen						
2.3 explizite Import-Anweisungen						
3.1 Schnittstellenkommentare („Was“)						
3.2 Implementationskommentare („Wie“)						
3.7 Methoden-Doku (“Javadoc-Tags“)						
4.2 Variableninitialisierung						
4.3 Klammerung von Blöcken(„{ ... }“)						
4.4 Klammerung von Bool-Ausdrücken						
5.1 Benennung von Klassen („Nomen“)						
5.2 Benennung von Methoden („Verben“)						
5.3 „Sprechende“ Variablen-/Methodennamen						
5.3 Konvention für Exemplarvariablen						
5.5 Namenskonvention für technische/fachliche Begriffe						
6.1 „private“ Exemplarvariablen						
6.1 Methoden mit einer „Return“-Anweisung						
Fowler: Bad Smells						
Comments: clarify "why" not "what"						
Klassengröße						
Methodengröße						

Abb. F: Ausschnitt von relevanten Punkten für den Code-Review

Die obere Abbildung zeigt einige Aspekte des Code Reviews, die als Konvention für die Softwareentwicklung an der Universität Hamburg dienen.

5. Gefundene theoretische Aspekte der Softwaretechnik

Nach dieser kurzen Einführung der relevanten Konzepte, die für dieses Interview ausgewählt worden sind, wird in diesem Abschnitt gezeigt, in welcher Form diese Aspekte mit der entwickelten Software in Beziehung stehen. Die zentrale Frage war für mich, ob sich Verbindungen zwischen den theoretischen Konzepten und der Praxis finden würden und wenn ja, inwiefern man sie nach praktikablen Maßnahmen für eine bessere Softwareentwicklung beurteilen könnte. Denn es ist bekannt, dass durch die Modulare Aufteilung der Themen, die ein Bachelor Studiengang an der Universität anbieten soll, nicht so viel Zeit für eine reale Anwendung der Konzepte, die vermittelt werden, und für die Diskussion nach einer optimalen Benutzung dieser Theorie für das reale Leben als Entwickler von Softwares zur Verfügung steht.

Die erste Gegenüberstellung war die Befragung des Interviewpartners nach den Konzepten, die für das Interview ausgewählt worden sind. Dem Interviewpartner waren einige theoretische Aspekte nicht bekannt (Use Cases) oder er hatte keinerlei Bedürfnisse einige der ausgewählten Softwaretechnikmethoden anzuwenden (Aktivitätsdiagramm und Mock-

Ups). Hier kann man leicht eine Begründung dafür finden: Der Programmierer ist auch der End-User. In diesem Sinne sind Use Cases, Aktivitätsdiagramme oder Mock-Ups überflüssig, denn der Programmierer hatte bereits die Relationen zwischen den möglichen Klassen und Funktionen des Programms im Kopf dargestellt und benötigte keinen User, der sich mit seiner Software auseinandersetzte, um festzustellen, ob Verbesserungen vorkommen sollten. Auch die Abwesenheit einer GUI (Graphical User Interface – Graphische Benutzeroberfläche) schließt die Notwendigkeit des Mock-Ups aus und macht eine Beschreibung von Aktionen zwischen dem möglichen Benutzer und der Software durch ein Aktivitätsdiagramm überflüssig.

Ein zentraler Punkt, der sehr interessant für mich war, war das Testen. Denn durch das Testen kann festgestellt werden, ob man der Software vertrauen kann. Für den wissenschaftlichen Bereich ist aus dieser Perspektive das Testen der zentrale Schlüssel, denn ohne eine Sicherheit, dass man den Ergebnissen vertrauen kann, gibt es keine Verwendung für die Resultate, die durch die Software generiert werden.

In Java werden normalerweise Tests mit einem Extra-Modul geschrieben, das dem Programmierer erlaubt, für jede Klasse des Programms eine Test-Klasse zu schreiben, um sich vergewissern zu können, dass die Klasse das leistest, was sie leisten soll. Außerdem gibt es in Java die Möglichkeit Assert-Anweisungen innerhalb von Methoden zu formulieren, die eine Art Kontrolle über Ausgaben einer Methode darstellen kann, solange die Methode die Eingaben erhält, die sie für ihre Berechnungen benötigt. Dieses Konzept wird als Vertragsmodell bezeichnet.

Das Testen wurde bei dieser Entwicklung in gewisser Weise in der Form vom Debugging angewendet. Wenn das Programm nicht die erwarteten Antworten geliefert hat, wurde das Debugging angewendet, um festzustellen, an welcher Stelle das Problem liegen könnte. An dieser Stelle war die Frage, inwiefern man der Software vertrauen kann, wenn keine Test-Klassen oder Tests im Allgemeinen existierten. Wenn die Aspekte wie Mock-Ups oder Aktivitätsdiagramm für diese Software nicht in Frage kamen, weil der Entwickler auch der Endanwender ist und auch keine GUI vorhanden ist, ist trotzdem das Testen ein zentraler Aspekt für die Benutzung der Ergebnisse für die weitere Forschung.

Dr. Seifert erklärte, dass bei Abwesenheit von Fehlern während der Durchführung der Softwareberechnung die Ausgabedaten mit Erwartungsdaten (Probe) verglichen wurden,

sodass erstens die Daten vertrauenswürdig sind und zweitens man schnell mit einem Teil/Portion der Analyse weiterarbeiten konnte. In diesem Sinne arbeitet die Software nicht allein und ist befreit von Test-Klassen. Mais-Pflanzen wurden bereits gekreuzt, sodass er die Werte seiner Software mit einer realen Probe vergleichen konnte, um festzustellen, ob das Programm tatsächlich das lieferte, was es liefern sollte.

Auch der Code Review wurde nur teilweise angewendet. Ein Review wurde während der Entwicklungsphase und nur bei Feststellung von Fehlern durchgeführt. Da der Quelltext nur von einem Programmierer behandelt wird, ist für Dr. Seifert das Code Reviewing nicht notwendig. Er kennt seinen Code vom Scratch, d.h., er hat den gesamten Code von Null geschrieben und kennt dessen Struktur und Relationen zwischen den Methoden und Klassen, sodass Verbesserungen im Sinne der Lesbarkeit oder Wartbarkeit nicht in Frage kommen.

6. Andere relevante Aspekte

Auch relevant für die Entwicklung der Software war die Frage, wie wichtig die Software für die Forschung von Dr. Seifert sei. Er meinte, eine Kreuzung mit den verschiedenen Pflanzen sei möglich, um festzustellen, welche von diesen die beste Auswahl für die Entwicklung von den Mais-Feldern sei. Allerdings würde eine manuelle Kreuzung eine riesige Anzahl an potentiellen Pflanz-Kandidaten erzeugen und somit auch die finanziellen und zeitlichen Kosten enorm sprengen. Ohne seine Software würde eine vollständige manuelle Kreuzung circa 10-15 Jahre dauern. Mit der Software bekommt er die notwendigen Ergebnisse für die weitere Forschung innerhalb von Stunden.

Auf die Frage ob die Software erweitert, gepflegt oder verbessert wird, hat Dr. Seifert mir mitgeteilt, dass keiner dieser Aspekte relevant für seine Forschung sei. Auch eine Analyse von Änderungsnotwendigkeiten an der Software findet nicht statt.

Die gesamte Entwicklung der Software erfolgt auf Freeware-Softwares. Das heißt, die Abteilung von Dr. Seifert trägt keine weiteren Software-Kosten für die Entwicklung. Kosten und Zeit spielen hier eine zentrale Rolle. Die Software musste so konzipiert werden, dass die Forscher mit ihrer Arbeit so schnell wie möglich voran kommen konnten. Auch wenn dies bedeuten würde, dass das Programmieren der Software nicht nach wissenschaftlichem Vorgehen geschehen könnte.

7. Zusammenfassung

Als ich am Anfang mich für das Interview entschieden habe, habe ich dies anhand meiner Notwendigkeit für eine nähere Beziehung zwischen Praxis und Theorie in der Softwareentwicklung getan. Ich habe mir gehofft, dass die Entwicklung einer Software in der wissenschaftlichen Umgebung eine enge Relation zwischen den theoretischen Aspekten und der praktischen Entwicklung der Software darstellen würde. Meine Erwartungen waren tatsächlich, einen Pfad für die Entwicklung zu sehen, eine Art festen Ablauf mit Schritten und Checkpoints, denn in diesem Bereich wie in einer schriftlichen Ausarbeitung zum Beispiel existieren sollte. D.h., die Entwicklungsschritte müssten dokumentiert werden, damit, falls notwendig, die gesamte Forschung reproduzier-, und nachvollziehbar bleibt. Allerdings war dies bei diesem Interview und der praktischen Analyse der Softwareentwicklung nicht der Fall.

Interessant wäre es noch zu wissen, ob die Softwareentwicklung in dem wissenschaftlichen Kontext im Allgemeinen in ähnlicher Form wie in diesem Beispiel mit den theoretischen Konzepten umgeht. Es wäre relevant, dieselben theoretischen Aspekte dieser Ausarbeitung zu nehmen und erneut anhand eines anderen Interviewpartners aus einem anderen Bereich - vielleicht eine Abteilung der Regierung, die eine Software für eine große Datenanalyse benötigt - zu führen, um einen Vergleich herstellen zu können. Auch der Vergleich bei einer Entwicklung, die nicht nur durch einen Softwareentwickler durchgeführt wird, wäre ebenso relevant für den Einsatz der hier ausgewählten wissenschaftlichen Aspekte.

Als Fazit des gesamten Interviews stellt man fest, dass die Software nur als Mittel für die Forschung gesehen wird. Meine Erwartungen in Bezug auf einer Annäherung von Praxis und Theorie im Bereich der Softwareentwicklung konnten nicht erfüllt werden. Die Entwicklung wird nicht streng kontrolliert und die entstandene Software ist nur ein Werkzeug für die weitere Forschung. Eine besondere schnelle Leistung ist kein Ziel für die Software. Die Kosten werden priorisiert. Die Softwaretechnik-Theorie war teilweise unbekannt oder bewusst nicht eingesetzt, denn sie war für das Ziel der Software nicht relevant. Insgesamt ist Dr. Seifert eine gute Software gelungen, obwohl keine intensive Auseinandersetzung mit der Theorie stattgefunden hat.

8. Literaturverzeichnis

Unterstanding the High-Performance-Computing Community: A Software Engineer‘s Perspective. Cruzes, Carver, Hochstein, Hollingsworth, Shull. IEEE 2008.

Vorlesung: Softwaretechnik. Gryczan, Riebisch. Sommersemester 2015.

Softwaretechnik in Wikipedia https://de.wikipedia.org/wiki/SoftwaretechnikLetztes Mal aufgerufen am 06.07.2015

9. Bildquellen

Abb. A: https://www.codecentric.de/files/2011/07/web-shop-beispiel-user-cases.png

Abb. B: http://www.highscore.de/uml/img/aktivitaet_anschrifteingeben.gif

Abb. C: http://www.highscore.de/uml/img/aktivitaet_anschrifteingeben.gif

Abb.D:http://www.softwaretestingclass.com/wp-content/uploads/2013/10/negative-testing.jpg

Abb.E:http://www.softwaretestingclass.com/wp-content/uploads/2013/10/postitive-testing.jpg

Abb. F: Checklist-Tabelle: Material nur innerhalb der Übungen für die Vorlesung der Softwaretechnik der Universität Hamburg verfügbar. Auf Nachfrage bitte unter: rhenesys@hotmail.com