Kapitel 1

Einleitung

Die Ausgangssituation Eine normale Softwarewartungs-Situation liegt vor, wenn man ein Softwaresystem hat, das bereits im Einsatz ist, das in weiten Teilen gut funktioniert, das aber auch Fehler enthält und das um neue oder verbesserte Funktionalitäten erweitert werden soll. Diese wird zu einer schlechten Wartungs-Situation, wenn das Softwaresystem nicht oder unzureichend dokumentiert ist, wenn dessen Strukturen und Algorithmen niemand mehr richtig versteht und die notwendigen Änderungen nicht vorgenommen werden können, ohne das Funktionieren des Vorhandenen zu gefährden.

Zu einer *Reverse Engineering* (RE) Situation wird diese Situation dann, wenn eine Entscheidung gefällt wird, die zum Ziel hat, das vorhandene Softwaresystem wieder vollständig zu verstehen und dieses Verständnis ausreichend in einer Dokumentation festzuhalten um anschließend das Softwaresystem wieder warten und weiterentwickeln zu können, wenn also die Entscheidung zum Ziel hat, möglichst viel des Vorhandenen zu retten[1].

Eine alternative Entscheidung wäre, von vorne anzufangen und ein neues Softwaresystem zu entwickeln. Wann welche dieser Entscheidungen die bessere ist, ist kaum allgemein zu beantworten und kann hier nicht weiter behandelt werden. Für ein *Reverse Engineering* könnte unter anderem sprechen, dass in dem vorhandenen System bereits viel Wissen über den Anwendungsbereich gespeichert ist, das zu bewahren sich lohnt – schließlich funktioniert das System bereits. Auch ermöglicht das RE eine schrittweise Annährung an ein besseres System und führt daher wahrscheinlich früher zu Zwischenergebnissen, die den Benutzern des Systems zu gute kommen, wogegen bei einer Neuentwicklung eine größere Zeitspanne vergehen dürfte bis der Funktionsumfang des aktuellen System erreicht ist.

Das Projekt „Softwaresanierung“, in dessen Rahmen diese Arbeit entstanden ist, war genau in einer solchen Situation. Das XCTL-Programm zur Steuerung von Versuchen zur Röntgentopographie, -diffratometrie und -reflektometrie wird am Institut für Physik der Humboldt-Universität intensiv benutzt. Es läuft im wesentlichen stabil, weist aber auch eine Reihe von Schwächen und Fehlern auf und soll um neue Funktionen erweitert werden. Es gab so gut wie keine Dokumentation und der mit ca. 27.000 Zeilen recht umfangreiche Quelltext war nur spärlich kommentiert[2].

[1] zu den Begriffen *Reverse Engineering* und *Forward Engineering* s. [Bal98, S. 663ff.].

[2] Zu Entstehung und bisherigem Verlauf des seit dem Wintersemester 98/99 laufenden Pro-

Inzwischen sind die einzelnen Funktionalitäten des Programmes relativ gut erkannt, verstanden und dokumentiert und es sind Subsysteme identifiziert worden, die für die Verteilung einzelner Aufgaben an die Projektteilnehmer die Grundlage darstellen[3]. Auch sind bereits einige der gewünschten Erweiterungen realisiert worden[4].

Eines der Subsysteme des XCTL-Programmes ist die *Motorsteuerung*. Es ist ca. 5.500 Zeilen groß, kapselt die hardwarenahe Ansteuerung der Motoren, stellt ein Programmier-Interface zur Nutzung der Motoren zur Verfügung und implementiert einige Dialoge für grundlegende Steuerungsaufgaben. Das Subsystem ist gut von den übrigen Subsystemen zu trennen; es ist in einer eigenen Bibliothek implementiert, die ausschließlich das Programmier-Interface exportiert.

Die Aufgabenstellung Die Aufgabe, die in der vorliegenden Arbeit behandelt wird, ist der *Test der Motorsteuerung*. Ziel ist, durch den systematischen Test Fehler aufzuspüren und durch die Realisierung eines automatisierten Regressionstests einzelne RE-Schritte wie z.B. die Portierungen oder die Restrukturierungen zu unterstützen. Der Regressionstest dient dabei der Sicherstellung, dass die letzten Änderungen am Softwaresystem nur die beabsichtigten Wirkungen haben und ohne unerwünschte Nebenwirkungen durchgeführt wurden.

Besonderes Augenmerk beim Test der Motorsteuerung liegt auf der Sicherheit der zu steuernden Hardware. Das betrifft zum einen den Test selbst und zum anderen das Verhalten der Software in der realen Umgebung. D.h., dass a) der Test der Software unabhängig von der realen Hardware ermöglicht werden sollte und b) dass die von der Software realisierte Steuerung der Hardware besonders beobachtet und getestet werden muss.

Ergebnisse der Arbeit Um die gestellte Aufgabe zu lösen, ist im Rahmen dieser Arbeit zuerst eine *Umgebungssimulation* geschrieben worden. Sie ermöglicht das Betreiben der hardwarenahen Softwarebestandteile, ohne dass echte Motoren vorhanden sein müssen. Neben dem Ein- und Ausgabeverhalten der Motorcontroller simuliert die Motorensimulation auch das Ort-Zeit-Verhalten der Motoren. Dadurch kann die Simulationskomponente auch für den Test abstrakterer Funktionen des XCTL-Systems dienen, falls diese die Motorsteuerung verwenden. Desweiteren realisiert die Simulationskomponente die Protokollierung der Kommunikation zwischen Software und Hardware und ermöglicht den Test spezieller Situationen und Konfigurationen.

Als nächstes ist dann zum einen ein relativ allgemeines *Testsystem* entworfen und implementiert worden und zum anderen sind *konkrete Testfälle* für den Test der Motorsteuerung entwickelt worden.

Das Testsystem integriert die erforderlichen Testaktivitäten — vom Testentwurf, über die Testprogrammgenerierung und -ausführung, bis hin zur Testauswertung und -dokumentation — in einem durchgehenden System von weitgehend automatisierbaren Arbeitsschritten. Zu den automatisierten Schritten gehört z.B. die Überführung der Analyse der Eingabedaten, die mit der

jektes s. [Bot00], [Bot01] und [BS01]. Einen guten Überblick über den Versuch und den Versuchsaufbau der Röntgentopographie bietet [FH01, S. 2-12]. Ähnliches für die Diffrato- und Reflektometrie ist bei [BU01, 2-6] zu finden.

[3] Zur Wiedergewinnung der Subsysteme s. [Sch01].

[4] s. [FH01], [Bus01] und [BU01].

Klassifikationsbaummethode durchgeführt wurde, in ein ausführbares Testprogramm und der Vergleich der Ist- mit den Soll-Ausgabedaten. Das Testsystem ermöglicht die Durchführung einzelner Aktivitäten, einzelner Testfälle oder ganzer Gruppen von Testfällen bzw. Regressionstests.

Für den Test der Motorsteuerung mit dem zuvor entwickelten Testsystem sind ca. 100 Testfälle realisiert worden. Die drei Testpakete, in die die Testfälle gegliedert worden sind, enthalten die Analyse der relevanten Eingabedaten, die Festlegung der konkreten Testdaten, die Auswahl der Testfälle sowie die Soll-Ausgabedaten. Durch den Test sind einige Fehler bzw. Fehlergruppen in der Motorsteuerung gefunden worden.

Übersicht über die Arbeit Die vorliegende Arbeit besteht nun aus folgenden Teilen.

In Kapitel 2 wird das Verhältnis von Softwaretest und *Reverse Engineering* untersucht und das Vorgehensmodell, das dieser Arbeit zugrunde liegt, erläutert.

Kapitel 3 beschreibt die Architektur und die Realisierung der Motorensimulation und ihrer Einbindung in die Motorsteuerung.

Kapitel 4 stellt die Struktur des Testsystems dar, erläutert einzeln die Funktionen und die integrierten Testaktivitäten und führt an einem ausführlicheren Beispiel deren Zusammenspiel vor.

Kapitel 5 enthält die Beschreibungen der für den Test der Motorsteuerung entworfenen Testpakete und eine Liste der gefundenen Fehler.

Im abschließenden Kapitel 6 wird zum einen eine Reihe von Arbeitsschritten aufgeführt, die notwendig sind um den Test der Motorsteuerung zu vervollständigen und zum anderen solche, die zur Verallgemeinerung und Verbesserung des Testsystems durchgeführt werden könnten.

Anhang A enthält die aktuelle Version der Verhaltensspezifikation der Motorsteuerung.

Anhang B stellt einige Informationen über die Motorhardware zusammen, die im Laufe dieser Arbeit besonders wichtig waren.

Und Anhang C enthält die Dokumentation der Perl-Pakete, die zur Implementation des Testsystems gehören.