

**Praktikum**(03.10. - 31.12.2016)

bei Eppendorf Instrumente GmbH

Stephan Pöhland

Matrikelnr.: 213204491

Informationstechnik/Technische Informatik

Universität Rostock

Inhalt

[Einleitung 3](#_Toc476161549)

[Firmenvorstellung 4](#_Toc476161550)

[Praktikumsablauf 5](#_Toc476161551)

[Detaillierte Aufgabenbeschreibungen 6](#_Toc476161552)

[Erlernen von git-Grundlagen 6](#_Toc476161553)

[Kennenlernen von EpBlue 7](#_Toc476161554)

[HexToBin-Converter 7](#_Toc476161555)

[Commitlogger / Changelog-Generator 8](#_Toc476161556)

[Release-Skript 10](#_Toc476161557)

[3D Modelle 11](#_Toc476161558)

[Modell-Tester 12](#_Toc476161559)

[Mischrack-Editor 13](#_Toc476161560)

[Zusammenfassung 15](#_Toc476161561)

# Einleitung

An der Universität Rostock gibt es für den Studiengang Informationstechnik/Technische Informatik die Möglichkeit im 6. Semester ein Pflichtpraktikum zu absolvieren. In ungefähr 12 Wochen sollen praktische Erfahrungen in einem Betrieb der Wahl gesammelt werden. Abschließend wird ein Praktikumsbericht erwartet, der reflektierend die Erlebnisse und Erwartungen des Studenten darstellen soll.

Die Entscheidung von mir, das Praktikum anstatt der Vorlesungen im 6. Semester zu wählen, fiel früh. Schon als ich mit dem Studium begonnen habe, war mir klar, dass ich einen praktischen Bezug brauche, um über die Spezialisierungen, die ich in Zukunft und vor allem beim Master wählen kann, entscheiden zu können. Mit der Wahl des Praktikumsbetriebs habe ich es mir allerdings nicht so einfach gemacht. Ich hatte einige Kriterien, die das Unternehmen erfüllen sollte und auf die ich bei der Suche besonders geachtet habe.

Eines der Kriterien war, dass es ein Betrieb sein sollte, der Software für spezialisierte Hardware produziert. Dies fand ich besonders deshalb wichtig, da ich der Meinung war, dass ich durch meinen Studiengang Technische Informationstechnik eine Laufbahn gewählt habe, bei der "normales" Softwareprogrammieren kein erstrebenswertes Ziel ist, sondern ein hardwarebezogenes Schreiben von Code, bei dem bestimmte Hardwarekomponenten angesteuert werden sollten. Mein Wunsch war es etwas mehr über die Mikrocontrollertechnik zu lernen und eventuell Firmware für kleinere Boards zu programmieren.

Ein weiteres Kriterium war der Standort und die Größe des Unternehmens, was in gewissen Maßen unmittelbar zusammenhängt. Ich wollte ein recht großes Unternehmen kennenlernen um zu sehen, wie die Strukturen und Abläufe in einem Betrieb mit der Größe von ungefähr über 1000 Mitarbeitern aussehen, um später bei einem anderen Praktikum mit einem kleineren Unternehmen vergleichen zu können. Da die Auswahl an größeren produzierenden Firmen mit Softwareabteilung in Rostock und Umgebung recht klein ist und ich nicht in die Windkraftbranche möchte, blieb für mich noch der Blick auf die alte Heimat in Schleswig-Holstein, wo die Situation allerdings ähnlich ist. Die Metropolregion Hamburg war für mich die beste Wahl, da hier sehr viele größere Betriebe angesiedelt sind, die Softwarebedarf haben und es noch in der näheren Umgebung gelegen ist.

Ich habe mich mit diesen Kriterien auf die Suche im Internet nach Firmen bzw. Praktikumsplätzen gemacht und mich bei einigen beworben. Ungefähr zwei Monate vor geplantem Beginn des Praktikums, hatte ich einige Bewerbungsgespräche, nach denen ich mich für die Firma Eppendorf AG entschieden habe. Sie erfüllte alle meine wichtigen Kriterien und schien, als ob hier öfter Studenten arbeiten, was eine gute Voraussetzung war. Da mir und auch dem Betrieb die drei Monate Pflichtpraktikum zu wenig waren, habe ich beschlossen noch drei Monate freiwillig an den Pflichtteil anzuhängen. So war gewährleistet, dass ich genug Zeit habe mich einzuarbeiten und am Ende auch noch für die Firma nutzbringend arbeiten konnte.

# Firmenvorstellung

Die Eppendorf AG ist ein weltweit agierendes Unternehmen mit ungefähr 3000 Mitarbeitern. Bei einem Umsatz von rund 630 Mio. EUR in 2015 werden Tochtergesellschaften in 25 Ländern unterhalten, die die Firma repräsentieren. Der Betrieb wurde 1945 in Hamburg gegründet und ist seitdem stetig gewachsen. Der Name stammt von dem Stadtteil Hamburgs, in dem die Gründung stattfand. Heute befindet sich der Hauptsitz allerdings in einem anderen Stadtteil Hamburgs.

Eppendorf entwickelt Laborgeräte für Biologie- oder Chemielabore. Zu der Produktpalette gehören unter anderem Pipetten, Zentrifugen, Mischer oder Pipettierautomaten. Die Geräte dienen im Allgemeinen zur Aufbewahrung, Aufbereitung oder Verarbeitung von biologischen oder chemischen Substanzen und sind nur teilweise technisch. Neben den normalen Pipettiersystemen, die per Hand bedient werden, bietet das Unternehmen auch automatische Pipettiereinheiten, die spezielle Software zur Bedienung benötigen. Eppendorf produziert nicht nur die Systeme sondern auch die dazugehörigen Verbrauchsartikel wie zum Beispiel Pipettenspitzen, Gefäße und Mikroliterplatten. Als Abnehmer dieser Produkte werden unter anderem industrielle oder akademische Labore, sowie Forensik oder Lebensmittelindustrie genannt.

Die Abteilung in der ich das Praktikum absolviert habe, ist für die Software der technischen Geräte der Firma zuständig. In der Abteilung sind ungefähr 20 Mitarbeiter angestellt, die sich auf verschiedene Geräte spezialisiert haben und von Firmware bis hin zu grafischen Benutzeroberflächen verschiedene Schwerpunkte besitzen.

# Praktikumsablauf

|  |  |
| --- | --- |
| 3.10. - 5.10. | Erlernen von git-Grundlagen |
| 6.10. - 11.10. | Kennenlernen von epBlue, Build-Prozesse überprüfen/Anleitung schreiben, optische Fehler aufdecken |
| 11.10. - 14.10. | HexToBin-Converter schreiben (C++) |
|  | Commit-Logger (Python) |
|  | Release-Skript (Python) |
|  | 3D-Modelle (Sketchup) |
|  | Coded-UI-Tests (C#) |
|  | MixRack-Editor entwerfen (C#,Wpf) |
|  | Barcode-ID-Tests (C#) |
|  |  |

# Detaillierte Aufgabenbeschreibungen

## Erlernen von git-Grundlagen

Der Start des Praktikums war erst einmal zum Kennenlernen der Software, Einrichten des Computers und Erlernen der wichtigsten Grundlagen vorgesehen. Aus diesem Grund startete ich mein Praktikum mit dem Kennenlernen von git. Git ist ein Softwareverwaltungstool um einfache, verteilte Entwicklung von Dokumenten zu gewährleisten. Es gibt verschiedene Anbieter die, die Richtlinien von git einsetzen und online Interfaces zur Verfügung stellen. Ich hatte bereits davon gehört, mich aber nie richtig damit beschäftigt, weshalb die Aufgabe des Kennenlernens der Möglichkeiten, die dieses Versionsverwaltungstool bietet, einen sinnvollen Start darstellte. Mit Hilfe von git werden in der Softwareabteilung die erstellten Programme verwaltet und gesichert. Deshalb können zu jedem Zeitpunkt alte Versionen des Programms wiederhergestellt werden und Fehler einfacher identifiziert werden. Außerdem bietet git die Möglichkeit parallel an den selben Dokumenten zu arbeiten und dieses hinterher einfach zu vergleichen und zu "mergen", was bedeutet, dass Änderungen von beiden Bearbeitern im Dokument übernommen werden, ohne dass eine Änderung verloren geht. Das passiert meistens automatisch, nur wenn an der selben Stelle etwas verändert wurde, wird diese Stelle angezeigt und man kann sich für eine Lösung entscheiden. Viele dieser Funktionen werden von grafischen Programmen, die eine Einbindung der Anbieter von git erlauben, verbessert oder überhaupt erst ermöglicht. Da git eigentlich mit Konsolenbefehlen verwendet wird, bieten solche Programme einen deutlichen Mehrwert und können überaus nützlich sein. Für mich war diese erste Phase sehr interessant, denn bisher habe ich noch nicht den Sinn gesehen, diese Technik bei meinen bisherigen Projekten einzusetzen und habe mich daher noch nicht sehr ausgiebig damit beschäftigt. Jetzt, da ich dieses System über drei Monate verwendet habe, sehe ich es auch für kleinere Aufgaben sehr nützlich und werde es auf jeden Fall auch weiter privat einsetzen.

## Kennenlernen von EpBlue

Nach diesen ersten Erkenntnissen bekam ich die Aufgabe, die Software, die in meinem Büro entwickelt wird, kennenzulernen und meinen PC für diese Programme und deren Entwicklung einzurichten. Da hierfür noch keine vollständige Anleitung zur Verfügung stand, sollte ich diese währenddessen für künftige neue Mitarbeiter erstellen. Das war eine gute Aufgabe, um mich näher mit der Software auseinandersetzen zu müssen und gleichzeitig aber noch nicht so tief in die Programmierung einzusteigen. Ich sehe das als einen sehr wichtigen Aspekt bei der Mitarbeit an einem bereits bestehenden Projekt, als erstes genügend Zeit und Hilfe zu bekommen um den Nutzen und die Funktionen, die das Programm bietet dem Benutzer bietet, kennenzulernen. Das Aufdecken von offensichtlichen Fehlern, wie zum Beispiel unerwartetes Verhalten oder fehlende Animationen bzw. 3D-Modelle, hat das Kennenlernen dann noch etwas gerechtfertigt. Jeder Entwickler in diesem Büro besitzt mindestens zwei Rechner, wobei auf einem Windows und auf dem Anderen Linux installiert ist. Das ist notwendig, da Teile die Firmware auf einem Linux Rechner entwickelt wird und die Benutzerschnittstelle in einer Windowsumgebung programmiert wird. Schwierigkeiten bei der Einrichtung der Rechner hatte ich meistens bei meinem Linux-PC. Da ich bisher nur wenig mit Linux in Berührung gekommen bin und eigentlich lieber auf Windows entwickle, war auch diese Grundlage neu für mich und ich musste mich erst einmal damit zurechtfinden, was aber auch recht zügig geklappt hat. Hilfe wurde mir häufig angeboten und auch bei Nachfragen oder Problemen habe ich immer freundliche Antworten bekommen, auch bei den offensichtlichsten Fehlern.

## HexToBin-Converter

Als erste richtige Programmieraufgabe, und wahrscheinlich auch zum ersten Ausloten meiner Fähigkeiten zu programmieren, bekam ich den Auftrag ein kleines Tool zu ersetzen, bei dem in unregelmäßigen Abständen Fehler während des Build-Prozesses ausgegeben wurden. Das Programm hatte die Aufgabe eine Konvertierung von Dateien im hex-Format in ein binäres Format umzuwandeln. Da es noch von früher stammte, war es in reinem C geschrieben und sollte nun mit C++ umgestaltet werden. Außerdem war es nicht gut geschrieben und für jemanden, der schnell etwas darin überprüfen wollte nicht lesbar. Das sollte ich ebenfalls beheben. Der erste Teil dieser Aufgabe bestand darin, herauszufinden, wie das hex-Format aufgebaut ist. Es ist ein Format von Intel, welches zur Speicherung und Übertragung von Binärdaten gedacht ist. Die Schwierigkeit bestand also darin, dieses Format zu entschlüsseln und nur die Nutzdaten in eine neue Datei zu schreiben. Als Grundlage diente mir das alte Programm. Daran konnte ich erkennen, wie die Daten empfangen und wieder geschrieben wurden, da die neu geschriebenen Daten im Binärformat ebenfalls eine bestimmte Struktur erforderten, die allerdings nicht gesondert spezifiziert war. Ich brauchte am Anfang etwas Zeit um mich an das ständige Programmieren zu gewöhnen, da man an der Uni doch öfters Pausen bekommt und auch nicht mehrere Tage nur an einer Aufgabe sitzt. Als ich eine fertige Version geschrieben hatte und mit von mir erstellten Beispieldaten getestet hatte, dass es genau das richtige Format ausgibt, habe ich ein paar Dateien bekommen, die im echten Build-Prozess erstellt wurden, mit denen ich ausführlicher testen konnte. Leider musste ich feststellen, dass meine Lösung den exakt gleichen Fehler an selber Stelle ausgab, wie die vorher genutzte. Mittels eines Batch-Skriptes, welches ich mir geschrieben hatte, konnte ich diese These mit sehr vielen Daten belegen. Mein Programm löste also das Problem nicht, ich hatte jetzt aber durch den von mir verständlicher geschriebenen Code die Möglichkeit, den Ursprung des Fehlers zu finden. Mit Hilfe von geskripteten Tests konnte ich feststellen, dass es schon vor dem Programm zu Inkorrektheiten kam. Die Datei war an manchen Stellen nicht richtig im hex-Format formatiert und so kam es zu Checksummen-Fehlern, bei der Überprüfung auf die Korrektheit der Daten. Da ich nicht tiefer in den Build-Prozess eingreifen sollte, war hier meine Arbeit beendet. Ich konnte mit dem Finden des Fehlers und dem neugeschriebenen, übersichtlicheren Code meine Aufgabe erfüllen und meinen Betreuern einen genaueren Anhaltspunkt geben, wo weitergesucht werden musste.

## Commitlogger / Changelog-Generator

Meine nächste Aufgabe war dann schon um einiges umfangreicher. Ich sollte mit dem am Anfang des Praktikums erworbenen Wissen über git, einen Commit-Logger schreiben. In git werden Änderungen an Dateien immer mit Hilfe eines Commits in das Dateisystem eingetragen. Erst mit diesem Commit werden die Änderungen in dem Projekt für alle, die darauf Zugriff haben sichtbar und übernommen. Anhand der Commits, die unter anderem aus einer Nachricht, sowie dem Autor und den geänderten Dateien, kann man nachvollziehen, zu welchem Zeitpunkt und von wem Änderungen gemacht wurden. So kann man zum Beispiel durch diese Commits zurück zu einem Stand gehen, bei dem ein bestimmter Fehler noch nicht aufgetreten ist und kann so die genauen Änderungen erkennen, die zu diesem Fehler geführt haben.

Mit Hilfe des geforderten Loggers sollte die Liste an Commits bestimmter Projekte durchsucht werden und Versionscommits, das bedeutet Commits, die von den Erstellern als neue Version gekennzeichnet wurden, erkannt werden und in eine .txt-Datei geschrieben werden. Dabei gab es mehrere Projekte, die dann nach dem Finden des Zeitpunktes der Version, durchsucht werden sollten und sämtliche Commits, die im Zeitraum von letzer bis zur gefunden Version getrackt wurden, sauber aufgelistet werden und am Besten noch nach Kategorien geordnet werden. Es sollte also ein vernünftiger Changelog erstellt werden, in dem die Versionen jeweils als Überschrift dienten.

Bald habe ich gemerkt, dass ich doch noch nicht alle Features von git kennengelernt und einiges auch noch nicht richtig verstanden hatte. Doch mit der neuen Aufgabe hatte ich einen praktischen Grund mich intensiver damit zu beschäftigen.

Eine weitere Hürde war die Programmiersprache, die ich verwenden sollte. Denn mit Python hatte ich vorher noch gar keine Berührung gehabt und musste mich also auch dort mit etwas Neuem zurechtfinden. Da Python allerding häufig als die beste Sprache für Anfänger deklariert wurde, habe ich mir keine großen Sorgen darüber gemacht. Python ist tatsächlich sehr einfach aufgebaut und praktisch in vielen Dingen, da vieles schon durch fertige Bibliotheken zur Verfügung steht und einfach benutzt werden kann. Vor Allem das Lesen und Schreiben von Dateien sowie das Bearbeiten von Strings erschien mir sehr einfach und praktisch. Die meisten Schwierigkeiten beim Erlernen hatte ich mit dem fehlenden Scope: Variablen, die in Schleifen deklariert werden können überall verwenden werden, auch wenn die Schleife beendet wurde. Das gibt es so in den von mir bisher kennengelernten Sprachen nicht. Außerdem ungewöhnlich fand ich es keine main-Funktion zu haben, sondern diese einfach in das Dokument ohne Funktionsdeklaration zu schreiben. Das ist jedoch dem Script-Charackter von Python zuzuschreiben, da es für Scripte entworfen wurde, bei denen eine Anweisung nach der Anderen ausgeführt wird.

Nachdem ich mit Python ein bisschen getestet hatte, konnte ich die eigentliche Aufgabe in Angriff nehmen. Die Befehle um git nutzen zu können, hatte ich schnell gefunden, da git eigentlich für die Konsole entworfen wurde, konnte ich einfache Befehle von Pyhon nutzen um Konsolenanweisungen auszuführen. Schwierigkeiten machte mir die unterschiedliche Formatierung der Commitnachrichten. Da sich die Ersteller erst vor kurzer Zeit auf ein einheitliches Format geeinigt hatte, konnte ich ältere Versionen nicht erkennen und musste so einige Ausnahmen aufnehmen. Danach ging die Erkennung jedoch gut von der Hand und ein erstes Changelog wurde relativ schnell erstellt. Ein neues Problem war jedoch die Einsortierung der Commits in Kategorien. Die geforderten Kategorien sollten sein: changes, features, bugfixes, deletions und known bugs. Als erstes habe ich Schlagwörter verwendet, die in den Nachrichten gesucht wurden und, falls gefunden, in die jeweilige Kategorie einsortiert. Dabei sind allerdings einige falsch sortierte Commits vorgekommen, da die Kategorien in dieser Hinsicht ineinander greifen. So kam es vor, dass Commits, die Beschreibungen wie zum Beispiel "tool change corrected" enthielten, in die Kategorie "change" gefiltert wurden, dies jedoch eigentlich ein Bugfix war. Da bei vielen Commits am Anfang bereits die Bedeutung des Inhalts stand, bin ich dazu übergegangen nur auf den Anfang der Nachricht zu schauen. Als ich meinen Betreuern von meinem Problem und meiner Lösung berichtete, waren sie begeistert von der Idee, eine Richtlinie zu erstellen, wie eine Commitnachricht zu beginnen hatte, damit sie richtig einsortiert würde. Das hatte ich schon während meinen Versuchen getan und konnte sie jetzt offiziell bekannt geben. So wurde der Changelog richtig erstellt und auch die Kategorien wurden richtig bedient.

## Release-Skript

Als fortführende Aufgabe sollte ich nun ein Release-Skript schreiben, in dem ein solcher Versions-Commit, wie oben beschrieben, erstellt und außerdem die Versionsnummer in verschiedenen Dateien erhöht wird. Da es immer zwei Versionsnummern gab, einmal die Firmware und einmal die Software Version, war dies nicht ganz so einfach. Ich sollte mit meinem Skript die aktuelle Nummer automatisch ermitteln und möglichst ohne Nutzereingaben die Aufgabe erfüllen. Hier hatte ich Schwierigkeiten, mit den Features von git. Da es möglich ist auf unterschiedlichen Branches zu arbeiten, gab es auch mehrere Möglichkeiten, wo die letzte Version zu finden ist. Mit meinen selbstgeschriebenen Funktionen, die git über die Kommandozeile bedienten, hatte ich hier keinen Erfolg mehr, da für mich unverständlicherweise das Timing nicht stimmte. So konnte kurz vorher eine neue Versionsnummer vom Nutzer geteilt werden, aber in meinem Skript wurde diese nicht erkannt. Nach langem Suchen, wo der Fehler sein konnte, entschied ich mich dafür eine externe Möglichkeit für git-Repositories in Python zu nutzen. Nach dem Umbau des Codes um die externe Bibliothek zu nutzen, war der Fehler verschwunden, jedoch kam es zu anderen Fehlern, die aber nicht so schlimm waren. Das Problem von externen Bibliotheken ist, dass man hier nicht die volle Kontrolle über die Funktionen hat. So wurden Dinge überprüft, die in meinem Fall unerheblich waren, aber in der Bibliothek zu Fehlern führten. Deshalb musste ich weitere Korrekturen vornehmen um schließlich ein funktionierendes Skript vorweisen zu können. Meine Betreuer waren soweit zufrieden, jedoch gab es weiterhin Ausnahmen, in denen das Programm abgebrochen wurde. Erst einmal sollte ich mich jedoch nicht weiter darum kümmern, da eine andere Aufgabe dringender nötig wurde.

## 3D Modelle

Es sollte in Kürze ein neues Feature der Software auf den Markt gebracht werden, nämlich der Simulator der EpMotion. Mit Hilfe dieses Simulators sollte es möglich sein die Methoden, die man in der EpBlue-Software erstellt hatte zu testen, ohne eine Maschine angeschlossen zu haben. Er beinhaltet 3D-Modelle von aller Labware, die in EpBlue standardmäßig gesetzt werden kann. Außerdem wird die komplette Maschine in 3D dargestellt, sowie die erstellten Abläufe animiert. Es gab bereits einmal einen 3D-Simulator, der jedoch jetzt in die Jahre gekommen war und außerdem einige Probleme erzeugte, weshalb er ersetzt werden musste. Die meisten 3D-Modelle waren also bereits vorhanden. Sie wurden damals von einer externen Firma erstellt und sollten nun überarbeitet werden. Da es relativ viel neue Labware gab, fehlten Modelle und einige waren kaputt, sahen also nicht mehr so aus, wie sie sollten. Meine Aufgabe war es nun, die fehlenden Modelle zu entdecken und durch neue zu ersetzen. Dazu wurde mir ein Messschieber gegeben, mit dem ich im Haus vorhandene Labware ausmessen und in 3D umsetzen sollte. Als Empfehlung mit welchem Programm ich das machen sollte, wurde mir Sketchup empfohlen.

Schwierigkeiten machten hier die alten Modelle, welche in einem Format vorlagen, das nicht direkt von Sketchup geöffnet werden kann. Das Format .3ds, habe ich nach kurzer Recherche herausgefunden, ist veraltet und wird nur noch aus historischen Gründen verwendet. Da das Format der Dateien nicht geändert werden sollte, musste ich also auch weiterhin in diesem entwickeln. Glücklicherweise gab es eine Erweiterung für Sketchup mit dem 3ds-Dateien im- und exportiert werden konnten. Jedoch wurden so keine Positionsdaten mitgeliefert, sodass ich die Modelle immer wieder neu positionieren musste, sobald ich sie importiert hatte. Außerdem waren die alten Modelle sehr groß, mit vielen kleinen Details, die gar nicht nach außen sichtbar waren. Deshalb dauerte es sehr lange, bis sie geladen werden konnten und sie waren sehr unübersichtlich durch viele unnötige Kanten und kaputte Formen. Alte Modelle zu bearbeiten war deshalb ziemlich unmöglich, weshalb ich mich entschied, die kaputten ganz neu zu machen. Die Labware, die ich dafür benötigte, bekam ich aus dem Labor, in dem diese täglich eingesetzt werden. Das Modellieren der Labware beanspruchte viel Zeit und war nicht sehr abwechslungsreich, aber ein fertiges Modell geschaffen zu haben, war ein gutes Gefühl. Am Ende hatte der 3D-Simulator rechtzeitig alle Modelle zur Verfügung und sah sehr gut aus.

## Modell-Tester

Da nun alle Modelle für die Labware vorhanden war, konnten andere Mitarbeiter am Simulator damit beginnen, deren Positionierung zu überprüfen und gegebenenfalls Verschiebungen oder andere Änderungen vornehmen. Um am Ende die korrekte Position von jeder einzelnen Labware automatisiert überprüfen zu können, sollte ich das erste Mal mit der EpBlue-Software arbeiten. Es ging darum eine Testmethode zu schreiben, in der immer eine Labware auf der Maschine platziert wird. Dies sollte mit jeder Labware passieren auch bei Zusammengesetzter sollte dies gehen. Anschließend sollten diese nacheinander ausgeführt und jeweils ein Screenshot gemacht werden, der einen einmaligen und eindeutigen Namen besitzen sollte. Die Screenshots wurden von einem anderen Mitarbeiter übernommen, deshalb konnte ich mich auf die Methodengenerierung konzentrieren.

Auch hier hatte ich es mit einer neuen Programmiersprache zu tun, da ich bisher noch nicht mit C# gearbeitet hatte. Doch ich konnte mich sehr schnell einfinden, da die Ähnlichkeiten zu Sprachen wie Java oder C++ sehr deutlich sind. Für meine Zwecke reichte anfangs die Grundkenntnisse, die ich durch Anschauen des schon geschriebenen Codes erlangte.

Da ich das erste Mal in den Code eines schon bestehenden großen Softwareprojektes geschaut habe, war ich anfangs sehr überfordert. Schon die Ordnerstrukturen zu verstehen und einzelne Funktionen zu suchen war für mich neu. Da in der Firma Visual Studio benutzt wird, hatte ich allerdings einiges an Unterstützung durch die IDE. Mit ein bisschen Erfahrung ging es dann auch deutlich schneller eine von einer Klasse benutzte Funktion nachzuverfolgen und zu finden. Hier hätte ich mir am Anfang mehr Einweisung gewünscht, um den Aufbau der Software von Beginn an besser zu verstehen. So hatte ich es aber auch nach einiger Zeit weitestgehend erkannt, vor Allem aber wie und wo ich nach den für mich relevanten Dingen zu suchen hatte.

Probleme gab es bei dieser Aufgabe nur beim Zusammensetzen der Labware. Da es viele verschiedene Möglichkeiten der Zusammensetzungen gibt, musste ich einige Ausnahmen einfügen. Um zu erkennen, wie Labware zusammengesetzt werden soll, musste ich die verschiedenen Eigenschaften der jeweiligen Klassen, die die Labware repräsentieren, wissen und auslesen können. Da es sehr viele Eigenschaften gab, die untereinander noch einmal in Klassen aufgeteilt wurden und außerdem auf Interfaces zurückgegriffen haben, war es nicht immer einfach eine Eigenschaft finden. Schwierig war es auch deshalb, weil ich in vielen Fällen gar nicht wusste, dass eine bestimmte Eigenschaft existiert, und nur durch Zufall auf viele Dinge gestoßen bin. Aber da die Aufgabe noch relativ klein war, konnte ich auch hier ziemlich schnell Erfolge erzielen.

Durch den Screenshot-Service, den ein anderer Mitarbeiter anschließend mit meinen Testmethoden verbunden hat, wurde aus unserer Arbeit ein automatisierter Test, der anschließend die gemachten Bilder mit Referenzbildern verglich und das Ergebnis loggte, damit daraus erkenntlich wurde, ob eine Labware durch die letzten Änderungen verschoben wurde. Dieser Test wird mit einigen anderen Tests jetzt vor jedem neuen Release durchlaufen.

## Mischrack-Editor

Als letzte und auch als größte Aufgabe sollte ich mir einen neuen Teil der EpBlue-Software ausdenken und anschließend selbst programmieren. Ich bekam eine genauere Spezifikation, was das neue Unterprogramm alles können soll und setzte mich dann als erstes an den Entwurf einer Oberfläche auf dem Papier.

Das Unterprogramm sollte hauptsächlich der bequemeren Zusammensetzung von Racks mit Tubes dienen. Es gibt verschiedene Racks, die unterschiedlich viele Löcher haben können, in die verschiedene Tubes gesetzt werden können. Bisher war es nur möglich ein Rack auszuwählen und eine Art Tube, die eingesetzt werden sollen. Jetzt sollte es aber so sein, dass es möglichst komfortabel möglich sein soll in jedes Loch ein eigenes Tube zu setzen und dem Kunden so mehr Freiheiten zu lassen, wie er ein Rack designen möchte. Das Hauptproblem an der alten Variante war, dass es sogenannnte Mischracks gibt, die verschieden große Löcher haben und somit nicht einheitlich mit einem Tube bestückt werden können. Wichtig war auch, dass die Anwendung mit Drag & Drop benutzbar sein sollte, da alle Teile der Software diese Technik bisher nutzen und es sonst verwirrend für den Nutzer sein könnte.

Nachdem ich den ersten groben Entwurf meinen Betreuern gezeigt hatte und sie diesen für gut befunden hatten, machte ich mich wieder einmal daran eine neue Programmiertechnik zu erlernen. Denn für die grafischen Benutzeroberflächen wurde in EpBlue Wpf verwenden, was der Nachfolger von Windows Forms ist und ein Mix aus C# und XAML ist. Die Oberfläche an sich wird dabei mittels XAML beschrieben, jegliche Logik in einer anderen Datei, dem "Code behind". Um damit meine ersten Entwürfe machen zu können, fing ich in einem eigenen Projekt an grundlegende Dinge zu entwickeln und einige Tutorials aus dem Internet umzusetzen. Nachdem ich mich hier einigermaßen sicher fühlte, machte ich mich daran eine weitere spezielle Technik anzuschauen. MVVM ist ein Pattern, dass häufig in der GUI-Entwicklung verwendet wird und was auch von WPF empfohlen wird. Dabei geht es darum die Logik, also das Programm an sich, von der Oberfläche zu trennen, so dass man nicht immer beides ändern muss, sobald eine Änderung notwendig ist. Um das zu erreichen gibt es in WPF die Möglichkeit des Bindens. Man bindet dabei eine Variable aus der Logik mit der Oberfläche um diese verändern zu können. Das bedeutet zum Beispiel, dass man in der Logik, dem Viewmodel, eine Liste an Objekten hat, die aus dem Programm stammen, dem Model, und diese an eine Listview auf der Oberfläche, der View, bindet um diese dort anzuzeigen und damit interagieren zu können. Um Interaktionen vom Nutzer an das Viewmodel zurückzusenden, werden sogenannte Commands verwendet, die auf ein bestimmtes Ereignis warten und immer dann ausgeführt werden, sobald dieses ausgelöst wird.

Als ich diese Mechanismen beherrscht habe, machte ich mich daran einen ersten Entwurf meiner GUI zu entwickeln, die auf grundlegende Interaktionen reagiert hat. Meine Idee war, ein Viereck in der Mitte darzustellen auf dem einige Kreise positioniert waren. Das sollte für den Nutzer das Rack von oben gesehen darstellen. Mit einem Klick auf die einzelnen Löcher konnte man diese markieren und über eine Liste auf der rechten Seite ein Tube auswählen, welches in diese Löcher hineinpasst. Am Anfang habe ich viel zu viele meiner Ideen auf einmal versucht umzusetzen. Nach einem Test eines Kollegen stellte sich das heraus und ich kürzte die Funktionen und entschied mich grundlegend für die oben beschriebene. Drag & Drop ließ ich anfangs noch außen vor, da mir mein Betreuer gesagt hat, dass dies nicht so einfach sei und ich mich erstmal auf die andere Variante konzentrieren soll.

Mein nächster Schritt war die Integration in die EpBlue-Software. Damit konnte ich die Datenstrukturen und Funktionen nutzen, die ich später sowieso nutzen sollte. Durch meine Erfahrungen vom Modell-Tester, den ich davor geschrieben hatte, war es kein großes Problem für mich die Integration zu schaffen. Mit den Daten von EpBlue wurde mein Programm schnell zu dem, was ich mir vorgestellt hatte. Nach einiger Zeit kam ich an den Punkt an dem alle Features implementiert waren und ich "nur noch" nach Fehlern suchen musste. Dass das einen Großteil der Arbeit eines Programmierers ausmacht, hatte ich schon gemerkt, als ich mich bei meinen Kollegen über deren Aufgaben zur Zeit umgehört habe. Auch bei mir dauerte es noch ein paar Tage bis ich behaupten konnte, dass mein Progamm jeden von mir getesteten Fall abfängt und keine Fehlermeldungen mehr auftauchen. Meine Betreuer waren mit dem Ergebnis sehr zufrieden und mein Teilprogramm wird wohl bald ein Teil der Software werden, die dem Kunden ausgeliefert wird.

# Zusammenfassung