Konzipierung und Erstellung eines Parametertabellenkonfigurators und Praxisnahe Mitarbeit im releasten Produkt LC-VISION

Projektarbeit T1000
des Studiengangs Informatik
an der Dualen Hochschule Baden-Württemberg Ravensburg

von

Janik Frick

Abgabe: 06.10.2022

Bearbeitungszeit: 17 Wochen
Kurs: TIT21, Matrikelnummer 4268671
Dualer Partner: Blum-Novotest GmbH, Standort Grünkraut
Betreuer: Mallmann, Guilherme, Dr.-Ing.

Erklärung

gemäß Ziffer 1.1.13 der Anlage 1 zu §§ 3, 4 und 5 der Studien- und Prüfungsordnung für die Bachelorstudiengänge im Studienbereich Technik der Dualen Hochschule Baden-Württemberg vom 29.09.2017.
Ich versichere hiermit, dass ich meine Projektarbeit mit dem Thema: Konzipierung und Erstellung eines Parametertabellenkonfigurators und Praxisnahe Mitarbeit im releasten Produkt LC-VISION selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Janik Frick

_____ _____

Ort, Datum Unterschrift

Sperrvermerk

gemäß Ziffer 1.1.13 der Anlage 1 zu §§ 3, 4 und 5 der Studien- und Prüfungsordnung für die Bachelorstudiengänge im Studienbereich Technik der Dualen Hochschule Baden-Württemberg vom 29.09.2017:
„Der Inhalt dieser Arbeit darf weder als Ganzes noch in Auszügen Personen außerhalb des Prüfungsprozesses und des Evaluationsverfahrens zugänglich gemacht werden, sofern keine anders lautende Genehmigung vom Dualen Partner vorliegt.“

Janik Frick

_____ _____

Ort, Datum Unterschrift

Inhaltsverzeichnis

Abkürzungsverzeichnis

Blum	Blum-Novotest GmbH
GUI	Grafische Benutzeroberfläche
MVP	Minimum Viable Product
UML	Unified Modeling Language

Abbildungsverzeichnis

Abstract

By developing a „Parametertabellenkonfigurator“ the dependency on experience of the persons knwoledge about configuring Blum products could be reduced. This is because the description for the current selected parameter is selected by the software and shown in the same window as the file with parameters which is currently worked on. By that the risk of reading the wrong description is eliminated. It also reduces the time in focus, because of less time needed to configure the parameters. These factors lower the risk of errors.
By the exmaple of the collaboration on the „LC-VISION“-Software a process of finding orientation in unknown, existing structures is described.

Zusammenfassung

Mit Hilfe der Entwicklung des „Parametertabellenkonfigurators“kann die Abhängigkeit von der Erfahrung, mit der Einrichtung von Produkten von Blum, des Inbetriebnehmers reduziert werden. Die Beschreibungen der einzelnen Parameter werden von der Software automatisch gesucht und im selben Fenster angezeigt, wie die Parametertabelle, die aktuell bearbeitet wird. Diese Automatisierung minimiert das Risiko die falsche Beschreibung zu lesen. Außerdem wird die Zeit reduziert, in der die Konzentration gehalten werden muss, da die Zeitersparnis bei der Suche nach der richtigen Beschreibung, die Dauer der Konfiguration verkürzt.
Am Beispiel der Mitarbeit an der „LC-VISION“-Software wird eine Möglichkeit beschrieben, mit der man Orientierung in unbekannten, schon existierenden Strukturen finden kann.

1 Einleitung

Die Blum-Novotest GmbH (Blum) ist im Bereich Maschinenbau für Mess- und Prüftechnik tätig.
Produkte von Blum werden in vielen anspruchsvollen Industrien im Bereich der Qualitätssicherung eingesetzt. Die Kunden kommen unter anderem aus der Automobil-, Luftfahrt- und der Medizinbranche.
An die Produkte dieser Industrien werden höchste Qualitätsansprüche gestellt. Daraus resultieren auch für die Produkte der Firma Blum höchste Ansprüche.
Um diese Ansprüche erfüllen zu können, werden sowohl bestehende Produkte laufend optimiert und weiterentwickelt, als auch neue Produkte entwickelt.
Ziel der Arbeit ist es die vielfältigen Aspekte der Softwareentwicklung kennenzulernen und einen Einstieg zu finden.
Um dieses Ziel zu erreichen, besteht die Arbeit aus drei Teilen. Im ersten Teil wird ein allgemeiner Überblick über das Vorgehen und die Schritte der Softwareentwicklung gegeben. Im zweiten Teil sollen mit der Entwicklung eines „Parametertabellenkonfigurators“ die verschiedenen Aufgaben der Softwareentwicklung an einem praktischen Beispiel veranschaulicht werden. Durch die Mitarbeit am schon bestehenden Produkt „LC-VISION“ wird das Einarbeiten in unbekannte Strukturen thematisiert.

2 Softwareentwicklung

Das Schreiben von Code ist nur einer von vielen Teilen der Softwareentwicklung. Das Erfassen des Problems, das Design und die Produktpflege sind ebenfalls Teil der Softwareentwicklung[1]. Der Prozess beginnt mit der Anforderungsanalyse. Auf Basis der Anforderungen wird ein Grobentwurf entwickelt, der als Grundlage für die Implementierung dient. Der Prozess endet mit der Produktpflege und dem Service für das Produkt.

2.1 Anforderungsanalyse

Eindeutige und präzise Anforderungen bilden die Grundlage für ein funktionierendes Projekt[2]. Daraus folgt, dass es nicht ausreichend ist, die Liste der Anforderungen zu lesen. Anforderungen müssen analysiert und mit weiterem Wissen über das Projektumfeld kombiniert werden.
Dafür gibt es mehrere Techniken, die eingesetzt werden können.

2.1.1 Interviews

Interviews bieten die Möglichkeit in direkten Gesprächen mit Personen aus dem Umfeld des Projekts Informationen zu sammeln. Diese Informationen können Risiken sein, die zu berücksichtigen sind, Schwierigkeiten bei der aktuellen Vorgehensweise, sowie Abläufe die beibehalten werden sollten.
Der Erfolg dieser Vorgehensweise hängt dabei von allen beteiligten Personen ab. Fehlt Wissen über das Umfeld des Projekts kann es zu Schwierigkeiten beim Sammeln der Fragen kommen, wodurch manche Aspekte nicht berücksichtigt werden. Auch die Befragten können Probleme haben ihre Gedanken und ihr Wissen wiederzugeben [3]. Das kann die Auswertung der Antworten erschweren.

2.1.2 Fragebögen

Fragebögen geben den befragten Personen die Möglichkeit ohne direkte Beeinflussung durch den Ersteller der Fragen, Informationen bereitzustellen. Zusätzlich reduzieren sie den Zeitdruck unter dem geantwortet werden soll und alle Befragten antworten auf die gleiche Fragestellung.
Fragebögen können eingesetzt werden um Annahmen zu bestätigen oder nach Meinungen und Vorschlägen zu fragen[3].
Bei Einsatz dieser Methode ist darauf zu achten die Fragen offen zu formulieren, um die Antworten nicht in eine Richtung zu lenken.

2.1.3 Beobachtung

Die Beobachtungsmethode dient dazu aus beobachteten Vorgängen Informationen zu sammeln.
Diese Beobachtung kann offen oder verdeckt durchgeführt werden. Bei der verdeckten Vorgehensweise ist das beobachtete Verhalten natürlicher und realistischer, als bei der offenen Vorgehensweise[4].
Mit diesem Vorgehen lassen sich Abläufe bei Aufgaben nachvollziehen. Gründe für dieses Abläufe sollten durch andere Techniken in Erfahrung gebracht werden.

2.1.4 Anforderungstypen

Anforderungen können in zwei Typen unterteilt werden: Die funktionalen und die nicht funktionalen Anforderungen. Dabei ist zu beachten, dass keine eindeutige Trennung zwischen den beiden Anforderungstypen sinnvoll ist, wie die Ergebnisse der Untersuchung von Eckhardt et al. zeigen[5].
Funktionale Anforderungen geben vor, für welche Aufgaben das Produkt geplant wird und welche Funktionen dafür benötigt werden. Die Validierung dieser Anforderungen ist gegeben, da am Ende eindeutig ist, ob eine Funktionalität verfügbar ist.
Die nicht funktionalen Anforderungen geben vor, mit welchem Vorgehen und welchen Tools eine Funktion umgesetzt werden soll[5].
Auch Anforderungen für Performance wie beispielsweise Reaktionszeiten, vorgegebene Schnittstellen zu anderer Software, wie dem Betriebssystem, können Teil der nicht funktionalen Anforderungen[5]. Die gegebenen Beispiele sind keine vollständige Aufzählung, da es sehr viele Bereiche gibt, aus denen nicht funktionale Anforderungen stammen können. Nicht funktionale Anforderungen sind häufig unpräzise formuliert und somit problematisch in der Validierung[5].

2.2 Dokumentation

Die Dokumentation von Softwareprojekten ist neben der eigentlichen Entwicklung ein relevanter Teilaspekt. Die Dokumentation dient zur Sicherung von Wissen und Grundlagen von Entscheidungen.
Die Dokumentation kann in zwei Bereiche unterteilt werden. Die Dokumentation des Codes und die Dokumentation für den Anwender[6].

2.2.1 Sourcecode-Dokumentation

Diese Form der Dokumentation soll Wissen über die Software beinhalten. Dazu zählen Grundlagen für Entscheidungen der Softwarearchitektur und die Aufgaben von Funktionen, Methoden und Variablen.
Die Relevanz besteht darin Wissensverlust zu verhindern, der entsteht, wenn der Entwickler nicht zur Verfügung steht. Dieser Verlust beeinflusst die Einarbeitungsdauer eines neuen Mitarbeiters und somit auch die Produktpflege.
Diese Problematik der mangelnden Softwaredokumentation ist in agilen Umfeldern bekannt. Sich ändernde Anforderungen verursachen die Notwendigkeit die Software anzupassen, was Anpassungen der Dokumentation erfordert. Geänderte Anforderungen bringen auch neue kurzfristige Aufgaben mit sich, denen eine höhere Priorität beigemessen wird, als der Aktualisierung der Dokumentation. Kommen viele Änderungen in kurzer Zeit stauen sich die anstehenden Änderungen auf und sorgen für einen hohen Zeitaufwand der Dokumentation. Dies resultiert oftmals in einer vernachlässigten Dokumentation.
Weitere potenzielle Schwierigkeiten resultieren aus der Ansicht, dass der Code bereits eine ausreichende Dokumentation für diesen Zweck darstellt und dass Informationen häufig informell und verbal kommuniziert werden, was dazu führen kann, dass diese Informationen nach der Verarbeitung nirgends festgehalten sind und dadurch verloren gehen können[6].

2.2.2 User-Dokumentation

Der Dokumentation für die Anwender, Servicepersonal und Support-Mitarbeiter wird meist mehr Bedeutung beigemessen, da sie für den Einsatz des Produktes relevant ist.
Diese Dokumentation beinhaltet Fehlermeldungen, Voraussetzungen für die Installation und Informationen für die Verwendung. Probleme mit dieser Dokumentation können Schwierigkeiten im Kundenservice verursachen. Probleme dieser Art können einen direkten Einfluss auf den Erfolg eines Produktes oder der Firma haben. Diese möglichen Konsequenzen sorgen dafür, dass dieser Dokumentation mehr Bedeutung beigemessen wird[6].

2.3 Grobentwurf

Beim Grobentwurf werden die Hauptbestandteile der Software und die Softwarearchitektur geplant. Dabei sollte der Kunde möglichst miteinbezogen werden, denn der Kunde hat großen Anteil am Gelingen des Projekts[7].
Um den Kunden einzubeziehen, können Prototypen eingesetzt werden.
Hat das Programm eine Grafische Benutzeroberfläche (GUI) so kann der Grobentwurf unterschiedlich geplant werden.

2.3.1 GUI mit Bildern

Der Grobentwurf der GUI kann mit Hilfe von Bildern in einem beliebigen Bilderstellungsprogramm nachgebildet werden. Die erstellten Entwürfe dienen als Gesprächsgrundlage für eventuell notwendige Veränderungen. Hierbei ist zu beachten, dass die Simulation von Verbindungen zwischen GUI-Elementen zeitaufwendig oder gar nicht möglich ist.

2.3.2 Prototypen

Ein Prototyp ist eine funktionierende, begrenzte Version der Anwendung, die als Basis für Gespräche und die Entwicklung weiterer Prototypen dient[8].
Dieser Prototyp kann durch Software, oder mit einem Mockup-Tool[1], oder auch in einer Software für Präsentationen, realisiert werden. Ein auf Software basierter Prototyp erfordert die Mitarbeit der Entwickler im Design Prozess.

2.3.3 User-Tests

Bei User Tests wird einer oder mehreren Testpersonen, abhängig von der Verfügbarkeit, der aktuelle Prototyp, siehe 2.3.2, vorgelegt. Dann kann man der Testperson eine konkrete Aufgabe geben, oder sie den Prototyp frei entdecken lassen. Bei beiden Vorgehensweisen werden Probleme dokumentiert. Stehen mehrere Testpersonen zur Verfügung können auch Gruppentests durchgeführt werden. Bei diesen arbeiten mehrere Personen an einer Aufgabe. Bei Einzel-Tests lassen sich Probleme im Ablauf besser erkennen. In Gruppen-Tests hingegen kann besser nachvollzogen werden, welchen individuellen Problemen Benutzer begegnen[9]. Beide Problemquellen sollten für den Entwurf einer intuitiven Benutzeroberfläche berücksichtigt und möglichst behoben werden.
Mit dieser Testmethodik lassen sich nur Prototypen, siehe 2.3.2, testen, da nur hier die Verbindungen funktional sind.

2.3.4 30-Sekunden Tests

Bei 30-Sekunden-Tests bekommen die Testpersonen 30 Sekunden Zeit, um den Prototyp zu erkunden. Nach 30 Sekunden wird ein Fragebogen ausgefüllt, auf dem die Übersichtlichkeit bewertet wird[10].
30-Sekunden Tests lassen sich mit Bildentwürfen, siehe 2.3.1, und mit Prototypen, siehe 2.3.2, durchführen, da es bei beiden Entwürfen möglich ist ein Element für 30 Sekunden zu zeigen.

[1] Tool um eine GUI zu simulieren

2.3.5 Heatmaps

Heatmaptools markieren farbig welche Bereiche einer Benutzeroberfläche oft genutzt werden und welche seltene[11]. Diese Daten geben Informationen darüber ob Kernfunktionen anders positioniert werden sollten.
Diese Methode lässt sich für Bildentwürfe, siehe 2.3.1, und Prototypen, siehe 2.3.2, verwenden.

2.4 Implementierung

Der Aufwand der Implementierung lässt sich nicht zwingend aus dem Vorgehen bei der Entwicklung eines Grobentwurfs ableiten. Wurden die Grobentwürfe, wie in 2.3.1 beschrieben, als einzelne Bilder entworfen, muss zwar von Grund auf neu begonnen werden den Code zu schreiben, es müssen aber auch keine Schwierigkeiten, die durch eine unsaubere Programmierung entstanden sind, gelöst werden.
Wird mit Prototypen, (2.3.2), gearbeitet, kann dies den Implementierungsprozess beschleunigen. Kommt ein Mockup-Tool zum Einsatz muss die Software ebenfalls von Grund auf programmiert werden, aber der Mock-Up Prototyp dient als Hilfestellung um die Übersicht über die notwendigen Verbindungen zwischen GUI-Elementen zu behalten.
Programmierte Prototypen bieten zwar die benötigten Funktionalitäten, sind aber nicht zwingend für den praktischen Einsatz geeignet. Ursachen können in unter anderem in fehlender Kompatibilität der Programmiersprache des Prototyps mit dem Zielsystem liegen. Auch wenn die Programmiersprache übereinstimmt kann es sein, dass für den Einsatz als Prototyp für den Grobentwurf externe Abhängigkeiten nicht berücksichtigt werden. Dies kann dazu führen, dass die Integration externer Abhängigkeiten so aufwendig wäre, dass es wirtschaftlicher ist die Implementierung von Grund auf zu beginnen.
Entscheidungen was Design und Technologien betrifft, sollten vor Beginn der Implementierung getroffen worden sein. Die Aufgabe der Entwickler besteht darin das geplante Design so in Module zu unterteilen, das es programmiert werden kann[12].

2.4.1 Tests

Durch Tests wird validiert, ob die Software die Anforderungen erfüllt. Dabei sollten alle Ergebnisse von Tests dokumentiert werden, um Fehlerquellen, Lösungen und die Kosten für das Beheben der Fehler im Überblick zu behalten[12].
Diese bestehen aus mehreren Stufen.

Modultests

Programmierte Module werden isoliert getestet, um zu prüfen, ob das Modul die Aufgaben erfüllt. Dazu kann auch ein simulierter Input verwenden, wenn benötigte

Module noch nicht verfügbar sind.
Diese Tests können in zwei Arten unterteilt werden. Bei 'white-box' Tests wird das Modul in einzelnen Schritten getestet. Der Tester weiß wie die Daten verarbeitet werden. Bei 'black-box' Tests versucht der Tester das Vorgehen eines Nutzers nachzustellen. Dabei ist dem Tester die genaue Verarbeitung egal[12].

Low-Level Integrationstests

In diesem Stadium werden die Aufrufe und Rückgabewerte von anderen Modulen getestet. Dabei spielt es keine Rolle, von wem die abhängigen Module entwickelt wurden. Stehen keine realen Daten zur Verfügung kann mit simulierten Daten gearbeitet werden.[12].
Ob diese Tests manuell oder automatisiert stattfinden hängt von unterschiedlichen Faktoren ab.

Mit diesem Vorgehen sind alle Module und ihre Abhängigkeiten auf ihre Funktionalität überprüft. Auf dieser Basis wird eine Version erstellt und getestet die in dieser Form praxistauglich ist. Parallel wird daran gearbeitet die benötigten Umgebungsbedingungen bereitzustellen. Dieses Setup wird auf Funktionalität geprüft. Bei erfolgreichem Verlauf der Tests kann die Dokumentation für den Kunden, siehe 2.2, fertiggestellt werden. Bestätigt der Kunde das Resultat kann mit dem Training betreuender Personen und Benutzern begonnen werden[12].

2.4.2 Test Automatisierung

Das manuelle Ausführen von Tests kann zeitaufwändig sein, wodurch man auf die Idee kommen kann diese Tests zu automatisieren. Diese Entscheidung ist dabei von unterschiedlichen Faktoren abhängig.
Die Erfahrung des Verantwortlichen beeinflusst die Dauer für die Automatisierung maßgeblich. Auch die Umgebung, in der die Anwendung funktionieren soll, muss automatisiert simuliert werden. Basiert die Funktionalität der Software auf Benutzereingaben, oder läuft sie nach dem Start alleine? Anwendungen die selbstständig ablaufen, sind meistens einfacher zu testen, als solche bei denen man eine automatisierte Benutzerinteraktion nachstellen muss[13].
Diese und noch weitere Faktoren beeinflussen den Aufwand für die Automatisierung von Tests. Es muss also eine Kosten-Nutzen-Analyse durchgeführt werden um entscheiden zu können, ob die Automatisierung einen Mehrwert bietet.

2.5 Objektorientierung

In der Objektorientierung können Objekte als grundlegende Elemente betrachtet werden. Objekte haben Datenattribute und Verarbeitungsattribute. Die Daten werden in Variablen verschiedener Datentypen gespeichert. Verarbeitungsattribute sind die Methoden eines Objekts. Auf 'Anfragen' reagiert das Objekt mit der Ausführung von Methoden. Durch die Zusammenfassung mehrerer Datentypen können Objekte als neue Datentypen betrachtet werden.
Objektorientierte Programmierung ist ein Softwaredesign bei man dem Systeme auf die Objekte reduzieren kann, die vom System verändert werden, oder aufeinander einwirken. Jedes Objekt ist eigenständig. Diese Eigenständigkeit der Objekte führt dazu, dass die Objekte dafür verantwortlich sind die richtigen Abläufe auszuführen. In der funktionalen Programmierung ist derjenige dafür verantwortlich die passenden Methoden aufzurufen, der das Resultat der Methoden benötigt.
In der Objektorientierten Programmierung gibt es die Möglichkeit den Zugriff auf Attribute von Objekten zu regulieren. Dadurch können Variablen die nur für die interne Verarbeitung benötigt werden, vor äußerer Manipulation geschützt werden.
Die Abstraktion von Abläufen ist ein wesentlicher Bestandteil der Objektorientierung. Abstraktion bedeutet, dass für den Sender einer Anfrage nicht relevant ist, wie die Information erzeugt wird, die er benötigt. Der Sender bekommt die Information für die er eine Anfrage gestellt hat.
Eine weitere Grundlage der Objektorientierung stellt die Vererbung dar. Durch Vererbung können Klassen Attribute von anderen Klassen übernehmen und bei Bedarf anpassen. Änderungen in 'Elternklassen' werden ohne weiteren Aufwand von den Unterklassen übernommen. Änderungen in Unterklassen führen nicht zu Änderungen in den 'Elternklassen'.
Die Objektorientierung ist eine Modularisierung von Systemen. Objekte, die gleichzeitig Module sind, werden durch das Aussenden von Anfragen so miteinander verknüpft, dass sie das gewünschte Gesamtsystem darstellen. Auf Grund der Eigenständigkeit der Objekte lassen sich Anpassungen durchführen, ohne die Notwendigkeit zu verursachen, die gesamten Abläufe anpassen zu müssen.
Die Objektorientierung kann die reale Welt näher abbilden, als die prozedurale Programmierung[14].

2.6 Clean-Code

Software sollte verständlich sein. Arbeitet man alleine an einem Projekt, scheint die Bedeutung von lesbarem Code gering zu sein. Möchte man mit anderen Personen über die Software sprechen, kann die Lesbarkeit des Codes an Bedeutung gewinnen. Eine Ursache

dafür liegt in der Lebensdauer von Code. Auch Jahre nach dem der Code zum letzten Mal verändert wurde, kann es passieren, dass Informationen über Details der Implementierung benötigt werden. In dieser Situation ist es hilfreich, wenn der Code verständlich geschrieben ist, da Gedanken die während der Entwicklung berücksichtigt wurden, nicht immer dokumentiert sind[15].
"Indeed, the ratio of time spent reading versus writing is well over 10 to 1. We are constantly reading old code as part of the effort to write new code. ...[Therefore,] making it easy to read makes it easier to write."[16]
Code kann aus vielen Gründen Probleme in der Einarbeitung verursachen. Die Namensgebung von Klassen, Methoden und Variablen spielt dabei eine große Rolle. Lassen sich auf Grund von Abkürzungen oder anderen Bezeichnungen keine eindeutigen Rückschlüsse auf die Funktion ziehen, ist man auf eine Dokumentation der Software angewiesen. Die Dokumentation des Codes ist aber wie in 2.2 beschrieben, aus mehreren Gründen häufig mangelhaft. Das Wissen muss also neu aufgebaut werden, wodurch Fehler im Verständnis entstehen können. Weitere Schwierigkeiten können auskommentierte Teile des Codes darstellen, da der Grund für die weitere Existenz dieses 'toten' Codes häufig ebenfalls nicht dokumentiert ist und durch Tools für die Versionsverwaltung nicht notwendig ist.
Es gibt mehrere Prinzipien, nach denen sich 'Clean-Code' erreichen lässt.

2.6.1 SOLID-Prinzipien

Die SOLID-Prinzipien bestehen aus fünf einzelnen Prinzipien, deren Anfangsbuchstaben zusammengesetzt SOLID ergeben[17].

Single-Responsibility Principle

Eine Klasse sollte nur aus einem Grund angepasst werden. Daraus lässt sich ableiten, dass eine Klasse nur für eine Funktion verantwortlich sein sollte. Der Schluss jede Klasse sollte nur eine Methode haben, ist nicht korrekt, da man für das Erfüllen einer Verantwortlichkeit auch mehr als eine Funktion notwendig sein kann[17].

Open/Closed Principle

Klassen sollten offen für Erweiterungen sein, jedoch geschlossen gegenüber Veränderungen. Die Idee dahinter ist, dass Veränderungen an einem bereits genutzten Objekt Fehler im Ablauf verursachen können.
Gibt es neue Anforderungen sollte die bestehende Klasse durch Vererbung um neue Funktionen erweitert werden, anstatt die bestehenden Klasse anzupassen. Die Erweiterungen sollten keine Änderungen in der Basisklasse verursachen[17].

Liskov Substitution Principle

Dieses Prinzip besagt, dass jede Klasse durch eine von ihr abgeleitete Klasse ersetzt

werden kann[17].
Daraus folgt, dass jede abgeleitete Klasse die gleichen Attribute der ‘Elternklasse’ implementieren muss. Es können weitere Attribute ergänzt werden. Attribute die von ‘Elternklassen’ stammen, sollten nicht verändert werden, da dies das erwartete Verhalten dieser Attribute nicht erfüllen würde.

Interface Segregation Principle

Nach diesem Prinzip sollten Interfaces nur die Funktionen bereitstellen, die auch tatsächlich notwendig sind[17]. Werden von einem Interface eine Basisklasse und eine spezialisierte Klasse abgeleitet, benötigt die spezialisierte mehr Attribute. Die zusätzlichen Attribute für die Spezialisierung werden für die Funktionalität der Basisklasse nicht benötigt. Diese Attribute sollten nicht über dasselbe Interface bereitgestellt werden, da die Basisklasse Attribute implementieren müsste, die keine Anwendung finden würden. Für die spezialisierte Klasse kann ein spezielles Interface entwickelt werden, oder die benötigten Zusatzattribute können direkt in der Klasse implementiert werden.

Dependency Inversion Principle

High-Level Klassen wie Benutzeroberflächen sollten nicht direkt von einer Low-Level Klasse, wie einer Datenbankmanagement-Klasse, abhängig sein. Stattdessen sollten beide Klassen von einem Interface abhängig sein, wodurch eine höherer Flexibilität entsteht, um beispielsweise den Datenbanktyp anzupassen[17].

2.6.2 DRY-Prinzipien

Ausgeschrieben bedeutet die Abkürzung DRY ‘Don’t Repeat Yourself’[17]. Mehrfaches Verwenden von Code erhöht den Aufwand in der Instandhaltung und bei Anpassungen des Codes, da Anpassungen an jeder Stelle vorgenommen werden müssen[18].
Um Mehrfachverwendung von Code zu verhindern können Konstanten, Funktionen und Modularisierung verwendet werden um Abläufe und Werte zu definieren. Diese Definitionen können dann an anderen Stellen aufgerufen werden[18]. Anpassungen können dann innerhalb der Definition vorgenommen werden und sind dann bei allen Aufrufen der Definition wirksam.

2.7 Versionsverwaltung

Um eine unübersichtliche Ordnerstruktur mit Softwareständen zu vermeiden, gibt es Tools für die Versionsverwaltung. Mit diesen Tools lassen sich Inhalte die relevant für das Projekt sind speichern und deren zeitliche Entwicklung nachvollziehen. Dateien können mit diesen Tools auf frühere Stände zurücksetzen. Diese Funktion hilft dabei funktionierende

Versionen der Software wiederherzustellen.
Tools für die Versionskontrolle sind nicht nur zur Datensicherung hilfreich. Sie unterstützen die Möglichkeit die Zwischenstände auf einem Server in einem Repository zu speichern. An einem Repository können mehrere Entwickler gleichzeitig arbeiten. Um bei einem Serverproblem nicht das gesamte Repository zu verlieren, arbeiten Tools wie Git dezentral. Lädt ein Benutzer einen Stand vom Server, lädt er nicht die aktuellen Versionen der Dateien herunter, sondern das ganze Repository. Somit gibt es auch lokale Backups des Repositories[19].
Durch das Verwenden eines Tools für Versionsverwaltung können auch 'tote' Codeteile wie in 2.6 erwähnt entfernt werden, ohne zu befürchten diese Informationen verloren zu haben. Der Einsatz dieser Tools unterstützt also auch bei der Anwendung von 'Clean-Code' Prinzipien.

2.8 Continous Integration

Continous Integration ist ein Vorgehen in der Softwareentwicklung, bei dem Team-Member mindestens einmal in einem definierten Zeitraum ihre Arbeit durch eine automatisierte Routine testen, ob die Änderungen funktionieren. Mit diesem Vorgehen lassen sich Integrationsprobleme reduzieren[20].

3 Parametertabellenkonfigurator

Mit Hilfe des „Parametertabellenkonfigurators" sollen die Installationsabläufe von Programmen und Produkten von Blum auf CNC-Maschinen unterstützt werden.

3.1 Problemstellung

Bei bestehenden Programmen für numerische Steuerungen müssen während der Inbetriebnahme Parametertabellen in einem beliebigen Texteditor angepasst werden. Aufgrund der Anzahl, wie auch der Einstellmöglichkeiten der Parameter, zu sehen in Abbildung 1, wird hierfür eine separate Installationsanleitung benötigt. Der Prozess der Parametrierung ist daher Fehleranfällig und von der Erfahrung des Inbetriebnehmers abhängig. Um den hohen Anforderungen der Kunden gerecht zu werden, gilt es potenzielle Fehlerquellen zu eliminieren. Hierfür soll selbstständig ein „Parametertabellenkonfigurator" entwickelt werden.

```
#680=0(LANGUAGE 0=ENGLISH,1=GERMAN,2=FRENCH,3=ITALIAN,4=SPANISH,5=DUTCH,)
(...6=SWEDISH,7=POLISH,8=DANISH,9=CZECH,10=PORTUGUESE,USE 19 FOR OEM)
#641=4
#694=0 (BASE ADDRESS FOR 10 MACRO VARIABLES FOR OPTION -AH)
#672=0 (BASE ADDRESS FOR 5 MACRO VARIABLES FOR CLIENT SERVER HEADER)
(#770=0) (BASE ADDRESS FOR LC-VISION)
#679=0 (BASE ADDRESS FOR 20 MACRO VARIABLES FOR LC50 COMMUNICATION)
#640=0 (BASE ADDRESS FOR 10 MACRO VARIABLES FOR CALIBRATION VALUES)
(FOR AUTO SETTING OF #601,#620,#621,#650,#651,#652,#653)
(ENTER HERE LINE NUMBERS OF AXIS SETTING PARAMETER 1006,1020)
#10=2(LINE OF RADIUS AXIS Y IS 2ND LINE)
#11=1(LINE OF LASER AXIS X IS 1ST LINE)
#12=3(LINE OF LENGTH AXIS Z IS 3RD LINE)
#601=#0 (#0=AUTO SETTING IN BLOCK N100)
#602=+1.0101
#603=+2.0204
#663=0
#620=#0 (#0=AUTO SETTING IN BLOCK N100)
#621=#0 (#0=AUTO SETTING IN BLOCK N100)
```

Abbildung 1: Auszug einer Parametertabelle

3.2 Anforderungen

An den „Parametertabellenkonfigurator“ gibt es mehrere Anforderungen.
Parametertabellen sollen eingelesen und angezeigt werden können. Zu den Parametern soll es ermöglicht werden Beschreibungen aus Text und Bild speichern und anzeigen zu können.
Diese Anforderungen sind aus dem Bereich der funktionalen Anforderungen (2.1.4)
Nicht funktionale Anforderungen (2.1.4) sind die Verwendung der Programmiersprache und das Verwenden des in C++ entwickelten Framework Qt.
Aus diesen Anforderungen lässt sich ableiten welche GUI-Elemente nötig sind und welche Funktionalitäten benötigt werden.
Um die Anforderungen an den „Parametertabellenkonfigurator“ zu sammeln, die nicht im Auftragsdokument stehen, wurde das Interview (2.1.1) gewählt. Die Entscheidung wurde getroffen, weil in offenen Gesprächen mehr Zusatzinformationen fließen, als in Umfragen (2.1.2) oder durch Beobachtung (2.1.3). Zusätzlich kann auf Verständnisschwierigkeiten reagiert werden, solange das Gespräch noch aktuell ist und der Kontext noch nicht vergessen ist.
Hauptergebnis ist die Erstellung einer übersichtlichen GUI.
Die Anforderungen zielen darauf ab ein Minimum Viable Product (MVP) zu erstellen. Ein MVP erfüllt die wichtigsten Anforderungen an das Produkt, ist aber noch nicht so weit entwickelt, dass es alle für den Einsatz notwendigen Anforderungen erfüllt.

3.2.1 Notwendige Funktionalitäten

Um die Parametertabellen anzeigen zu können müssen diese aus einer Datei eingelesen werden. Um Konfigurationen festzuhalten wird eine Funktion zum Speichern der Tabellen benötigt.
Für die Beschreibungen wird zusätzlich eine Funktion benötigt um diese mit den richtigen Parametern zu verknüpfen.

3.2.2 Notwendige GUI-Elemente

Die GUI benötigt Anzeigemöglichkeiten für die Parametertabellen und Beschreibungen, Angaben von Namen, Steuerung und Produkttyp zur aktuellen Parametertabelle und den aktuellen Parameter.
Benötigte Steuerelemente sind Möglichkeiten um Parametertabellen und Beschreibungen zu laden und zu speichern.

3.3 GUI-Entwurf

Um die GUI zu entwerfen, kommt ein Prototyp (2.3.2) zum Einsatz, der mit dem Mockup-Tool "Balsamiq Mockup 3“ erstellt wurde. Das Tool bietet nahezu alle gängigen GUI-Elemente an und kann diese miteinander verknüpfen, so dass schon in einem frühen Stadium Abläufe nachvollzogen und besprochen werden können. Zusätzlich bietet das Tool eine unkomplizierte Möglichkeit alternative Entwürfe zu entwerfen und in den Ablauf der Simulation zu integrieren. Das unterstützt den Prozess während der Entwicklung der GUI, da neue Vorschläge in kurzer Zeit besprochen und weiterentwickelt werden können.

Die GUI ist aus mehreren einzelnen Fenstern aufgebaut. Dieser Aufbau unterstützt die Übersichtlichkeit und der Benutzerführung bei der Verwendung, da jeweils nur die benötigten Informationen und Funktionen angezeigt werden.
Insgesamt besteht die GUI aus vier Fenstern, die dem Benutzer die benötigten Funktionen zur Verfügung stellen.
Das in Abbildung 2 dargestellte Fenster dient dem Zweck bei erster Benutzung den Pfad zu einer Datenbank, in der die Beschreibungen gespeichert sind, einzugeben. Im nächsten Fenster, zu sehen in Abbildung 3, lässt sich die Auswahl der Parametertabellen konfigurieren. Nach der Auswahl einer Parametertabelle wird diese im Hauptfenster, siehe Abbildung 4, angezeigt. In diesem Fenster werden die Beschreibungen der Parameter angezeigt und die Parametertabelle kann angepasst werden. Es gibt Steuerelemente für die Navigation durch die Parametertabelle, für die Navigation durch die Fenster und die Möglichkeit die geänderte Parametertabelle zu speichern.

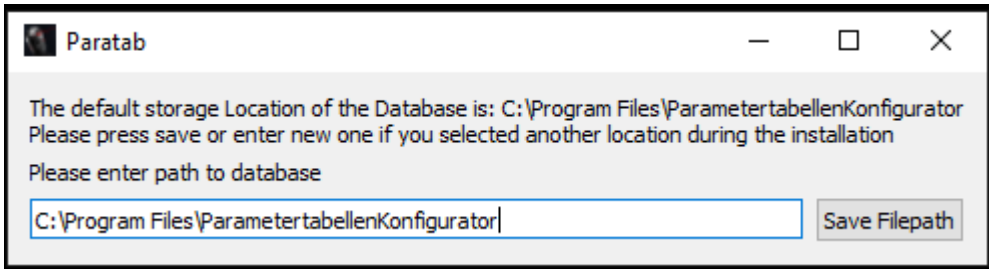

Abbildung 2: Fenster um den Dateipfad der Datenbank einzugeben, es gibt einen Standardpfad

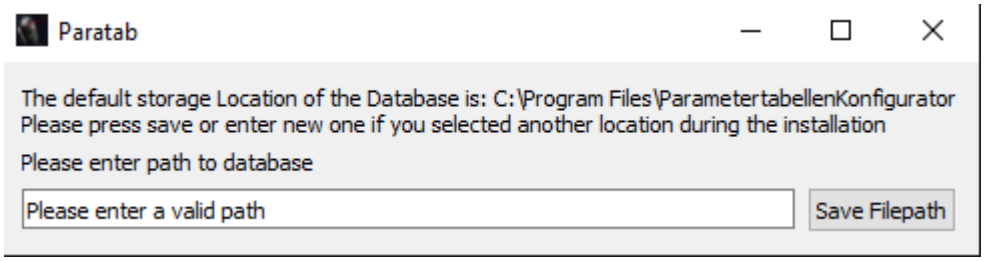

Abbildung 3: Wurde ein nicht vorhandener Pfad eingegeben erscheint die Fehlermeldung "Please enter a valid path"

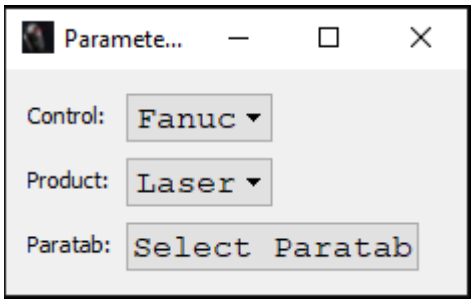

Abbildung 4: Fenster um Auswahl der Parametertabellen zu konfigurieren

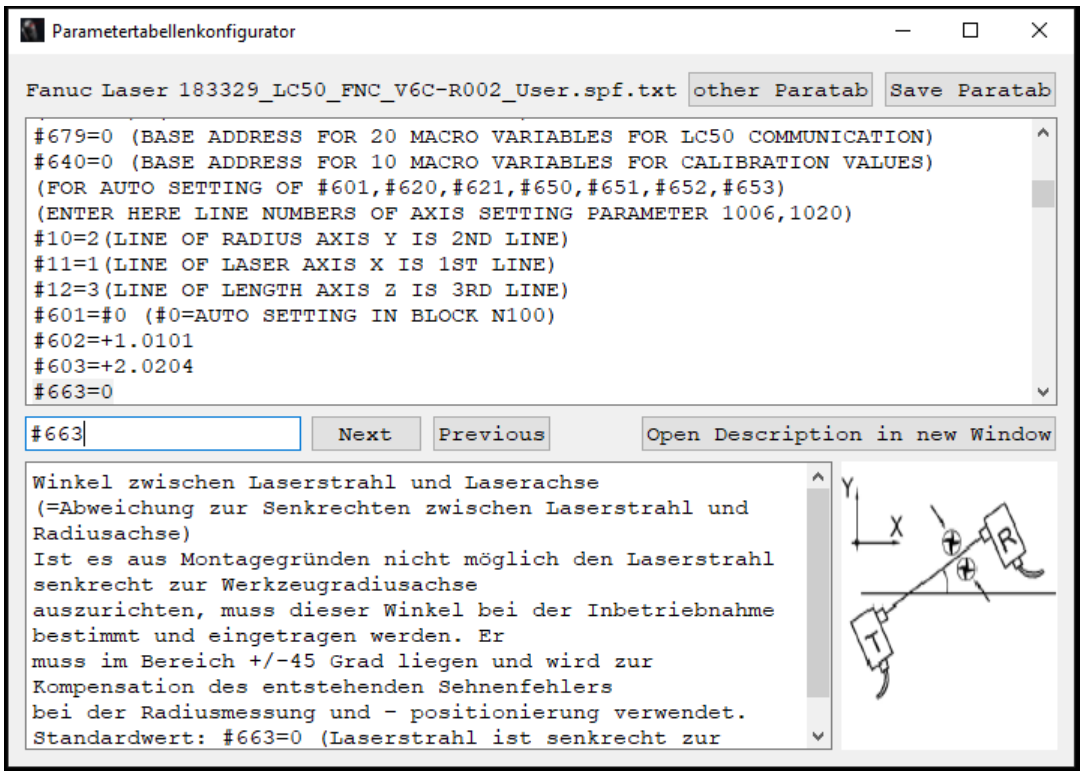

Abbildung 5: Fenster für die Anzeige der Parametertabellen und deren Beschreibungen

In Abbildung 5 werden im oberen Bereich Informationen zur aktuell ausgewählten Parametertabelle angezeigt. Oben rechts gibt es Steuerelemente um eine andere Parametertabelle auszuwählen, oder um die Aktuelle zu speichern.
Unterhalb der Anzeige für die Parametertabelle ist der Bereich für detaillierte Informationen zum aktuellen Parameter. Es gibt Steuerelemente um andere Parameter auszuwählen. Dies ist mit den Buttons „Next“ und „Previous“ möglich, die jeweils um einen Parameter nach oben oder nach unten springen. Durch direkte Eingabe eines Parameters in das Parameterfeld kann zu einem beliebigen Parameter gesprungen werden. Um größere Beschreibungen übersichtlich sehen zu können gibt es die Möglichkeit die Beschreibung isoliert in einem neuen Fenster zu öffnen.

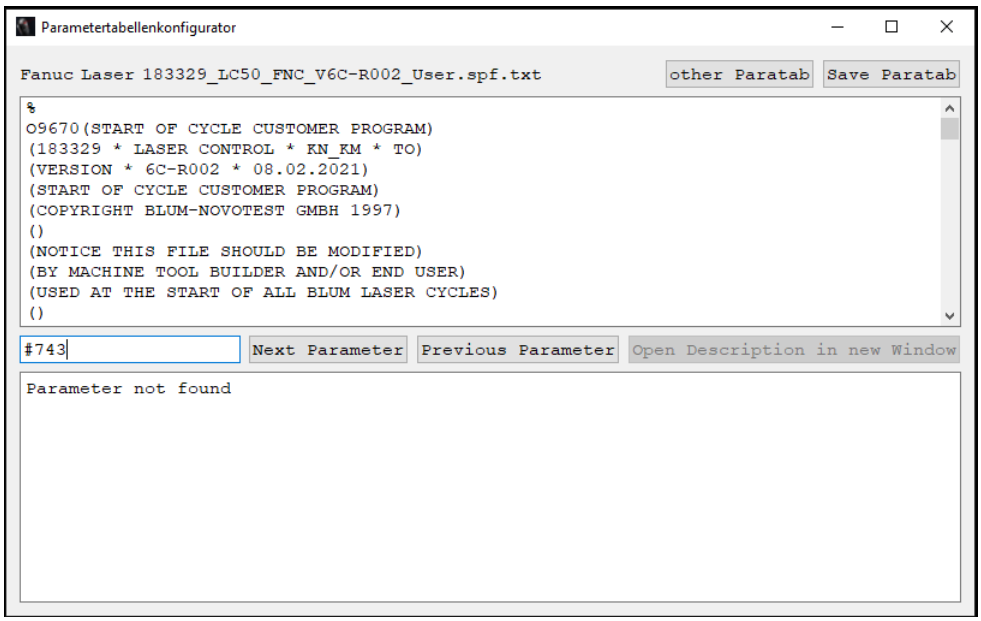

Abbildung 6: Wird ein nicht vorhandener Parameter in das Feld eingegeben wird eine Fehlermeldung angezeigt.

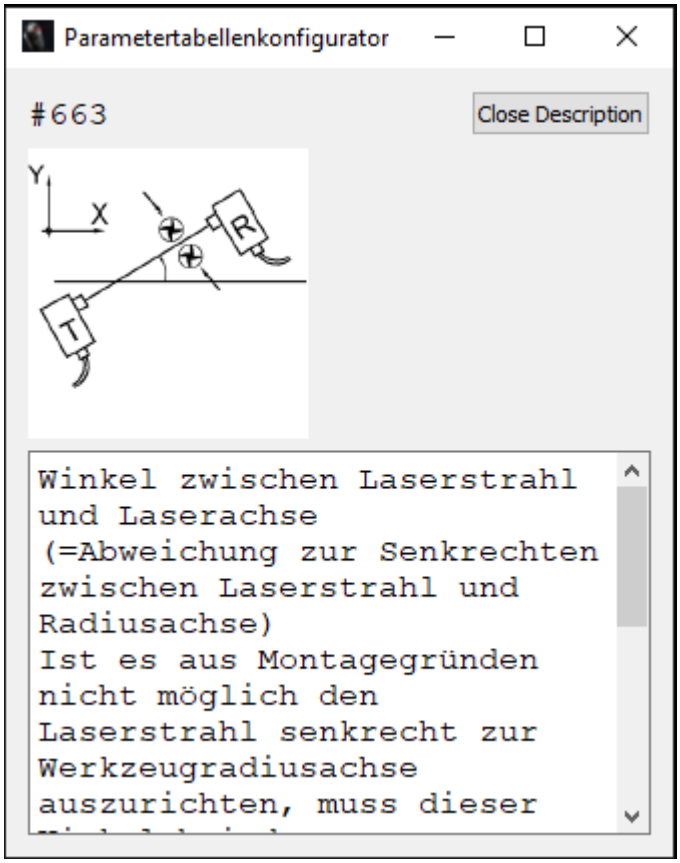

Abbildung 7: Isolierte Ansicht einer Beschreibung eines Parameters

Das in Abbildung 7 sichtbare Bild der Beschreibung hat eine konstante Größe. Der Grund dafür ist, dass ansonsten bei Änderung der Fenstergröße das Bild verzerrt werden würde

und das Erkennen von wichtigen Informationen im Bild erschwert werden würde.

3.4 Programmierung

Für die Programmierung wird wie in den nicht funktionalen Anforderungen (3.2) vorgegeben C++ mit dem für GUI optimierten Framework Qt verwendet.
Da Qt für den Aufbau von Benutzeroberflächen Klassen bereitstellt, wird die Software objektorientiert (2.5) programmiert. Um den Code auch mit wachsender Größe lesbar und verständlich zu halten kommen 'Clean-Code Prinzipien' (2.6) zum Einsatz.

3.4.1 Softwarestruktur

Um die Software flexibel zu gestalten, wird diese mit Hilfe von modular programmiert. Die Modularität wird durch die Projektorientierte Programmierung (2.5) unterstützt. Die Logik und die GUI bestehen aus mehreren einzelnen Klassen, die als Module betrachtet werden können. Um die einzelnen Module zu verbinden, gibt es eine 'Managerklasse' für den Logikteil und eine für die Module der GUI. Um diese beiden 'Managerklassen' ebenfalls unabhängig zu halten, gibt es eine weitere 'Managerklasse' die dafür verantwortlich ist, die Daten zwischen Logik und GUI zu transferieren.
Diese Trennung, erkennbar in Abbildung 8, ermöglicht es Anpassungen an den einzelnen Modulen vorzunehmen, ohne die höheren Ebenen grundlegend anzupassen.

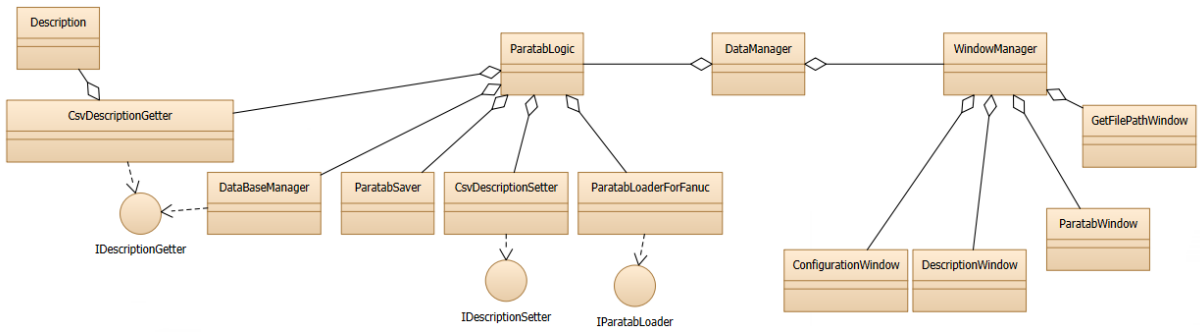

Abbildung 8: UML-Diagramm der Software ohne attribute der Klassen

Das Diagramm in Abbildung 8 dient dazu die Modularität der Software zu veranschaulichen, daher wurde auf die Darstellung von Inhalten der Klassen verzichtet.
UML dient dazu die Architektur von Software zu visualisieren. Klassen werden durch Rechtecke visualisiert, Interfaces durch Kreise.

3.4.2 Verarbeitung von Benutzeraktionen

Interagiert der Benutzer mit der GUI werden entsprechende Veränderungen erwartet.
Qt bietet für die Verarbeitung der Benutzeraktionen mit dem 'Signal-Slot Mechanismus' Unterstützung. Dieser Mechanismus bietet die Möglichkeit Signale auszusenden, die von einem verbundenen Slot empfangen werden können. Dabei muss das Senderobjekt keine Verbindung zum Empfängerobjekt haben. Die Verbindung zwischen Signal kann entweder direkt im Empfängerobjekt, oder in einer Instanz einer 'Managerklasse' eingerichtet werden. Grundvoraussetzung ist, dass das Objekt, in dem die Verbindung eingerichtet ist, sowohl eine Verbindung mit dem Sender- als auch mit dem Empfängerobjekt hat.
Die meisten Klassen die von Qt bereitgestellt werden, haben vordefinierte Signale. Diese erlauben es die Benutzerinteraktionen zu verarbeiten und entsprechend darauf zu reagieren.
Neben den vordefinierten Signalen von GUI-Elementen können auch selbst definierte Signale eingesetzt werden, um Methoden aufzurufen. Signale von GUI-Elementen werden ohne weitere Implementierungen durch den Entwickler gesendet. Selbst definierte Signale müssen mit einem Befehl gesendet werden. Signale können auch Daten übergeben, die vom verbundenen Slot verarbeitet werden können. Dabei muss die Zahl der Argumente des Slots nicht mit der des Signals übereinstimmen. Signale können mit allen Slots verbunden werden, die gleich viele oder weniger Argumente entgegennehmen. Begrenzend wirkt hier die Reihenfolge der Datentypen der Argumente. Diese muss übereinstimmen.
Dieser Mechanismus ermöglicht weitere Flexibilität, da jederzeit neue Verbindungen erstellt werden können, oder bestehende entfernt werden können. Diese Funktionalität ist hilfreich, wenn das Ausführen einer Methode Signale aussenden würde, die unerwünschte Folgen hätten. In diesem Fall kann man die Verbindung von Signal und Slot zu Beginn der Methode trennen und am Ende der Methode wieder herstellen.

3.4.3 Speicherung der Beschreibungen

Die Beschreibungen der Parameter werden in einer relationalen Datenbank gespeichert. Relationale Datenbanken speichern Daten in Tabellen, in denen eine Spalte als eindeutiger Schlüssel , 'Primary Key' für die Reihen dient. Relationen zwischen Elementen werden durch eine zusätzliche Spalte symbolisiert. In dieser Spalte wird der 'Primary Key' der Reihe eingetragen, zu der eine Verbindung besteht. Es ist nicht relevant, ob die verbundene Reihe in der selben oder einer anderen Tabelle steht. In relationalen Datenbanken werden werden die Daten nicht über die Position, sondern über den 'Primary Key' direkt adressiert. Die genaue Position der Objekte in der Datenbank ist nicht bekannt[21].
Der Grund für die Wahl dieses Models wurde getroffen, da Beschreibungen und Bilder

eindeutig einem Parameter zugeordnet werden können. Da der Parameter, für den die Beschreibung geladen werden soll, bekannt ist, ist der fehlende Zugriff über die Position kein Hindernis.

3.4.4 Tests

Die Software wird abgesehen von der GitLab Pipeline (3.4.5) manuell getestet. Dies hat den Grund, dass die Automatisierung von Tests anspruchsvoll und zeitaufwendig ist (2.4.1).

3.4.5 Verwendete Tools

Um die Entwicklung zu unterstützen werden Tools für die Versionsverwaltung und 'Continous Developement/ Integration' (2.8) eingesetzt.
Für die Versionsverwaltung kommen GitKraken und GitLab zum Einsatz. GitKraken dient als Schnittstelle zwischen dem lokalen Projekt auf dem Computer und der Plattform für 'Continous Developement/ Integration' 2.8 GitLab. In GitKraken lässt sich der gesamte Verlauf des Projekts einsehen. Außerdem können alte Commits 2.7 ausgewählt werden und direkt in einer Entwicklungsumgebung geöffnet werden. Für die Sicherung auf dem Server bietet GitKraken die Git Befehle, die durch eine graphische Benutzeroberfläche oder durch das integrierte Terminal genutzt werden können.
GitLab wird nicht nur für die Sicherung von Softwareständen auf einem Server eingesetzt. Ebenfalls genutzt werden die Möglichkeiten für das Organisieren von To-Do's durch 'Issues', sowie die GitLab Pipeline. Die GitLab Pipeline ist ein Feature im Kontext von Continous Integration 2.8. Zusätzlich werden während dem Prozess Artefakte erstellt, in denen Dateien gesammelt werden, die für das Erstellen eines Installers benötigt werden. Diese Artefakte lassen sich nach dem erfolgreichen Durchlauf der Pipeline als ZIP-Ordner herunterladen.

3.5 Anwendung der Software

Vor der Bearbeitung wird im ConfigurationWindow, Abbildung 3, die Steuerung und das Produkt festgelegt. Durch klicken auf „Select Paratab" wird ein Dateiexplorer-Fenster geöffnet. Durch die getroffene Auswahl im ConfigurationWindow, Abbildung 3, werden die angezeigten Dateien im Dateiexplorer gefiltert um die Auswahl der richtigen Parametertabelle zu vereinfachen. Die ausgewählte Parametertabelle wird im ParatabWindow, Abbildung 4 und 5, angezeigt.
Um einen Parameter zu bearbeiten, muss die entsprechende Zeile im Textfeld ausgewählt zu werden. Dazu gibt es unterschiedliche Möglichkeiten. Man kann eine Zeile direkt durch

anklicken mit der Maus auswählen. Möchte der Benutzer jeden Parameter nacheinander konfigurieren bieten die Buttons ‘Next’ und ‘Previous’ die Möglichkeit direkt zum nächsten, beziehungsweise zum vorherigen Parameter zu springen, ohne danach zu suchen. Geht es darum bestimmte Parameter an verschiedenen Stellen zu bearbeiten, kann im Feld für die Anzeige des aktuellen Parameters ein Parameter eingegeben werden. Ist der eingegebene Parameter in der Tabelle enthalten, springt die Anzeige direkt zu diesem Parameter.
Um zu prüfen, ob eine Zeile einen Parameter enthält kommen ‘Reguläre Ausdrücke’ zum Einsatz. Mit Diesen lassen sich Texte auf bestimmte Inhalte oder Muster untersuchen.
In dieser Software wird mit Hilfe der Cursor-Klasse von Qt die gesamte Zeile ausgewählt und mit einem ‘Regulären Ausdruck’ untersucht.
Um die Bearbeitung der Parameter zu unterstützen wird unterhalb der Anzeige für die Parametertabelle die Beschreibung des Parameters angezeigt. Die Option „Open Description in new Window“, Abbildung 4 und 5, hilft bei langen Beschreibungstexten und Bildern das Lesen der Beschreibung zu vereinfachen. Es öffnet sich ein neues Fenster, zu sehen in Abbildung 6, in dem die Beschreibung und das Bild isoliert angezeigt werden.
Ist die Konfiguration der Parametertabelle abgeschlossen, kann in der oberen rechten Ecke, zu sehen in Abbildung 4 und 5, die Parametertabelle gespeichert werden.

4 LC-VISION

„LC-VISION“ ist eine Software von Blum für die Visualisierung und Auswertung von Messdaten der „DIGILOG-Lasermesssysteme“. Bei der Entwicklung der Software treten Unstimmigkeiten auf, die die Funktionalität der Software nicht beeinträchtigen, aber dennoch behoben werden sollten. Während der Behebung von Unstimmigkeiten kann man von der Erfahrung anderer Entwickler profitieren. Da die Lösung in den bestehenden Code integriert wird, kommt man mit den Techniken anderer Entwickler in Kontakt. Die eigene Arbeit kann reflektiert werden[22], wenn darüber gesprochen wird. Außerdem fördert die aktive Mitarbeit nicht nur Kenntnisse über die Theorie der Softwareentwicklung sondern auch über die praktischen Aspekte[22].
Ein weiterer Vorteil ist das der Lernprozess in einer relevanten Umgebung stattfindet[22]. Hat eine Aufgabe Relevanz kann dies einen positiven Einfluss auf die Motivation haben. Bevor mit der Behebung von Unstimmigkeiten angefangen werden konnte, sollte man den Code und die Benutzeroberfläche kennenzulernen.
Um die Benutzeroberfläche kennenzulernen kann eine Kombination aus statischer und dynamischer Analyse verwendet werden. Bei der statischen Analyse wird nur der Code betrachtet. Um hieraus qualitative Informationen zu gewinnen, wird ein großer Teil des Codes benötigt, da nur so Zusammenhänge und Abläufe erkannt werden können[23]. Der

Vorteil liegt darin, das dieses Verfahren auf jedem System durchgeführt werden kann, das einen Texteditor bietet.
In der dynamischen Analyse werden Informationen während der Laufzeit der Anwendung gesammelt. Die dynamische Analyse bringt häufig die besseren Ergebnisse, da viele Abhängigkeiten, zum Beispiel „dynamic link libraries“, erst zur Laufzeit eingebunden werden. Mit diesen Analyseverfahren kann der Code optimiert werden und das Verständnis des Codes verbessert werden[23].
Für die Einarbeitung in die Software „LC-VISION“ liegt der Fokus auf der Verbesserung für das Verständnis der Abläufe. Hierzu kommt eine Kombination von statischer und dynamischer Analyse zum Einsatz. Um die obersten Schichten der Software ausfindig zu machen, wurde die dynamische Analyse genutzt. In der gestarteten Anwendung wurde nach Elementen gesucht, die im Code leicht wiederzufinden sind. Solche Elemente können Texte, Bilder oder Menüabläufe sein. Werden Muster aus der GUI im Code gefunden kann man von dort aus Abläufen in tiefere Schichten der Software folgen. Dieser Schritt ist Teil der zuvor erwähnten statischen Analyse. Treten hierbei Probleme oder Fragen auf, kann es hilfreich sein mit einen Entwickler der Software zu sprechen, ob dieser Unterstützung bieten kann.
Das erworbene Wissen kann durch die Behebung von Unstimmigkeiten, wie der Anpassung von Texten und Übersetzungen, sowie inkonsistenter Datenanzeige in Sonderfällen, gefestigt werden.
Danach kann an Erweiterungen oder der Umstrukturierung der Software gearbeitet werden. In diesem Fall bestehen die Aufgaben aus Anpassungen und Erweiterungen der Benutzeroberfläche. Ist die Signalstärke des Lasers zu schwach, soll ein Handlungsvorschlag angezeigt werden, mit dem die Signalstärke optimiert werden kann. Eine zweite Anpassung ist eine Fehlerbehebung im Bereich der Lizenzanzeige. Informationen zu selten genutzten Lizenzen sollten nur angezeigt werden, wenn diese gekauft sind.

Literatur

[1] A. F. Blackwell, "What is programming?," p. 20, 2002.

[2] S. Lane, P. O'Raghallaigh, and D. Sammon, "Requirements gathering: the journey," *Journal of Decision Systems*, vol. 25, no. sup1, pp. 302–312, 2016.

[3] S. Tiwari, S. S. Rathore, and A. Gupta, "Selecting requirement elicitation techniques for software projects," pp. 1–10, 2012.

[4] R. Silhavy, P. Silhavy, and Z. Prokopova, "Requirements gathering methods in system engineering," *Recent Researches in Automatic Control*, pp. 105–110, 2011.

[5] J. Eckhardt, A. Vogelsang, and D. M. Fernández, "Are "non-functional"requirements really non-functional? an investigation of non-functional requirements in practice," p. 832–842, 2016.

[6] T. Theunissen, U. van Heesch, and P. Avgeriou, "A mapping study on documentation in continuous software development," *Information and Software Technology*, vol. 142, p. 106733, 2022.

[7] S. Nerur, R. Mahapatra, and G. Mangalaraj, "Challenges of migrating to agile methodologies," *Commun. ACM*, vol. 48, p. 72–78, may 2005.

[8] R. Budde, K. Kautz, K. Kuhlenkamp, and H. Züllighoven, "What is prototyping?," *Information Technology & People*, 1992.

[9] J. C. Bastien, "Usability testing: a review of some methodological and technical aspects of the method," *International Journal of Medical Informatics*, vol. 79, no. 4, pp. e18–e23, 2010. Human Factors Engineering for Healthcare Applications Special Issue.

[10] P. Šimek, J. Vaněk, and P. Pavlík, "Usability of ux methods in agrarian sector-verification," *Agris on-line Papers in Economics and Informatics*, vol. 7, no. 665-2016-45086, pp. 49–56, 2015.

[11] J. Matejka, T. Grossman, and G. Fitzmaurice, "Patina: Dynamic heatmaps for visualizing application usage," p. 3227–3236, 2013.

[12] J. J. Rakos, "Software project management," 1990.

[13] M. Fewster and D. Graham, "Software test automation," 1999.

[14] B. P. Pokkunuri, "Object oriented programming," *SIGPLAN Not.*, vol. 24, p. 96–101, nov 1989.

[15] A. Filazzola and C. Lortie, "A call for clean code to effectively communicate science," *Methods in Ecology and Evolution*, vol. n/a, no. n/a.

[16] D. Hutton, "Clean code: a handbook of agile software craftsmanship," *Kybernetes*, 2009.

[17] V. Sarcar, "Use the dry principle," pp. 109–128, 2021.

[18] M. Schaffner *et al.*, "Programming for experimental economics: Introducing corala lightweight framework for experimental economic experiments," *QUT Business School: Tech. rep*, 2013.

[19] M. Denker, S. Srecec, D. A. liegt das im Literaturverzeichnis, P. Git, S. Chacon, and B. Straub, "Versionsverwaltung mit git," 2015.

[20] M. Fowler and M. Foemmel, "Continuous integration," 2006.

[21] E. F. Codd, "Relational database: A practical foundation for productivity," p. 1981, 2007.

[22] M. Johnson and M. Senges, "Learning to be a software engineer in a complex organization: A case study focusing on apprenticeship/practice based learning for getting new engineers productive in contributing to the google codebase," *Journal of Workplace Learning*, vol. 22, no. 3, pp. 180–194, 2010.

[23] M. Mock, "Dynamic analysis from the bottom up," p. 13, 2003.