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

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

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.

_____ _____

Ort Datum Unterschrift

Abkürzungsverzeichnis

Blum	Blum-Novotest GmbH
GUI	Graphische Benutzeroberfläche

Abbildungsverzeichnis

Inhaltsverzeichnis

1 Einleitung

Die Blum-Novotest GmbH (Blum) ist im Bereich Maschinenbau für Mess- und Prüftechnik tätig. Am Standort Grünkraut (Baden-Württemberg) ist der Hauptsitz der Firma. Dort befinden sich Entwicklung und Fertigung für den Bereich Messtechnik. Am zweiten Standort in Willich (Nordrhein-Westfalen) sind die Entwicklung und Fertigung im Bereich Prüftechnik untergebracht.
Produkte von Blum werden in vielen anspruchsvollen Industrien im Bereich der Qualitätssicherung eingesetzt. Die Kunden kommen unter anderem aus der Automobil-, Luftfahrt- und der Werkzeugmaschinenindustrie.
An die Produkte dieser Industrien werden höchste Ansprüche in Sachen Qualität 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.
Die Arbeit wird im Umfeld von Blum am Standort Grünkraut verfasst.
Ziel der Arbeit ist es die vielfältigen Aspekte der Softwareentwicklung kennenzulernen und den Einstieg zu schaffen.
Um dieses Ziel zu erreichen, besteht der praktische Teil aus zwei Projekten.
Mit der Entwicklung eines „Parametertabellenkonfigurators“ sollen die verschiedenen Aufgaben der Softwareentwicklung veranschaulicht werden.
Durch die Mitarbeit am schon bestehenden Produkt „LC-Vision“ wird die Produktpflege und Weiterentwicklung thematisiert. Außerdem wird dabei das Einarbeiten in unbekannten Code relevant, was ein wichtiger Bestandteil der Produktpflege ist.

2 Softwareentwicklung

Softwareentwicklung besteht nicht nur aus dem Schreiben von Software. Auch die Entwicklung von Software ist ein Projekt. Man beginnt mit dem Sammeln der Anforderungen, startet die Entwicklung von Prototypen und geht in die Implementierung. Der Prozess endet mit der Produktpflege und dem Service.

2.1 Anforderungsanalyse

Eindeutige und präzise Anforderungen bilden die Grundlage für ein funktionierendes Projekt[1].
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[2]. 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[2].
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[3].
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.
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[4].
Nicht funktionale Anforderungen sind häufig unpräzise formuliert und somit problematisch in der Validierung[4].

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 der Software und die Dokumentation für den Anwender[5].
Software-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 brin-

gen 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 potentielle 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[5].
User-Dokumentation
Der Dokumentation für die Anwender und die Betreuer 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. Da Probleme dieser Art einen direkten Einfluss auf den Erfolg eines Produktes oder der Firma haben können. Diese möglichen Konsequenzen sorgen dafür, dass dieser Dokumentation mehr Bedeutung beigemessen wird[5].

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[6].
Um den Kunden einzubeziehen können Prototypen eingesetzt werden.
Hat das Programm eine Graphische 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[7].

Dieser Prototyp kann durch Software, oder mit einem Mockup-Tool[1], oder auch in einer Software für Präsentationen, realisiert werden. Ein 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 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. Hat man 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 werden die tatsächlichen Probleme der Testpersonen besser erkenntlich.[8].
Mit dieser Testmethodik lassen sich nur Prototypen 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[9].
30-Sekunden Tests lassen sich mit Bildentwürfen 2.3.1 und mit Prototypen 2.3.2 durchführen, da es bei beiden Entwürfen möglich ist ein Element für 30 Sekunden zu zeigen.

2.3.5 Heatmaps

Heatmaptools markieren farbig welche Bereiche einer Benutzeroberfläche oft genutzt werden und welche seltene[10]. Diese Daten geben Informationen darüber ob Kernfunktionen anders positioniert werden sollten.
Diese Methode lässt sich für Bildentwürfe 2.3.1 und Prototypen 2.3.2 verwenden.

2.4 Implementierung

In der Implementierung ist der Aufwand abhängig von der Vorgehensweise. Wird die GUI mit Hilfe von Bildentwürfen 2.3.1 entwickelt, ist der Aufwand der Implementierung höher, da die Applikation von Grund auf programmiert werden muss.
Wird mit Prototypen 2.3.2 gearbeitet, ist die Implementierung weniger aufwendig. 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

[1]Tool um eine GUI zu simulieren

notwendigen Verbindungen zwischen GUI-Elementen zu behalten.
Ist der Prototyp programmiert werden, ist die Struktur fertig und muss gegebenenfalls um Funktionen ergänzt werden, die im Grobentwurf nicht berücksichtigt wurden.
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[11].

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[11].
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[11].
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.[11].
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 2.2 fertiggestellt werden. Bestätigt der Kunde das Resultat kann mit dem Training betreuender Personen und Benutzern begonnen werden[11].

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[12].
Diese und noch weitere Faktoren beeinflussen den Aufwand für die Automatisierung von Test. Es ist also wichtig sich Gedanken über mögliche Probleme zu machen, bevor man den Zeitaufwand auf sich nimmt um die Automatisierung umzusetzen.

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.
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 Methoden von anderen Klassen übernehmen und bei Bedarf anpassen. Eine Unterklasse kann von beliebig vielen ‘Eternklassen ’ erben. Vererbung unterstützt

Anpassungen. Änderungen in ‘Eternklassen ’ werden ohne weiteren Aufwand von den Unterklassen übernommen. Änderungen in Unterklassen führen nicht zu Änderungen in den ‘Eternklassen ’ .
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 das gesamte System umzubauen.
Die Objektorientierung kann die reale Welt näher abbilden, als die prozedurale Programmierung[13].

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, den auch Jahre nach dem der Code zum letzten Mal verwendet wurde, kann es passieren, dass man darauf angesprochen wird. In dieser Situation ist es hilfreich, wenn der Code verständlich geschrieben ist, da Gedanken die während der Entwicklung berücksichtigt wurden, nicht immer dokumentiert sind[14].
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.
Es gibt mehrere Prinzipien, nach denen sich ‘Clean-Code’ erreichen lässt.
Die bekanntesten sind die SOLID-Prinzipien von Robert Cecil Martin und die DRY-Prinzipien von Andy Hunt and Dave Thomas.

2.6.1 SOLID-Prinzipien

Die SOLID-Prinzipien bestehen aus fünf einzelnen Prinzipien, deren Anfangsbuchstaben zusammengesetzt SOLID ergeben[15].
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[15].

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[15].

Liskov Substitution Principle

Dieses Prinzip besagt, dass jede Klasse durch eine von ihr abgeleitete Klasse ersetzt werden kann[15].

Interface Segregation Principle

Nach diesem Prinzip sollten Interfaces nur die Funktionen bereitstellen, die auch tatsächlich notwendig sind[15].

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[15].

2.6.2 DRY-Prinzipien

Ausgeschrieben bedeutet die Abkürzung DRY 'Don't Repeat Yourself'. Das Code mehrmals vorkommt ist nicht selten, da an unterschiedlichen Stellen eine ähnliche Funktion benötigt wird. Der Code der an einer Stelle funktioniert, kann auf Grund von kleinen Unterschieden an einer anderen Stelle Probleme verursachen und muss angepasst werden. Ist der ursprüngliche Code kommentiert, werden oft auch Kommentare mit kopiert, die dann ebenfalls angepasst werden müssen. Wird das vergessen, sind gleiche Kommentare im Code verteilt, die nicht an jeder Stelle passend sind[15].

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 Repositries[16].
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[17].

3 Parametertabellenkonfigurator

Mit Hilfe des „Parametertabellenkonfigurators“ sollen die Abläufe der Konfiguration 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, 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 potentielle Fehlerquellen zu eliminieren. Hierfür soll selbstständig ein „Parametertabellenkonfigurator“ entwickelt werden.

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 Anforderungsen 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 Bedeutung einer übersichtlichen GUI.

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 szu 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 3érstellt wurde. Das Tool bietet nahezu alle gängigen GUI-Elemente an und kann diese miteinander verknüpfen, so dass schon in einem frühen Stadium die Funktionalität gegeben ist und besprochen werden kann. 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.
Ein Fenster dient dem Zweck bei erster Benutzung den Pfad zu einer Datenbank, in der die Beschreibungen gespeichert sind, einzugeben. Im nächsten Fenster lässt sich die Auswahl der Parametertabellen konfigurieren. Nach der Auswahl einer Parametertabelle wird diese im Hauptfenster 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.

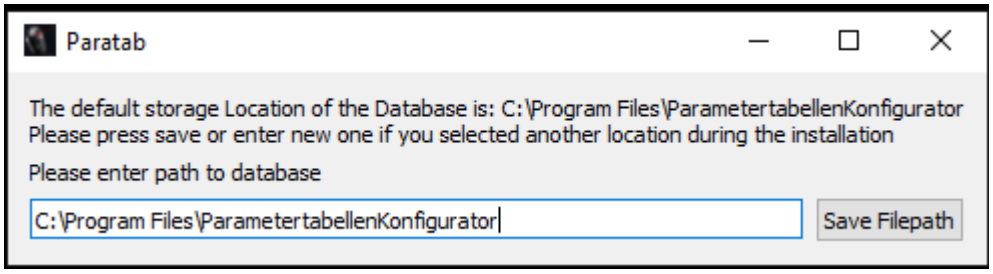

Abbildung 1: Fenster um den Dateipfad der Datenbank einzugeben, es gibt einen Standardpfad

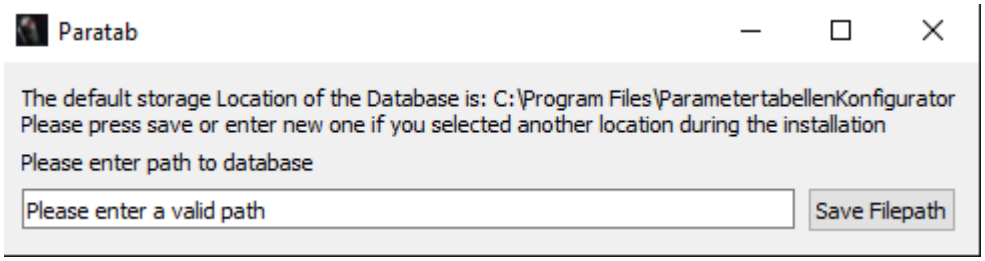

Abbildung 2: Wurde ein nicht vorhandener Pfad eingegeben erscheint die Fehlermeldung "Please enter a valid path"

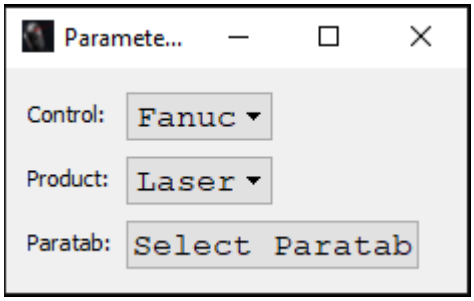

Abbildung 3: Fenster um Auswahl der Parametertabellen zu konfigurieren

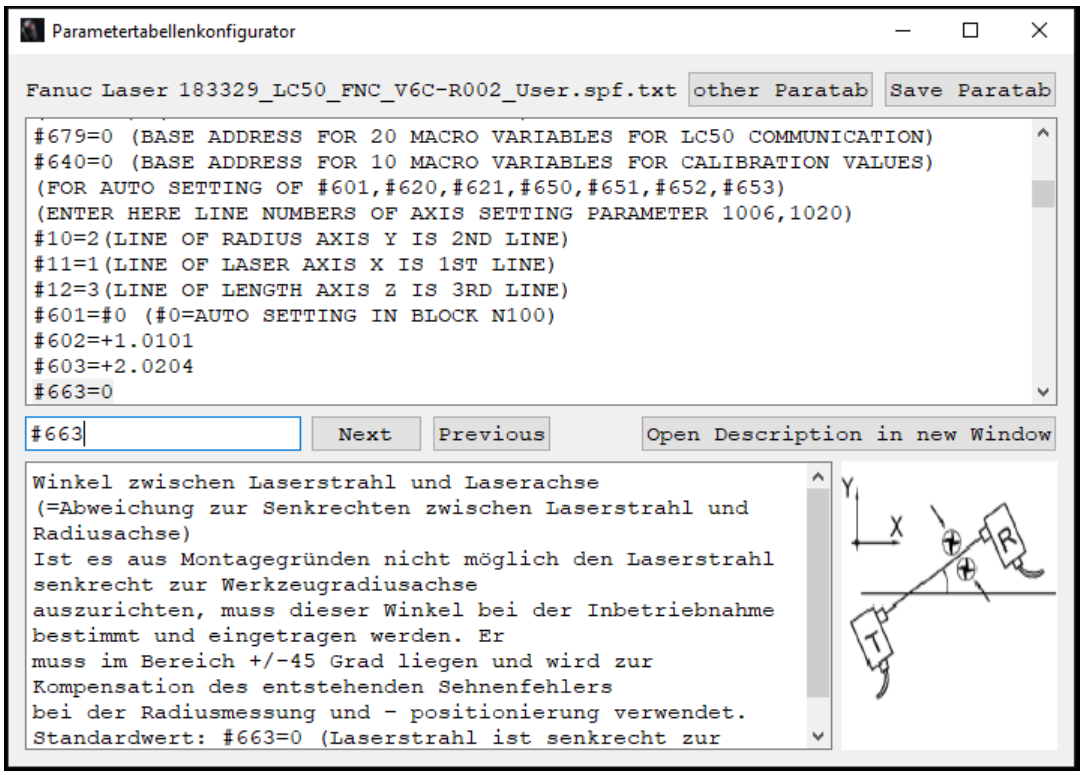

Abbildung 4: Fenster für die Anzeige der Parametertabellen und deren Beschreibungen

In Abbildung 4 werden im Oberen Bereich werden Informationen zur aktuell ausgewählten Parametertabelle angezeigt. Oben rechts gibt es Steuerelemente um eine andere Parametertabelle auszuwählen, oder 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 "Nextünd "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.

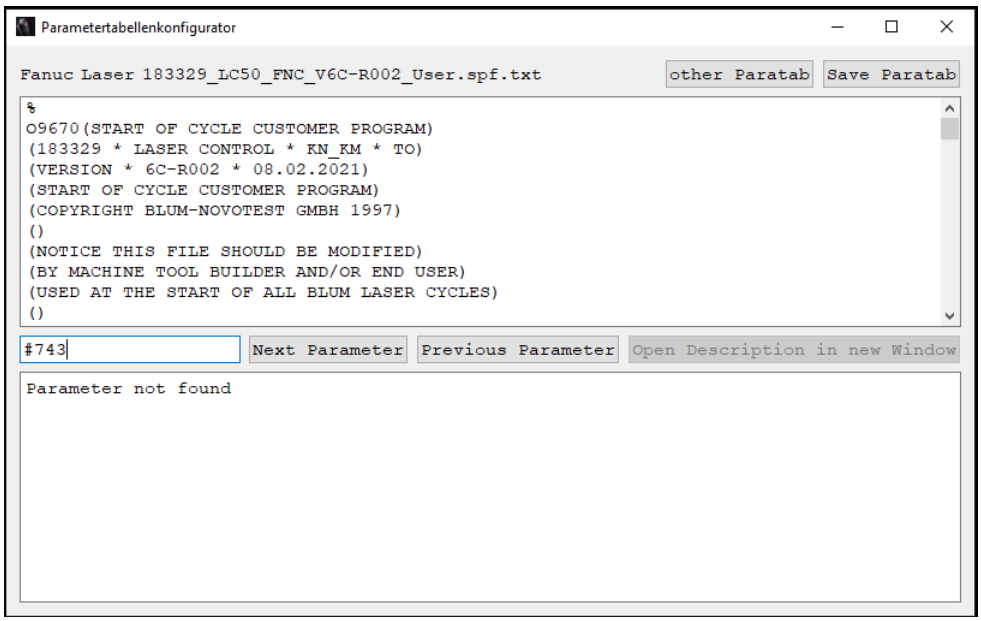

Abbildung 5: Wird ein nicht vorhandener Parameter in das Feld eingegeben wird eine Fehlermeldung angezeigt.

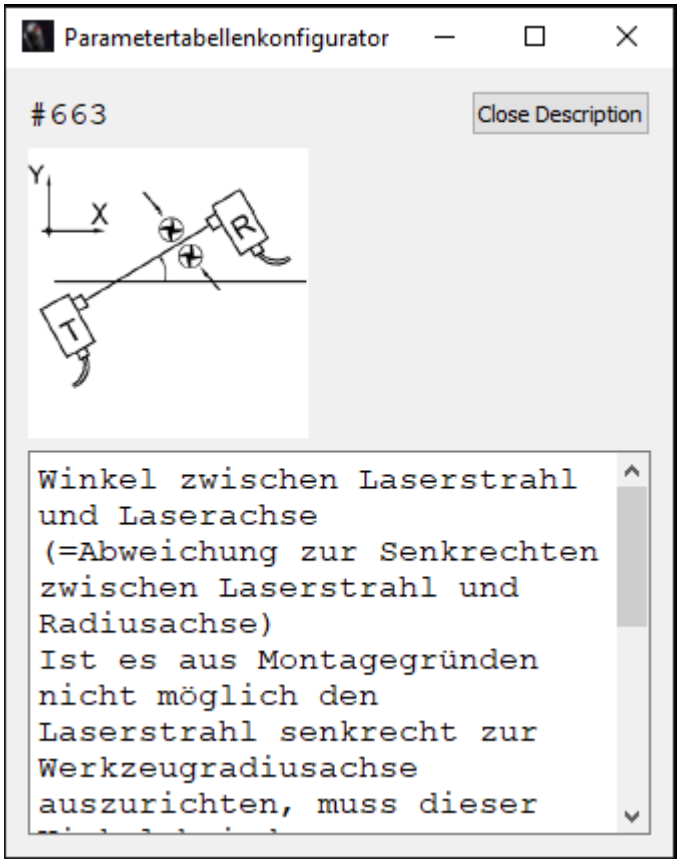

Abbildung 6: Isolierte Ansicht einer Beschreibung eines Parameters

Das in Abbildung 6 sichtbare Bild 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 ermöglicht es Anpassungen an den einzelnen Modulen vorzunehmen, ohne die höheren Ebenen grundlegend anzupassen.

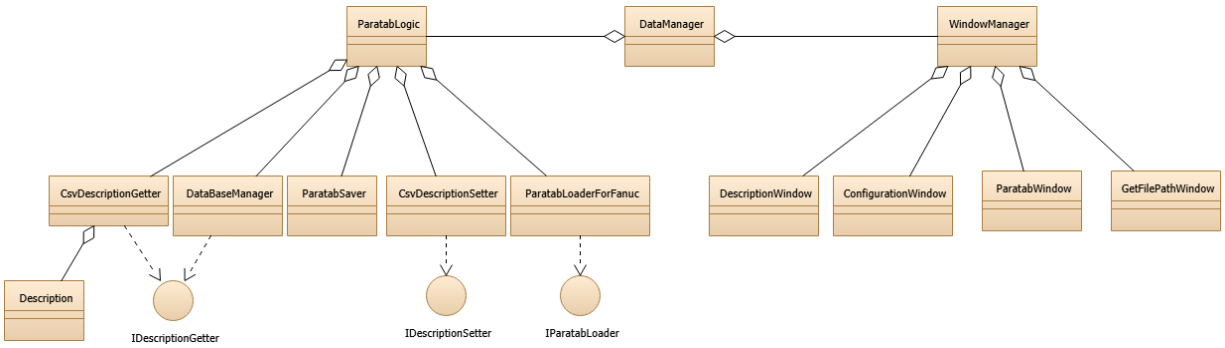

Abbildung 7: UML-Diagramm der Software ohne Inhalte der Klassen

Das Diagramm in Abbildung 7 dient dazu die Modularität der Software zu veranschaulichen, daher wurde auf die Darstellung von Inhalten der Klassen verzichtet

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 Signalen von GUI-Elementen können Signale auch selbst definierte Signale eingesetzt werden um Methoden aufzurufen. Signale vonGUI-Elementen werden ohne weitere Implementierungen durch den Entwickler gesendet. Selbstdefinierte Signale müssen mit einem Befehl gesendet werden. Signale können auch Argumente 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 könne, 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 Bearbeitung der Parametertabelle

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.

3.4.4 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[18].
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.

3.4.5 Tests

Die Software wird abgesehen von der GItLab Pipeline 3.4.6 manuell getestet. Dies hat den Grund, dass die Automatisierung von Tests anspruchsvoll und zeitaufwendig ist 2.4.1.

3.4.6 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.

4 LC-Vision

Literatur

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

[2] S. Tiwari, S. S. Rathore, and A. Gupta, "Selecting requirement elicitation techniques for software projects," in *2012 CSI Sixth International Conference on Software Engineering (CONSEG)*, pp. 1–10, IEEE, 2012.

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

[4] J. Eckhardt, A. Vogelsang, and D. M. Fernández, "Are "non-functional"requirements really non-functional? an investigation of non-functional requirements in practice," in *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, (New York, NY, USA), p. 832–842, Association for Computing Machinery, 2016.

[5] 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.

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

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

[8] 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.

[9] 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.

[10] J. Matejka, T. Grossman, and G. Fitzmaurice, "Patina: Dynamic heatmaps for visualizing application usage," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, (New York, NY, USA), p. 3227–3236, Association for Computing Machinery, 2013.

[11] J. J. Rakos, *Software Project Management*. Prentice Hall, 1990.

[12] M. Fewster and D. Graham, *Software test automation*. Addison-Wesley Reading, 1999.

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

[14] 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.

[15] V. Sarcar, "Use the dry principle," in *Simple and Efficient Programming with C#*, pp. 109–128, Springer, 2021.

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

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

[18] E. F. Codd, *Relational Database: A Practical Foundation for Productivity*, p. 1981. New York, NY, USA: Association for Computing Machinery, 2007.