

Bachelorarbeit
in
Medieninformatik
Zu Erlangung des Grades Bachelor of Science

**Entwicklung eines speziellen DevOps
Projekt-Konfigurator Werkzeuges für
Projektinitialisierung /– aktualisierung**

Referent: Herr Prof. Dr. Eisenbiegler
Korreferent: Herr Dr.-Ing- Christoph Rathfelder
Vorgelegt am: 24.02.2022
Vorgelegt von: Nick Stecker
25195
Haselstiegstr. 5/1
78052 Villingen
nick.stecker@hs-furtwangen.de

Eidesstattliche Erklärung

Ich versichere, dass ich die vorstehende Arbeit selbständig verfasst und hierzu keine anderen als die angegebenen Hilfsmittel verwendet habe. Alle Stellen der Arbeit, die wörtlich oder sinngemäß aus fremden Quellen entnommen wurden, sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt oder an anderer Stelle veröffentlicht.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben kann.

[VS-Villingen, 24.02.2022 Nick Stecker]

Abstract

Die Entwicklung großer Softwareprojekte kann eine sehr schwierige Aufgabe sein, insbesondere wenn das Projekt nicht gut strukturiert ist. Darüber hinaus sind viele moderne Softwareprojekte aus modularen Teilen aufgebaut, die später für verschiedene Anwendungsfälle wiederverwendet werden können. Um die Wiederverwendbarkeit dieser Module zu gewährleisten, werden sie oft selbst als Softwareprojekte mit einer eigenen Struktur und zugehörigen Werkzeugen erstellt. Es ist kein unwahrscheinliches Szenario, dass sich die Struktur oder die Werkzeuge dieser Projekte im Laufe der Entwicklungszeit verändern, was zu einer Aktualisierung auffordert. Dies manuell zu tun, kann eine mühsame Aufgabe sein, insbesondere wenn eine große Anzahl dieser Module vorhanden ist. Um dieses Problem zu lösen, wurde im Rahmen dieser Thesis ein Werkzeug mit einer grafischen Benutzeroberfläche entwickelt, mit der die Strukturänderungen auf einfache und leicht verständliche Weise auf jedes Softwaremodul angewendet werden kann.

Inhaltsverzeichnis

| | |
|--|----|
| Eidesstattliche Erklärung | 2 |
| Abstract | 3 |
| Abbildungsverzeichnis | 6 |
| Abkürzungsverzeichnis | 8 |
| 1. Einleitung | 9 |
| 1. Grundlagen | 11 |
| 1.1 Softwareprojekte | 11 |
| 1.2 Die Bedeutung von DevOps | 11 |
| 1.3 Zusammenfassung | 13 |
| 2. Stand der Technik | 15 |
| 2.1 Versionsverwaltung | 15 |
| 2.2 Integrierte Entwicklungsumgebung (IDE) | 15 |
| 2.2.1 Eclipse IDE für C/C++ | 16 |
| 2.2.2 CLion IDE | 22 |
| 2.2.3 Visual Studio Code | 30 |
| 2.3 Zusammenfassung | 35 |
| 3. Lösungsansatz | 37 |
| 4. Softwareentwurf | 39 |
| 4.1 Die Programmiersprache und das GUI Framework | 39 |
| 4.2 GUI Skizzen | 40 |
| 4.2.1 Das Hauptmenü | 42 |
| 4.2.2 Fenster für die Projekterstellung | 44 |
| 4.2.3 Fenster für die Projektaktualisierung | 46 |
| 4.2.4 Fenster für die Erweiterung der Projektmodule | 50 |
| 4.2.5 Fenster für die Ausführung der Dienstprogramme | 52 |
| 4.3 Softwarearchitektur | 54 |
| 4.3.1 Basis-Architektur | 54 |

| | | |
|-------|--|-----|
| 4.3.2 | Vollständige Architektur | 57 |
| 4.4 | Zusammenfassung | 61 |
| 5. | Implementierung der Software | 63 |
| 5.1 | Interne Projekt-Schablone | 63 |
| 5.2 | Implementierung der Software | 66 |
| 5.2.1 | Das Hauptmenü | 66 |
| 5.2.2 | Fenster für die Projekterstellung | 69 |
| 5.2.3 | Fenster für die Projektaktualisierung..... | 75 |
| 5.2.4 | Fenster für die Erweiterung der Projektmodule..... | 86 |
| 5.2.5 | Fenster für die Ausführung der Dienstprogramme | 92 |
| 5.2.6 | Zusammenfassung | 96 |
| 5.3 | Mögliche Erweiterungspunkte | 99 |
| 5.4 | Zusammenfassung | 100 |
| 6. | Zusammenfassung | 101 |
| | Literatur | 104 |

Abbildungsverzeichnis

| | |
|---|----|
| Abbildung 1: "New Project"-Dialog in Eclipse | 18 |
| Abbildung 2: C++ Wizard in Eclipse | 19 |
| Abbildung 3: "Import"-Dialog Seite 1 in Eclipse | 20 |
| Abbildung 4: "Import"-Dialog Seite 2 in Eclipse | 21 |
| Abbildung 5: "Pfad auswählen"-Dialog in CLion | 23 |
| Abbildung 6: "Trust Project"-Dialogin CLion | 23 |
| Abbildung 7: "Open Project"-Dialog in CLion..... | 24 |
| Abbildung 8: "Open as Project"-Dialog in CLion | 24 |
| Abbildung 9: "New Project"-Dialog für Executable Projekte in CLion | 25 |
| Abbildung 10: "New Project"-Dialog für Library Projekte in CLion | 26 |
| Abbildung 11: "Save File as Template"-Dialog in CLion | 27 |
| Abbildung 12: Liste aller Dateivorlagen in CLion | 28 |
| Abbildung 13: Kopie einer Dateivorlage in CLion | 29 |
| Abbildung 14: Explorer in VSCode | 31 |
| Abbildung 15: Projekt als Template speichern in VSCode..... | 32 |
| Abbildung 16: Standardspeicherorte der Projekte der VSCode Extension | 32 |
| Abbildung 17: Änderung des Speicherortes für die Projekte der VSCode Extension | 33 |
| Abbildung 18: Projekt von einem Template erstellen in VSCode..... | 34 |
| Abbildung 19: Art der Integration für Platzhalter der VSCode Extension..... | 35 |
| Abbildung 20: Skizze für den Screen des Hauptmenüs..... | 42 |
| Abbildung 21: Skizze für den Screen der Projekterstellung | 44 |
| Abbildung 22: Skizze für den Screen der Projektaktualisierung | 46 |
| Abbildung 23: Skizze für den Screen der Projektmodulerweiterung | 50 |
| Abbildung 24: Skizze für den Screen der Dienstprogrammausführung | 52 |
| Abbildung 25: Basis Klassendiagramm | 54 |
| Abbildung 26: Layout der Klasse QMainWindow [29]..... | 55 |
| Abbildung 27: Struktur der Projekt-Schablone..... | 63 |
| Abbildung 28: Implementierung des Hauptmenüs | 66 |
| Abbildung 29: Flussdiagramm für den Suchmechanismus | 67 |
| Abbildung 30: Implementierung des Fensters für die Projekterstellung..... | 69 |
| Abbildung 31: Flussdiagramm für die Projekterstellung..... | 70 |
| Abbildung 32: Flussdiagramm für den Check der Projekterstellungsformularwerte.. | 71 |
| Abbildung 33: Flussdiagramm für Erstellung des Projektordners | 72 |

| | |
|---|----|
| Abbildung 34: Projekt-Konfiguration JSON-Datei | 73 |
| Abbildung 35: Implementierung des Fensters für die Projektaktualisierung | 75 |
| Abbildung 36: Flussdiagramm der Initialisierung der Projektaktualisierungsdaten ... | 76 |
| Abbildung 37: Flussdiagramm für die Erstellung der Baumliste..... | 77 |
| Abbildung 38: Flussdiagramm für das Hinzufügen der Splitterwidgets | 79 |
| Abbildung 39: Flussdiagramm für die Projektaktualisierung | 83 |
| Abbildung 40: Implementierung des Fensters für die Projektmodulerweiterung | 86 |
| Abbildung 41: Flussdiagramm für das Hinzufügen der lokalen Abhängigkeiten | 87 |
| Abbildung 42: Flussdiagramm für die Erstellung eines Projektmoduls | 89 |
| Abbildung 43: Flussdiagramm für den Check der Projektmodulformularwerte | 90 |
| Abbildung 44: Implementierung des Fensters für die Dienstprogrammausführung .. | 92 |
| Abbildung 45: Flussdiagramm für die Auflistung der Dienstprogramme | 93 |
| Abbildung 46: Flussdiagramm für die Ausführung des "--help"-Befehls..... | 94 |
| Abbildung 47: Flussdiagramm für die Ausführung der benutzerdefinierten Befehle . | 95 |

Abkürzungsverzeichnis

CI/CD *Continuous Integration und Continuous Delivery*

DevOps Development und IT Operations

GUI Graphical User Interface

IDE Integrated Development Environment

JRE *Java Runtime Environment*

JSON *JavaScript Object Notation*

1. Einleitung

Die Entwicklung großer Softwareprojekte kann eine sehr schwierige Aufgabe sein, insbesondere wenn das Projekt nicht gut strukturiert ist. Darüber hinaus sind viele moderne Projekte aus modularen Teilen aufgebaut, die später für verschiedene Anwendungsfälle wiederverwendet werden können. Um die Wiederverwendbarkeit und Verbesserung dieser Module zu verbessern, werden sie oft selbst als Softwareprojekte mit einer eigenen Struktur und zugehörigen Werkzeugen erstellt. Die Struktur der Softwareprojekte wird dabei durch eine Projekt-Schablone, mit vordefinierten Dateien und Unterordern vorgegeben. Die Dateien der Schablone enthalten in der Regel Platzhalter an verschiedenen Stellen, die für jedes neue Projekt mit realen Projektinformationen ersetzt werden müssen. Aus dieser Ausgangssituation folgen zwei maßgebliche Probleme:

Das erste Problem bezieht sich auf die Erstellung von neuen Softwareprojekten. Hierbei wird die Projekt-Schablone zu einem gewissen Grad konfiguriert, sodass aus der Schablone ein reales Softwareprojekt entsteht. Dazu müssen zum einen die Platzhalter, die sich in den Dateien befinden, mit real Daten des Projekts ersetzt werden. Zum anderen müssen nicht benötigte Dateien und Unterorder der Schablone, für das neue Projekt entfernt werden. Dies manuell zu tun ist mühsam und fehleranfällig. Es existieren Werkzeuge, mit denen gewisse Teile der Projekterstellung automatisiert werden können, allerdings bietet keines dieser Werkzeuge einen Funktionsumfang an, mit dem die Projekterstellung vollständig automatisiert werden kann. Einzelne Konfigurationen müssen trotz der Werkzeuge manuell durchgeführt werden.

Das zweite Problem bezieht sich auf die Aktualisierung von bestehenden Softwareprojekten. Es ist nämlich kein unwahrscheinliches Szenario, dass sich die Struktur oder die Tools dieser Projekte im Laufe der Zeit und mit zunehmendem Entwicklungswissen ändern und daher zu einer Aktualisierung auffordern. Das bedeutet, dass die Änderungen, die in der Projekt-Schablone vorgenommen wurden, nachträglich in allen bestehen Softwareprojekten, die aus dieser Schablone entstanden sind, ebenfalls angewendet werden müssen. Es existieren keine Werkzeuge, die diese Aktualisierung automatisieren bzw. erleichtern würden, weshalb die Aktualisierung der Projekte bisher manuell durchgeführt werden muss. Diese

Vorgehensweise ist gleichermaßen mühsam, wie fehleranfällig, insbesondere wenn es eine große Anzahl dieser Softwareprojekt gibt.

Um die beiden Probleme zu lösen, wurde im Rahmen dieser Thesis ein spezielles Projekt-Konfigurator Werkzeug entwickelt. Der Projekt-Konfigurator besitzt eine grafische Benutzeroberfläche, mit der Softwareprojekte auf eine einfache Weise erstellt und aktualisiert werden können.

1. Grundlagen

In diesem Kapitel wird kurz erläutert, was ein Projekt im Kontext der Softwareentwicklung bedeutet, was DevOps ist, wie es im Rahmen eines Softwareprojekts funktioniert und schließlich, wie DevOps auf konzeptioneller Ebene durchgeführt wird.

1.1 Softwareprojekte

Ein Softwareprojekt ist im Kontext dieser Bachelorarbeit ein Ordner, der alle notwendigen Dateien und Unterordner beinhaltet, die für die Entwicklung des Softwareprojektes notwendig sind. Die wichtigsten Dateien sind dabei Quellcodedateien, in denen der Quellcode für das Projekt steht, und Konfigurationsdateien, in denen die wichtigsten Einstellungen für das Projekt und für die Werkzeuge des Projektes stehen.

1.2 Die Bedeutung von DevOps

Bei der Bezeichnung DevOps handelt es sich um ein Kofferwort, dass sich aus den Begriffen „Development (Entwicklung)“ und „IT Operations (IT-Betrieb)“ zusammensetzt. Damit sind die jeweiligen Organisationseinheiten im IT-Bereich gemeint, die in einer traditionellen Organisation klassischerweise getrennt sind und mit unterschiedlichen Zielsetzungen arbeiten [1, 2]. Ohne den DevOps-Ansatz kommt es zu einer mangelnden Zusammenarbeit zwischen den beiden Bereichen. Das Ziel der Softwareentwicklung ist die Bereitstellung der fertigen Software („Release“). Die anschließende Installation („Deployment“) ist hingegen der Startpunkt des IT-Betriebs. Wurden in der Entwicklung die Anforderungen des IT-Betriebs für die Installation nicht berücksichtigt, kann das zu Verzögerungen oder fehlschlagen des Deployments führen [3].

Angesichts derartiger Probleme ist DevOps entstanden. Es umfasst eine Reihe von Technologien, Prozessen und Werkzeugen, die darauf ausgelegt sind, typische Probleme in der Zusammenarbeit zwischen Entwicklungs- und Betriebseinheiten zu lösen und dadurch das Kundenerlebnis und die Kundenzufriedenheit zu verbessern. Entscheidend dafür sind Veränderungen in der Zusammenarbeit dieser Bereiche sowie eine möglichst große Automatisierung der täglichen Aufgaben [3]. Um die beiden Welten der Entwicklung und des IT-Betriebs zusammenzubringen, werden die Prozesse Continuous Integration und Continuous Delivery eingesetzt.

Continuous Integration und Continuous Delivery (CI/CD) ist ein grundlegender Prozess innerhalb von DevOps. Es umfasst eine Reihe von Tools, die nacheinander als eine Art Werkzeugkette (eng.: Toolchain) eingesetzt werden. Dieser Prozess wird auch CI/CD-Pipeline genannt. Die CI/CD-Pipeline ist im Wesentlichen ein Arbeitsablauf, mit dem der Prozess der Softwarebereitstellung automatisiert wird. Ohne die Pipeline müssten die DevOps Entwickler den Arbeitsablauf manuell durchführen, was zeitaufwändig und fehleranfällig wäre. Der Aufbau einer CI/CD-Pipeline ist nicht standardisiert, jedes DevOps-Team wählt die Tools selbst aus, mit denen sie die Pipeline aufbauen möchten. In der Regel decken die verwendeten Tools aber folgende vier Phasen ab [4]:

Quellcode-Entwicklungsphase: In dieser Phase arbeiten Entwickler an dem Quellcode einer bestehenden Anwendung. Der Quellcode ist dabei in einem zentralen Repository wie GitHub / GitLab gespeichert. Um an dem Quellcode zu arbeiten, erstellen die Entwickler einen neuen Zweig im Repository und klonen (kopieren) diesen anschließend in einen lokalen Ordner auf dem Dateisystem. Anschließend können die Entwickler mit einer Entwicklungsumgebung in dem neuen Zweig eine neue Funktion einbinden oder bestehende Fehler beheben. Nachdem neuer Code geschrieben wurde, übertragen (pushen) die Entwickler den neuen Code in das zentrale Repository zurück [4].

Build-Phase: Die Build-Phase ist ein Prozess bei dem eine ausführbare Software erzeugt wird [5]. Sie wird ausgelöst, wenn neuer Code in das zentrale Repository übertragen (gepusht) wird. Dabei werden die Codes der Repository-Zweige gesammelt und in eine, für den Computer verständliche, Sprache übersetzt („Code-Kompilierung“). Anschließend wird der kompilierte Code an Bibliotheken „gelinkt“. Durch die Verlinkung werden die einzelnen Programmmodule zu einer eigenständigen, ausführbaren Software verbunden. Der Build-Vorgang wird dabei von einzelnen Build-Tools (teil der CI/CD-Pipeline) durchgeführt. Sie lesen dazu eine Anweisungsdatei, in der allgemeine Rahmenbedingungen sowie spezielle Anweisungen für die einzelnen Schritte eines Build-Prozesses steht [6]. Wenn der übertragene Code fehlerhaft ist, kann der Build-Prozess fehlschlagen. Durch die CI/CD-Pipeline erfahren die Entwickler, an welcher Stelle der Build-Prozess abgebrochen wurde und welche Stelle im Code dafür verantwortlich ist. Dadurch können die Entwickler so schnell wie möglich einen Fix implementieren [4].

Testphase: Nachdem ein fehlerfreier Build erstellt wurde, durchläuft der Build mehrere Tests, um sicherzustellen, dass der Code das macht, was er machen muss. Unter anderem gehören dazu meist Komponententest (eng.: „Unit-Tests“), bei denen der Code in kleine Einzelteile zerlegt wird, die daraufhin einzeln auf verschiedene Funktionen getestet werden. Des Weiteren können in dieser Phase auch Benutzerakzeptanz-, Sicherheits-, Last- oder andere Test durchgeführt werden. Jede Tests haben dabei andere Testkriterien [4].

Verteilungsphase: Die Verteilungsphase ist die letzte Phase der Pipeline. Während dieser Phase wird eine fertige Anwendung verteilt, bis alle Zielsysteme über die aktualisierte Version der Anwendung verfügen. Für dieses Vorgehen gibt es verschiedene Verteilungsstrategien, die angewendet werden können. Die Auswahl der besten Strategie richtet sich nach Art der Anwendung, dem Grad der Überarbeitung, der Zielumgebung und anderen Faktoren [4].

1.3 Zusammenfassung

In Kapitel 2 wurden die Grundlagen erklärt, die wichtig für das weitere Verständnis dieser Bachelorarbeit sind. Nachfolgend werden diese Grundlagen kurz zusammengefasst und dabei erklärt, weshalb diese für das weitere Verständnis dieser Arbeit wichtig sind:

Softwareprojekte sind Projekt bzw- Ordner, in denen sich alle notwendigen Dateien und Unterordner des Projektes befinden. Zu den wichtigsten Dateien zählen unter anderem die Quellcodedateien, mit denen die Software des Softwareprojektes entwickelt wird. Es ist wichtig zu verstehen was ein Softwareprojekt ist, da der Projekt-Konfigurator, der im Rahmen dieser Arbeit entwickelt wurde, Softwareprojekte erstellen und aktualisieren kann.

DevOps umfasst eine Reihe von Technologien, Prozessen und Werkzeuge, mit denen die Probleme in der Zusammenarbeit zwischen Entwicklungs- und Betriebseinheiten gelöst werden und dadurch das Kundenerlebnis und die Kundenzufriedenheit verbessert wird. Dazu benutzt der DevOps-Ansatz einen CI/CD-Prozess, bei dem eine Reihe von Tools, nacheinander in einer Toolchain eingesetzt werden. Dieser ist im Wesentlichen ein Arbeitsablauf, mit dem der Prozess der Softwarebereitstellung automatisiert wird. Es ist wichtig zu verstehen was ein CI/CD-Prozess ist und welche Schritte dabei in der Regel automatisiert werden, da der Projekt-Konfigurator

Softwareprojekte erstellt, die während ihrer Entwicklungszeit mehrfach einen CI/CD-Prozess durchlaufen. Damit die CI/CD-Prozesse nicht fehlschlagen, müssen die Softwareprojekte eine bestimmte Verzeichnisstruktur aufweisen. Diese ist durch eine spezielle Vorlage bzw. Schablone vorgegeben. Der Projekt-Konfigurator erstellt die Softwareprojekt nach Vorgabe dieser Schablone.

2. Stand der Technik

In diesem Kapitel werden einige mögliche Lösungen, für die in Kapitel 1 erklärten Probleme aufgeführt und anhand der zuvor genannten Anforderungen bewertet.

2.1 Versionsverwaltung

Eine Versionsverwaltung, auch bekannt als Versionskontrolle, ist eine spezielle Software, mit der Änderungen am Softwarecode verfolgt und verwaltet werden kann. Die Versionsverwaltungssoftware verfolgt dabei jede Änderung am Code und speichert diese in einer Datenbank. Wenn ein Entwickler einen Fehler macht, kann er zurücktreten und mit früheren Codeversionen vergleichen, um das Problem zu beheben, ohne die Teammitglieder zu stören [7]. Nachfolgend wird mithilfe der Versionsverwaltungssoftware „Git“, laut dem „Stack Overflow Developer Survey 2021“ die meistgenutzte Versionsverwaltungssoftware 2021 [8], erklärt wie Projekte auf Basis einer Projekt-Schablone erstellt werden können.

Um ein neues Projekt mit Git zu erstellen, muss die Projekt-Schablone als erstes aus einem online Repository in ein neues lokales Verzeichnis geklont werden. Das bedeutet, dass durch den entsprechenden Git-Befehl, eine Kopie des Repositories in dem neuen Verzeichnis erstellt wird. Anschließend muss die Kopie manuell bearbeitet werden, um aus der Kopie ein reales Softwareprojekt zu machen. Diese Bearbeitungen sind zeitaufwändig und fehleranfällig.

Die Aktualisierung eines Projektes, nachdem Änderungen an der Projekt-Schablone im Repository vorgenommen wurden, kann mit Git durch Merge-Konflikte gelöst werden, was zeitaufwändig und fehleranfällig ist.

2.2 Integrierte Entwicklungsumgebung (IDE)

Um einfache Programme entwickeln zu können, benötigen Softwareentwickler grundsätzlich nur zwei Werkzeuge. Einen Code-Editor, um den Quellcode zu schreiben und einen Compiler bzw. Interpreter, um die Programme in den für den Computer verständlichen, ausführbaren Maschinencode zu übersetzen. Für eine effiziente und professionelle Entwicklung von komplexeren Anwendungen sind allerdings weitere Werkzeuge notwendig, um den Entwicklern einzelne Arbeitsschritte abzunehmen oder zu erleichtern [9]. Damit bei der Entwicklung einer Anwendung nicht jedes Tool einzeln verwendet werden muss, kamen in der ersten Hälfte der 1980er Jahre sogenannte integrierte Entwicklungsumgebungen (engl.: „integrated

development environment“, IDE) auf den Markt [10]. Das Besondere der IDEs ist, dass diese die einzelnen Tools, in einer gemeinsamen Benutzeroberfläche vereinen. Dadurch können verschiedene Aufgaben in der Softwareentwicklung möglichst ohne Medienbrüche bearbeitet werden. Die wichtigsten Funktionalitäten, die die meisten modernen IDEs gemeinsam haben sind z.B.:

- Direkter Aufruf des Compilers/Interpreters
- Funktionen zur effizienten Bearbeitung des Programmcodes (bspw. Syntax Highlighting oder Autovervollständigung)
- Funktionalität zur Fehlersuche und –behebung
- Versionsverwaltung der Dateien basierend auf Git
- Funktionen zur Erstellung und Verwaltung ganzer Projekte mit mehreren Dateien [9]

Letztere ist zugleich auch die wichtigste Funktionalität in Bezug auf diese Bachelorarbeit. Im weiteren Verlauf dieses Kapitels wird diese Funktionalität anhand der drei meistgenutzten IDEs für Softwareprojekte näher analysiert. Im Besonderen werden folgende Funktionen untersucht:

- Einfache Projekterstellung
- Integrierung von eigenen Template-Projekten
- Projekterstellung anhand von Template-Projekten
- Erstellung von Platzhaltern in Template-Dateien

Nachfolgend wird mittels einer Analyse der drei meistgenutzten IDEs für C++-Projekte deutlich erläutert, warum diese nicht ausreichen, um das Problem dieser Bachelorarbeit zu lösen. Analysiert wird: die Eclipse IDE für C/C++, CLion IDE und Visual Studio Code. Der Fokus liegt hier auf C++ basierte IDEs, da der Projekt-Konfigurator vorerst nur für C++-Projekte entwickelt wurde.

2.2.1 Eclipse IDE für C/C++

Eclipse ist eine kostenlose integrierte Entwicklungsumgebung für C und C++ auf Basis der Eclipse Plattform. Die IDE ist plattformübergreifend auf den Betriebssystemen Windows, Linux und macOS verfügbar [11]. Voraussetzung der Eclipse IDE ist ein vorinstallierter Java Runtime Environment (JRE), um den Betrieb von Eclipse zu gewährleisten [12].

Eclipse ist eine projektbasierte IDE. Das bedeutet, dass alles was in Eclipse passiert, im Kontext eines Projektes passiert. Diese Projekte sind dabei lediglich Ordner, in denen sich weitere Unterordner und Quelldateien befinden. Die erstellten Projekte werden standardmäßig unter einem vorgegebenen Pfad gespeichert, welcher von Eclipse als Workspace, zu Deutsch Arbeitsbereich genannt wird. Der Workspace ist dementsprechend ein Ordner in Dateisystem, in dem alle Projekte und deren Einstellungen abgelegt werden. Zusätzlich dazu werden in dem Workspace auch die Einstellungen der Plattform selbst gespeichert. Allerdings können in Eclipse nach Bedarf auch weitere Workspaces eingerichtet werden. In der geöffneten IDE kann daraufhin zu den eingerichteten Workspaces umgeschaltet werden [13].

Um ein neues Projekt zu erstellen, muss in dem Menü **File > New > Project...** ausgewählt werden. Daraufhin öffnet sich ein „New Project“-Dialog, in dem das Format des Projektes ausgewählt werden muss. Wichtig hierbei ist nur die Auswahl in dem „C/C++“ Ordner. Darin kann unter anderem zwischen einem C- bzw. C++-Projekt ausgewählt werden (In diesem Beispiel wurde ein C++ Projekt ausgewählt).

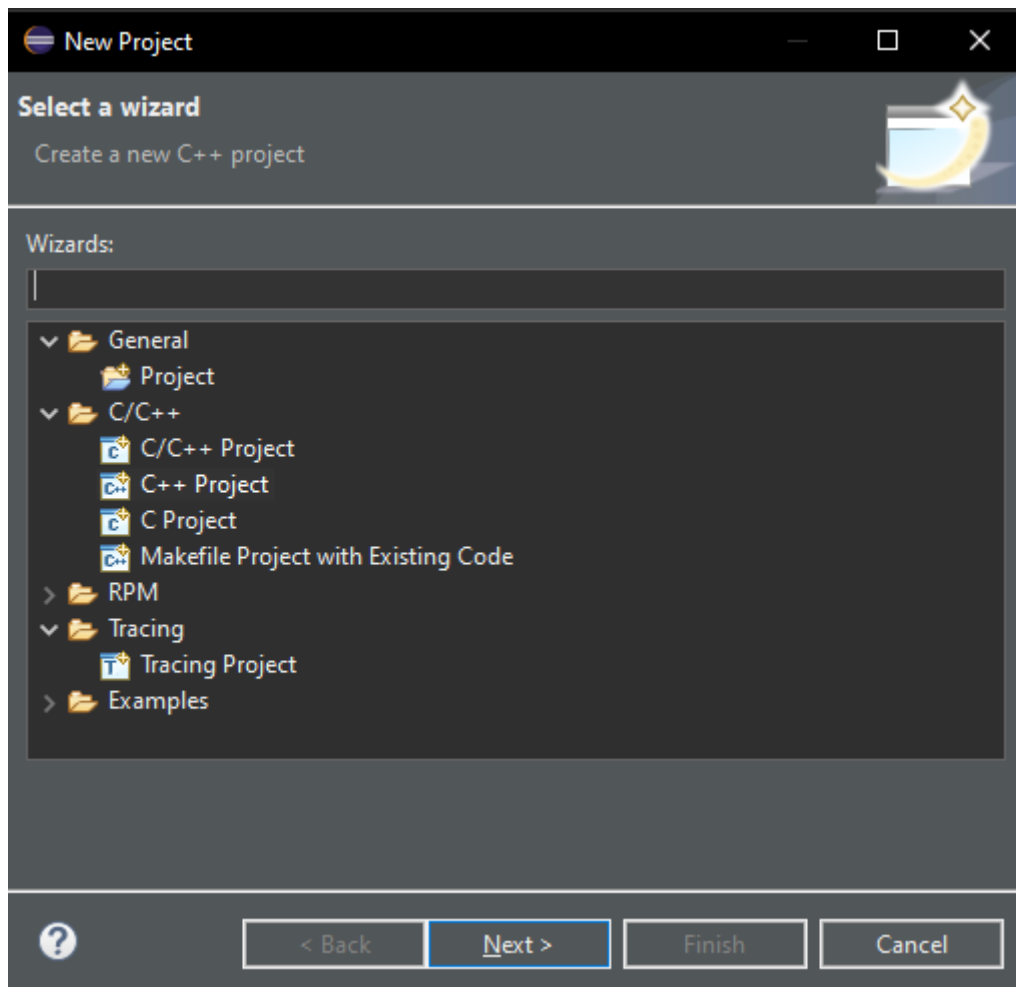


Abbildung 1: "New Project"-Dialog in Eclipse

Nachdem einer dieser zwei Projektformate ausgewählt und auf die „Next >“-Taste gedrückt wurde, öffnet sich ein Wizard, der nach dem Namen des Projekts fragt. Alternativ zu dieser Vorgehensweise kann auch im Popup-Menü der „New C/C++ Project“-Taste der Eintrag „C Project“ bzw. „C++ Project“ ausgewählt werden, um den Wizard zu öffnen.

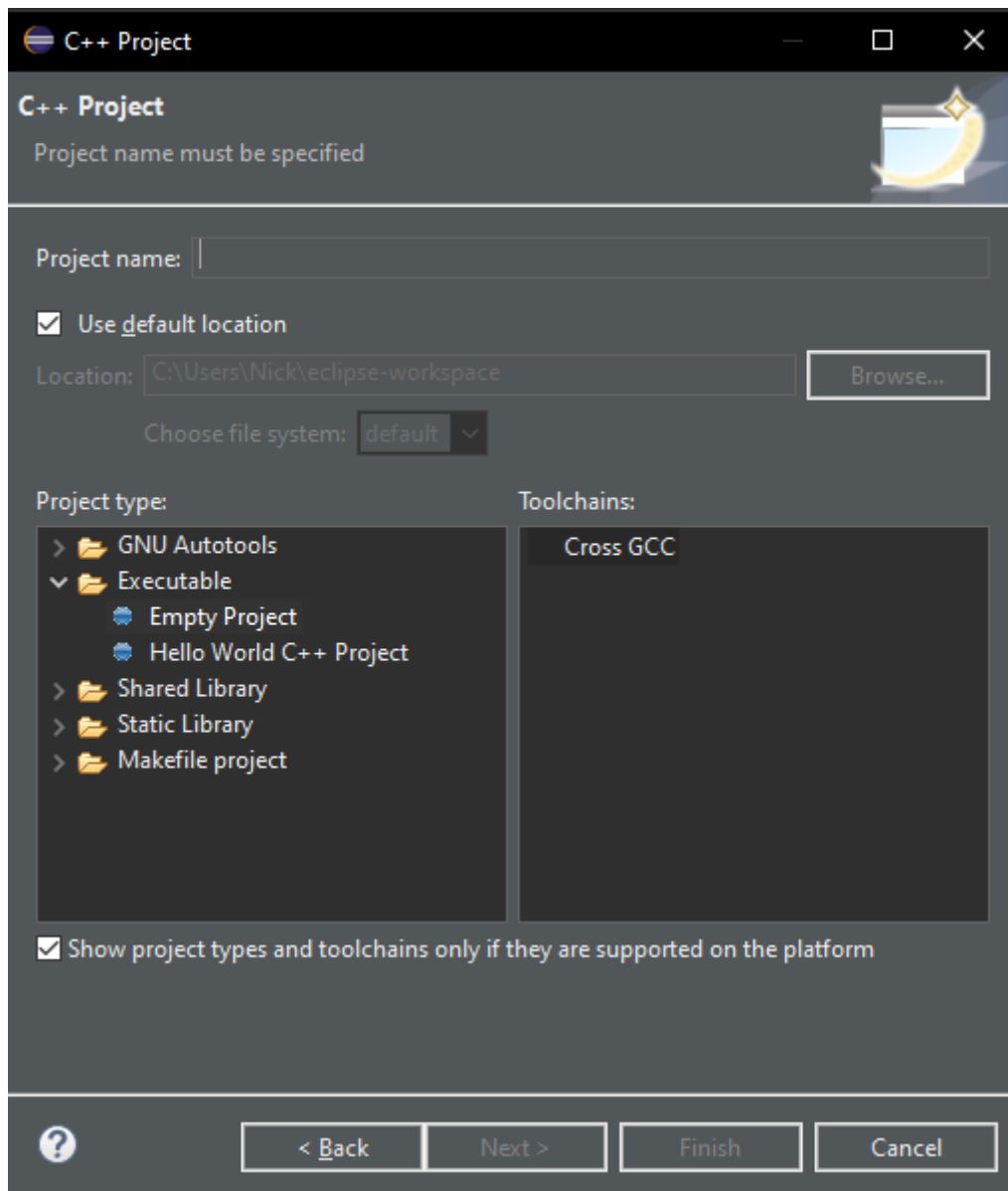


Abbildung 2: C++ Wizard in Eclipse

Als erstes muss der Name des Projekts in dem Feld „Project Name“ eingegeben werden. Daraufhin kann entschieden werden, ob das Projekt in dem geöffneten Workspace gespeichert werden soll oder unter einem anderen Pfad. Falls das Projekt nicht in dem Workspace gespeichert werden soll, muss der Haken bei „Use default location“ entfernt werden und der entsprechende Pfad unter „Location“ eingegeben werden. Als nächstes muss der Projekttyp in dem Feld „Project type“ ausgewählt werden (In diesem Beispiel wurde ein leeres Executable Projekt ausgewählt). Als letztes muss nur noch die „Finish“-Taste gedrückt werden, um alle Angaben zu bestätigen und den Wizard zu schließen. Nun sollte in dem „C/C++ Projects“-View ein neues Projekt mit dem eingegeben Namen erscheinen [13].

Dabei ist zusehen, dass neue Projekte in Eclipse nicht als leere Ordner, sondern mit einer gewissen Verzeichnisstruktur erstellt werden. Diese Struktur stammt aus intern gespeicherten Informationen, aus denen hervorgeht welche Unterordner und Dateien das Projekt enthalten soll. Diese Informationen können allerdings nicht ohne weiteres verändert bzw. erweitert werden. Um in Eclipse trotzdem Projekte aus einer eigenen Vorlage erstellen zu können gibt es zwei Möglichkeiten. Die erste Möglichkeit ist eine eigene Eclipse-Erweiterung zu programmieren, die es erlaubt ein neues Projektformat in Eclipse zu integrieren. Zusätzlich muss mit der Erweiterung ein neuer Projekt-Wizard aufrufbar sein, mit dessen Hilfe ein neues Projekt aus dem neuen Projektformat erstellt werden kann. Die zweite Möglichkeit ist der Import eines zuvor mit Eclipse erstellten Projektes, in den aktuellen Workspace. Dafür muss in dem Hauptmenü **File** > **import...** ausgewählt werden, um ein „Import“-Dialog zu öffnen.

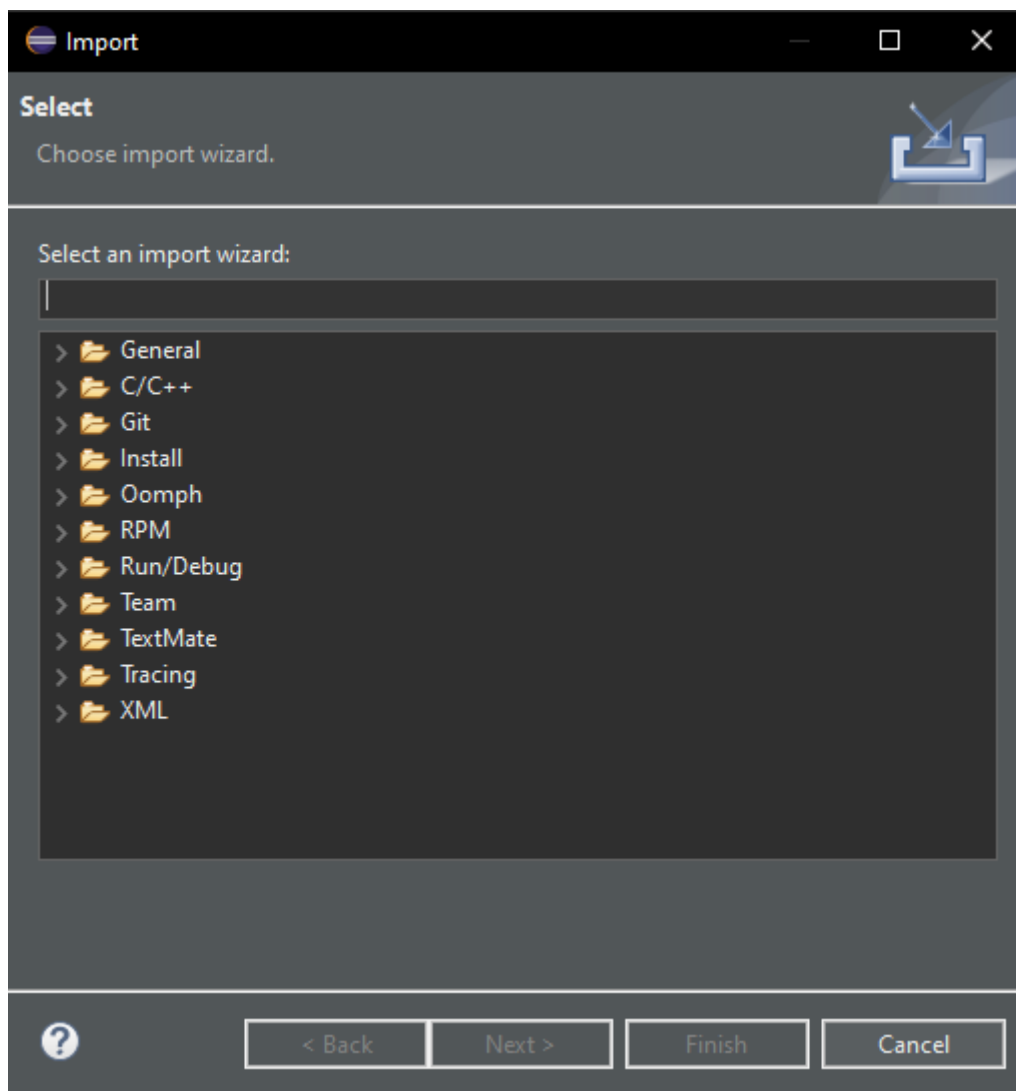


Abbildung 3: "Import"-Dialog Seite 1 in Eclipse

Auf der ersten Seite muss **General > Existing Projects into Workspace** ausgewählt und bestätigt werden. Dadurch wird die zweite Seite des Dialogs aufgerufen, auf der in „Select root directory“ das Verzeichnis der Projekt-Vorlage ausgewählt werden muss.

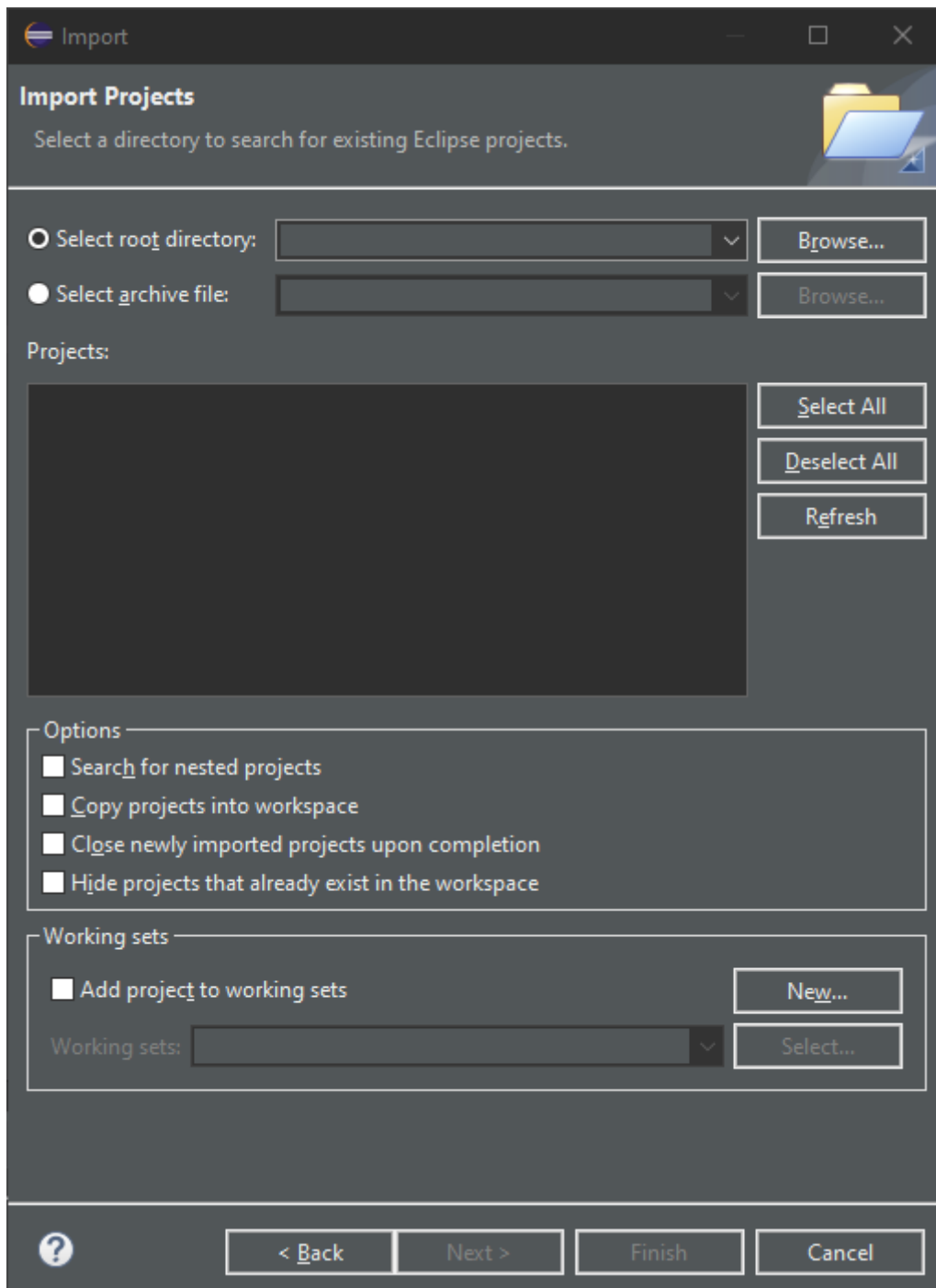


Abbildung 4: "Import"-Dialog Seite 2 in Eclipse

Als letztes muss das richtige Projekt in der „Projects“-Liste ausgewählt und bestätigt werden. Dadurch wird das Projekt in den aktuellen Workspace kopiert.

Beide Möglichkeiten sind nicht ausreichend, um das Problem dieser Bachelorarbeit zu lösen. Für die Umsetzung der ersten Möglichkeit benötigt man spezielle Kenntnisse über die Architektur von Eclipse, sowie fortgeschrittene Programmierkenntnisse, um eine eigene Erweiterung für Eclipse zu entwickeln. Außerdem können mit einer solchen Erweiterung Projekte nur erstellt und nicht aktualisiert werden. Das bedeutet, dass trotz der Erweiterung in Eclipse keine Möglichkeit besteht existierende Projekte mit der Vorlage, aus der das Projekt hervorgeht, zu vergleichen und zu aktualisieren, falls die Vorlage nach der Erstellung des Projekts verändert oder erweitert wurde. Die zweite Möglichkeit ist zwar die einfachere, aber auch die schlechtere Option, da bei dem Import Befehl lediglich ein existierendes Projekt in den aktuellen Workspace kopiert wird. Das bedeutet, dass bei dieser Methode keine richtige Projekterstellung mittels einer Projekt-Vorlage abläuft, sondern nur eine einfache Kopie erstellt wird, bei der alle Metadaten im Nachhinein manuell angepasst werden müssen.

2.2.2 CLion IDE

CLion ist eine plattformübergreifende IDE von JetBrains für C/C++. Anders als Eclipse (Abschnitt 2.4.1) und VSCode (Abschnitt 2.4.2) ist CLion kostenpflichtig mit einer kostenlosen 30-tägigen Testphase [14].

Wie Eclipse ist CLion eine projektbasierte IDE, in der alles was in CLion gemacht wird, im Kontext eines Projektes passiert. Das Projekt ist eine Organisationseinheit mit einer gewissen Verzeichnisstruktur und stellt dadurch ein komplettes Softwareprojekt dar [15]. CLion unterstützt dabei vier verschiedene Projektformate: Cmake, Gradle, JSON-Kompilierungsdatenbank (dieses Format ist im Rahmen dieser Bachelorarbeit nicht weiter relevant und wird deshalb nicht weiter beachtet), GNU Makefile [16]. Allerdings können durch CLion nur Cmake-Projekte neu erstellt werden. Projekte mit den anderen drei Projektformaten müssen extern initialisiert werden und können anschließend in CLion geöffnet und bearbeitet werden [17–20]. Um ein Projekt zu öffnen, müssen in CLion die Menüpunkte **File > Open...** ausgewählt werden. Dadurch öffnet sich ein „Pfad auswählen-Dialog“. Im Falle eines CMake-Projektes, reicht es, in diesem Dialog das Verzeichnis des Projekts auszuwählen, wenn sich in diesem Projekt eine „CMakeLists.txt“ Datei befindet.

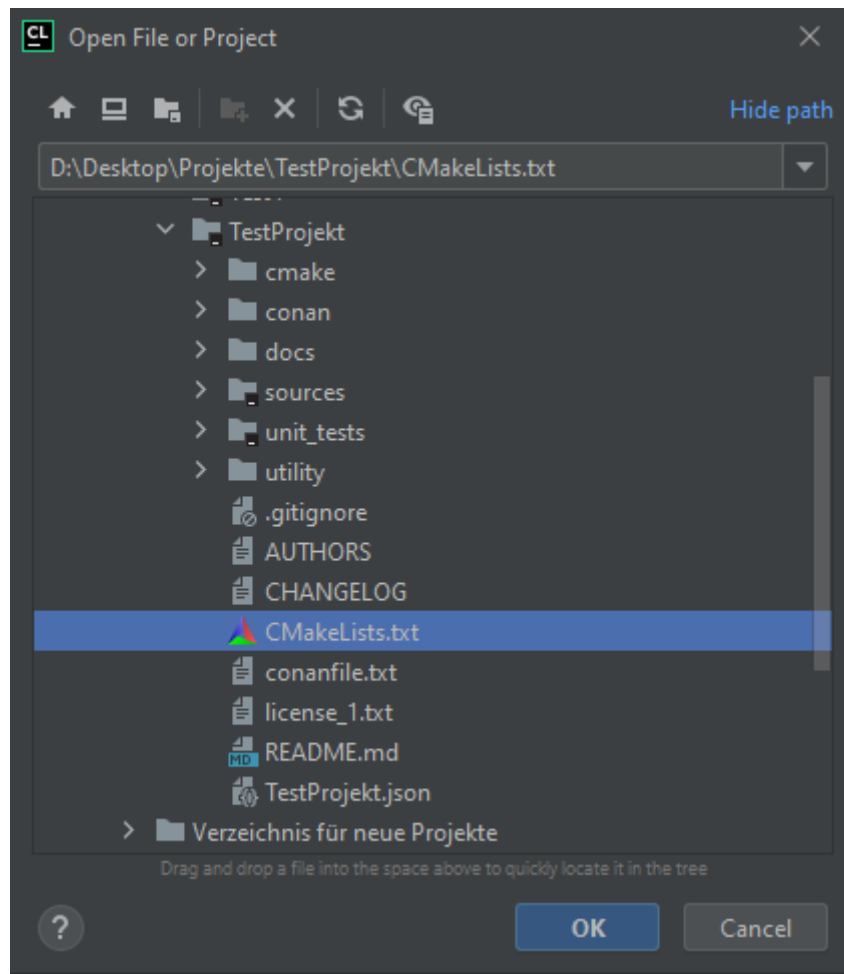


Abbildung 5: "Pfad auswählen"-Dialog in CLion

Als zweites öffnet sich ein weiterer Dialog, in dem „Trust Project“ ausgewählt werden muss, um den Inhalten des Projekts zu vertrauen.

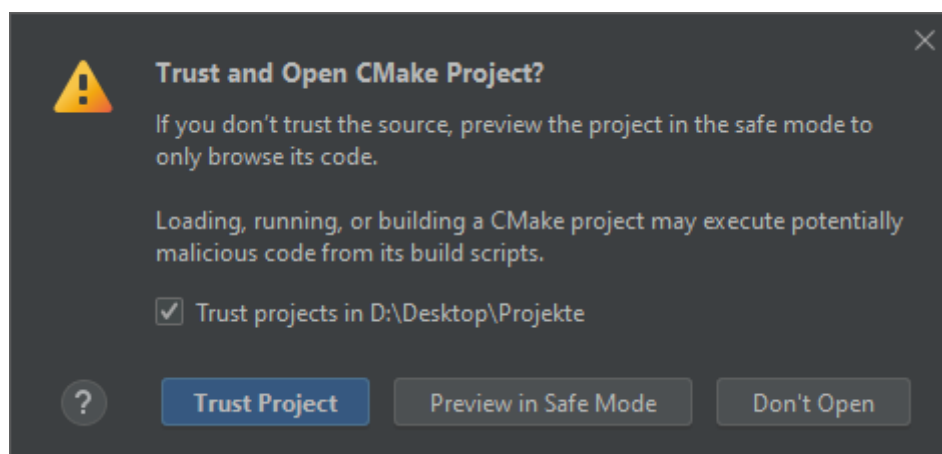


Abbildung 6: "Trust Project"-Dialog in CLion

Als drittes öffnet sich der letzte Dialog, in dem man entscheiden kann, ob das Projekt in dem aktuellen oder einem neuen Fenster geöffnet werden soll.

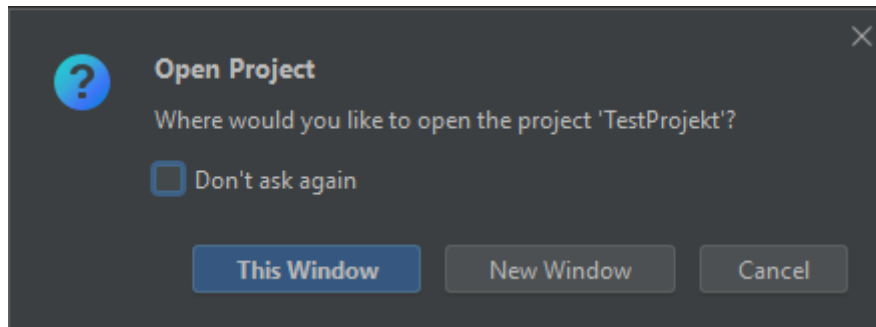


Abbildung 7: "Open Project"-Dialog in CLion

Wenn alle drei Dialoge bestätigt wurden, sollte sich das CMake-Projekt in dem ausgewählten Fenster öffnen [21].

Für die Projekte mit anderen Formaten, muss in dem „Pfad auswählen-Dialog“ der Pfad zur Build-Datei des Projektes ausgewählt werden. Für Gradle-Projekte ist das die „build.gradle“ Datei und für Make-Projekte die „Makefile“ Datei. Nachdem die entsprechende Datei ausgewählt wurde und der „Pfad auswählen-Dialog“ bestätigt wurde, erscheint ein neuer Dialog, mit dem man die Datei mit der Taste „Open as a Project“ als Projekt öffnen kann.

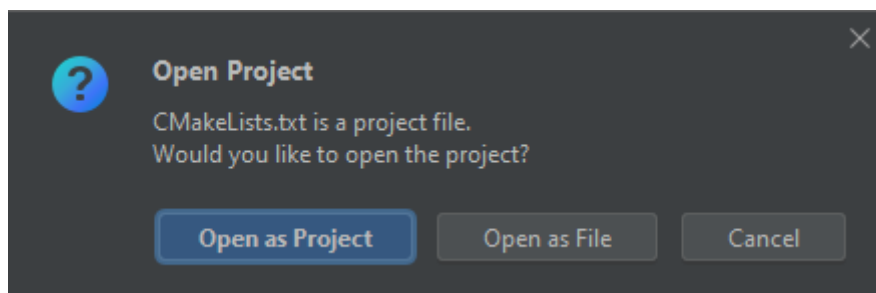


Abbildung 8: "Open as Project"-Dialog in CLion

Nach der Bestätigung dieses Dialogs folgen die gleichen zwei Dialoge wie bei dem Öffnen des CMake-Projektes, gekennzeichnet durch die Abbildungen 6 und 7, bei denen dieselben Tasten bestätigt werden müssen, um das Projekt in dem ausgewählten Fenster zu öffnen [21].

Wie im zweiten Absatz dieses Kapitels erklärt, können in CLion nur CMake-Projekte neu erstellt werden. Dazu muss in dem Menü **File > New > Project...** ausgewählt werden, um das „New Project“ -Dialog zu öffnen.

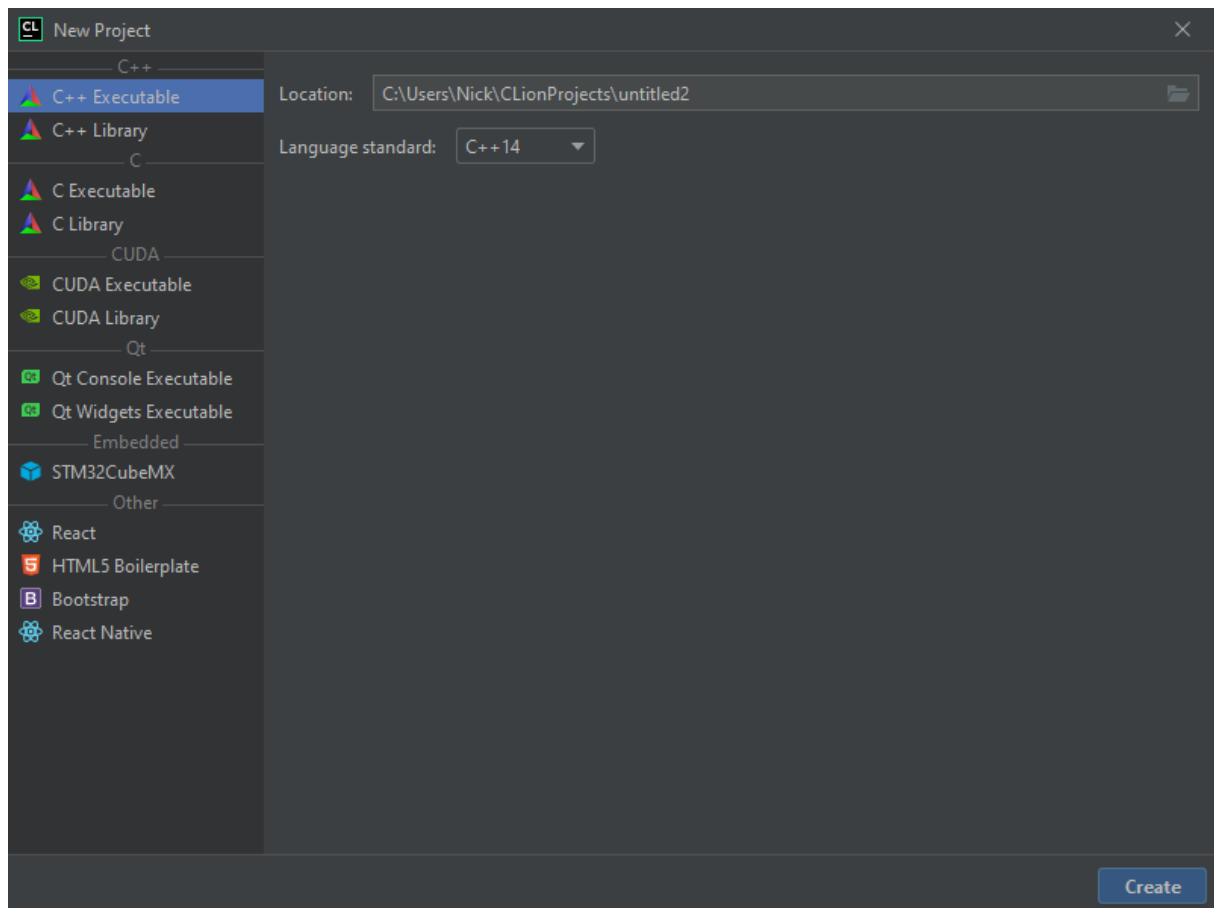


Abbildung 9: "New Project"-Dialog für Executable Projekte in CLion

In diesem Dialog muss nun die Sprache (C oder C++) und der Zieltyp (executable oder library) des Projektes ausgewählt werden. Nachdem eine Auswahl für beide Optionen getroffen wurde, muss in dem Eingabefeld auf der rechten Seite des Dialogs die Lage und der Name des Projekts eingegeben werden (In dem folgenden Beispiel wurde ein C++ Library Projekt ausgewählt). Diese Information kann entweder manuell eingegeben werden oder durch Drücken der Durchsuchen-Taste, innerhalb des Eingabefelds. Dadurch öffnet sich nämlich ein Ordner-Dialog, mit dessen Hilfe das richtige Verzeichnis ausgewählt und bestätigt werden kann.

Als nächstes muss der Sprachstandard (eng.: „Language standard“) des Projektes ausgewählt werden. Mit der „Language standard“ DropDown-Taste kann dazu eine Liste geöffnet werden, in der der gewünschte Sprachstandard ausgewählt werden muss (In diesem Beispiel wurde C++ 14 ausgewählt).

Zum Schluss muss für alle Library-Projekte ein Bibliothekstyp (eng.: „Library Type“) ausgewählt werden (Für Executable-Projekte existiert diese Auswahl nicht). Mit der

DropDown-Taste unter „Library Type“ kann hierfür zwischen „static“ und „shared“ Bibliothekstypen ausgewählt werden (In diesem Beispiel wurde shared ausgewählt).

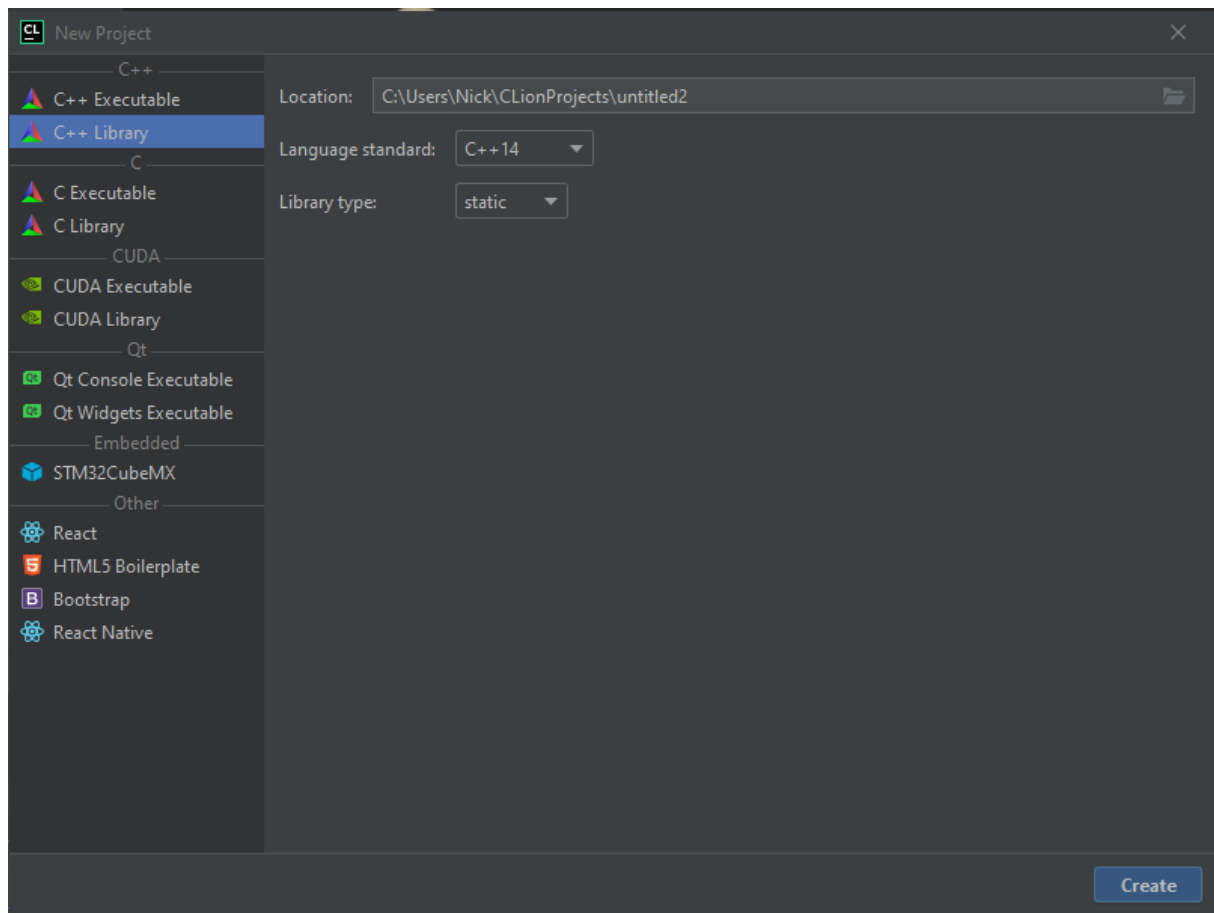


Abbildung 10: "New Project"-Dialog für Library Projekte in CLion

Nachdem alle Konfigurationen durchgeführt wurden, muss die Create-Taste gedrückt werden, um ein neues CMake-Projekt zu erstellen. Dieses Projekt wird standardmäßig auf Basis eines intern gespeicherten Templates generiert. Das bedeutet, das Projekt besteht nach dessen Erstellung aus einer bestimmten Verzeichnisstruktur mit vordefinierten Ordnern und Dateien. Eine dieser Dateien ist beispielsweise die Root-CMakeLists.txt Datei, welche, basierend auf den bereitgestellten Informationen (Projekt Name, Sprachstandard, Bibliothekstyp), automatisch generiert wird [22].

Ähnlich wie Eclipse, verwendet CLion eine intern gespeicherte Projekt-Vorlage, um neue Projekte zu erstellen. Verglichen mit Eclipse, gibt es in CLion jedoch keine Möglichkeit die interne Projekt-Vorlage zu verändern oder eine neue Projekt-Vorlagen hinzuzufügen. Allerdings können in CLion selbsterstellte Datei-Vorlagen integriert werden. Dafür muss zuerst eine Datei in dem Editor geöffnet werden. Anschließend

muss in dem Hauptmenü **File > Save File as Template** ausgewählt werden, um ein „Save File as Template“-Dialog zu öffnen.

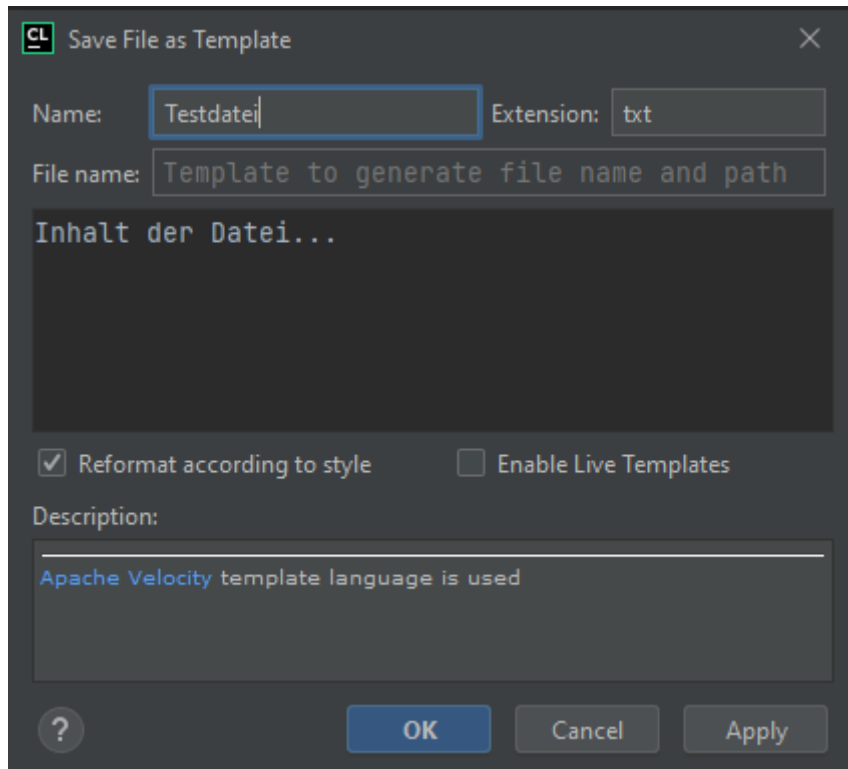


Abbildung 11: "Save File as Template"-Dialog in CLion

In diesem Dialog kann daraufhin ein neuer Template Name für das Template vergeben werden. Zum Schluss müssen die Eingaben mit der „OK“-Taste bestätigt werden, um die Datei als neue Vorlage zu speichern.

Um eine Datei aus einer Vorlage zu erstellen, müssen mit der Kombination **Strg+Alt+S** die Einstellungen der IDE geöffnet werden. Darin muss zu **Editor > File and Code Templates** navigiert werden, um die Liste aller Datei Vorlagen zu öffnen.

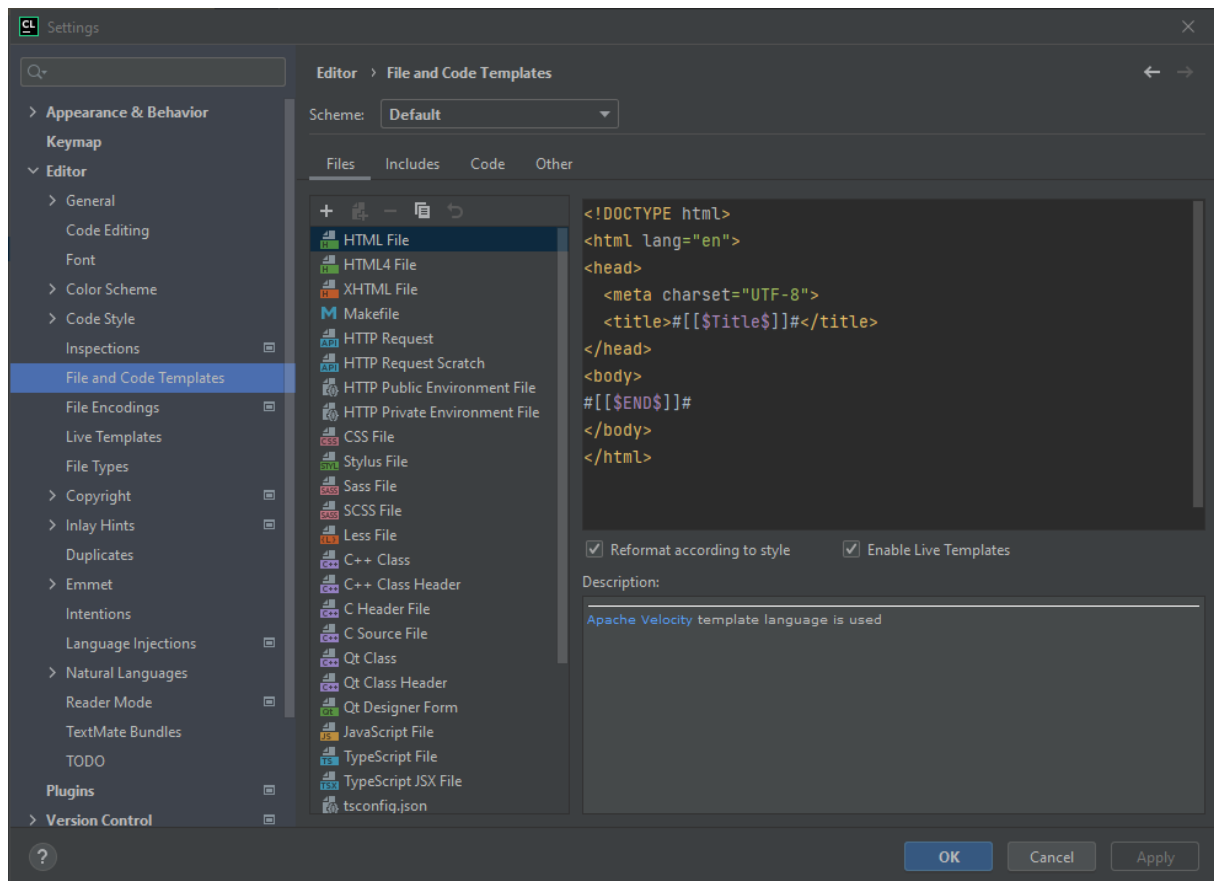



Abbildung 12: Liste aller Dateivorlagen in CLion

Nachdem eine Vorlage in der Liste ausgewählt wurde, muss anschließend das  - Symbol angeklickt werden. In dem neu erschienen Bereich kann dann der Name, die Dateierweiterung und wenn nötig der Textkörper der Vorlage verändert werden.

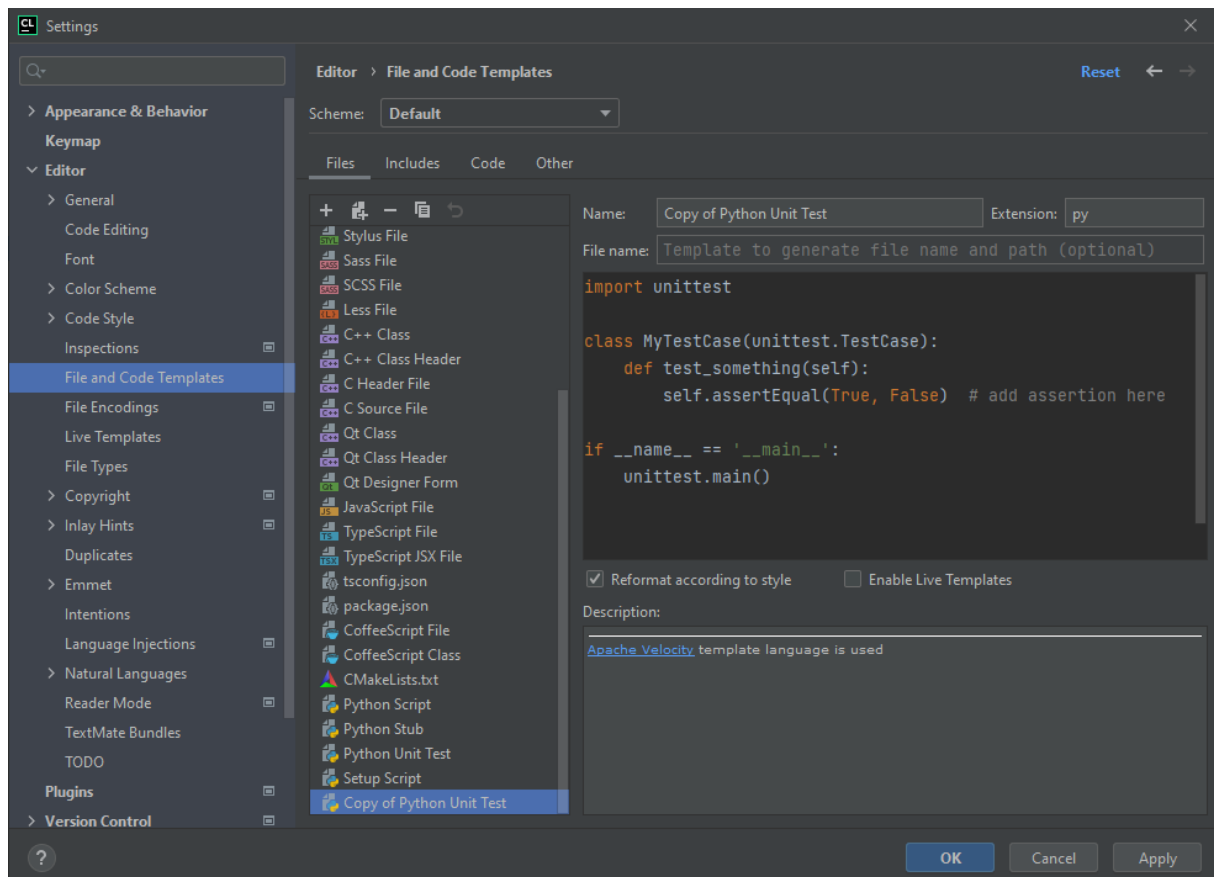


Abbildung 13: Kopie einer Dateivorlage in CLion

Zum Schluss müssen die Einstellungen mit der „OK“-Taste bestätigt werden, um die Einstellungen zu schließen und die neue Datei zu erstellen [23]. Die selbst erstellten Vorlagen können zudem auch Platzhalter beinhalten, die bei der Erstellung der realen Datei mit projektspezifischen Daten ersetzt werden. CLion bietet dafür eine Reihe von vordefinierten Platzhaltern an. Allerdings können auch selbst erstellte Platzhalter verwendet werden. Dazu müssen in den Template-Dateien das Schlüsselwort „#set“ benutzt werden [24].

Wie auch Eclipse, bietet CLion keinen ausreichenden Funktionsumfang, der das Problem dieser Abschlussarbeit hinreichend lösen würde. Je nach Projektformat wird jedes neue Projekt in CLion nämlich, mittels einer intern gespeicherten Projekt-Vorlage erstellt, die nicht veränderbar oder erweiterbar ist. Zudem gibt es keine Möglichkeit weitere Projekt-Vorlagen hinzuzufügen. Dadurch ist ausgeschlossen, dass das Problem der Projektaktualisierung mit CLion behoben werden kann. Auch wenn es eine Möglichkeit gäbe existierende Projekte mit der internen Vorlage zu vergleichen und diese bei unterschieden zu aktualisieren, würde sie nicht helfen, da man keinen Einfluss auf die Änderungen der internen Vorlage hat. Die einzige Option mit eigenen

Vorlagen zu arbeiten, ist die erklärte Dateierstellung mittels Datei-Vorlagen. Dadurch können zwar alle benötigten Dateien, die nicht in der internen Projekt-Vorlage vorhanden sind, mittels einer Vorlage erstellt werden, doch diese Erstellungen würden dabei einzeln nacheinander passieren, was zeitaufwändig ist. Zudem gibt es auch bei den Dateien keine Möglichkeit diese zu aktualisieren.

2.2.3 Visual Studio Code

Visual Studio Code (abgekürzt VSCode) ist ein kostenloser und open source (github.com/microsoft/vscode) basierter Code-Editor von Microsoft. Es ist plattformübergreifend auf den Betriebssystemen Windows, Linux und macOS verfügbar [25].

Anders als Eclipse und CLion ist VSCode keine richtige IDE, sondern ein Quellcode-Editor, dessen Hauptzweck es ist das Schreiben von Quellcode zu vereinfachen. Der Hauptunterschied zu richtigen IDEs liegt darin, dass in VSCode viele Tools, die für die Entwicklung von Softwareprojekten wichtig sind, standardmäßig nicht integriert sind. Nichtsdestotrotz können diese Tools in dem Tab „Erweiterungen“ nachträglich in VSCode installiert werden. Zusätzlich kann VSCode auch mit weiteren Programmiersprachen erweitert werden [26].

Wie in Abschnitt 2.4 erwähnt, ist die wichtigste Funktionalität die, der Erstellung und Verwaltung ganzer Projekte mit mehreren Dateien. Um ein neues Projekt in VSCode anzulegen, muss unter **Datei > Ordner öffnen...** des Hauptmenüs, ein Verzeichnis ausgewählt werden, in welches ein neues Projekt angelegt werden soll. Nachdem ein Verzeichnis ausgewählt und geöffnet wurde, kann darin, Mithilfe des VSCode Explorers, ein neuer Ordner angelegt werden. Dieser Ordner repräsentiert das Softwareprojekt.

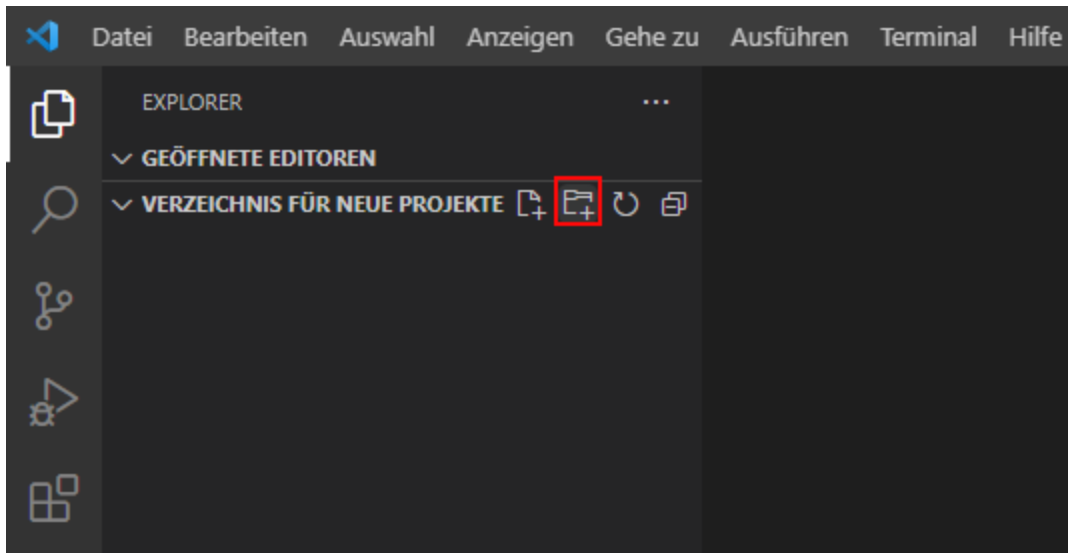


Abbildung 14: Explorer in VSCode

Diesem Ordner können im Nachhinein auf die gleiche Weise weitere Unterorder und Dateien, die für das Projekt notwendig sind, hinzugefügt werden [27].

Diese Vorgehensweise sind jedoch nicht dafür geeignet, wenn ein Softwareprojekt (wie in der Problemstellung (Abschnitt 1.1) erklärt) nach einer bestimmten Vorlage erstellt werden soll. Mit der Erweiterung „Project Templates“ von cantonios (marketplace.visualstudio.com/project-templates) kann dieses Problem allerdings gelöst werden. Nach der Installation dieser Erweiterung ist es zum einen möglich ein bestimmtes Projekt als ein Template-Projekt zu speichern. Dazu muss als erstes ein zuvor initialisiertes Projekt in VSCode geöffnet werden. Nachdem das Projekt geöffnet wurde, muss im Explorer ein Kontextmenü geöffnet werden, indem man mit der rechten Maustaste auf das Projekt klickt. Wenn die Erweiterung richtig installiert wurde, sollte nun „Save Project as Template“ als Auswahl in dem Kontextmenü erscheinen.

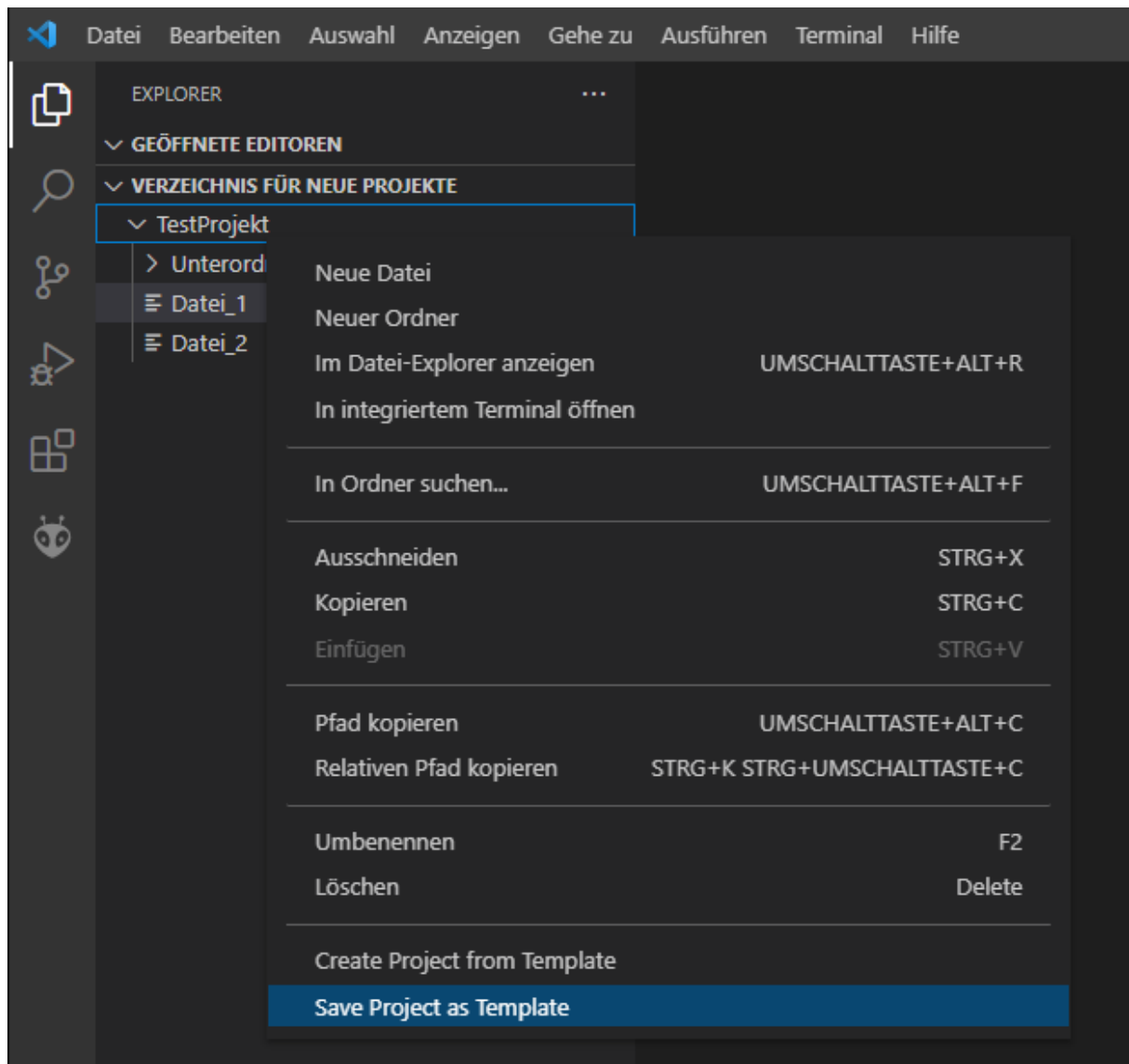


Abbildung 15: Projekt als Template speichern in VSCode

Nachdem dieser Menüpunkt ausgewählt wurde, öffnet sich die Befehlspalette von VSCode, in der man einen beliebigen Namen für dieses Template-Projekt vergeben kann.

Zum Schluss muss der eingegebene Name mit der Eingabetaste auf der Tastatur bestätigt werden, um das Projekt automatisch als Template-Projekt zu speichern. Mit den Standardeinstellungen von „Project Templates“ werden alle Template-Projekte unter folgenden Pfaden gespeichert:

```
$HOME/.config/Code/User/ProjectTemplates      # Linux
$HOME/Library/Application Support/Code/User/ProjectTemplates # macOS
%APPDATA%\Code\User\ProjectTemplates          # Windows
```

Abbildung 16: Standardspeicherorte der Projekte der VSCode Extension

Diese können aber in den Benutzereinstellungen unter **Verwalten > Einstellungen > VSCode Project Template Configuration > In „settings.json“ bearbeiten** geändert werden, indem man folgende Zeile addiert:

```
"projectTemplates.templatesDirectory": "path/to/my/templates"
```

Abbildung 17: Änderung des Speicherortes für die Projekte der VSCode Extension

Zum anderen kann die Extension dazu benutzt werden, um ein Projekt aus einem gespeicherten Template-Projekt zu erstellen. Dazu muss in VSCode ein Verzeichnis geöffnet werden, in dem ein leerer Ordner liegt. In diesem leeren Ordner wird das neue Projekt erstellt. Wie in dem Abschnitt zuvor muss man dafür mit der rechten Maustaste auf den leeren Ordner klicken, damit sich ein Kontextmenü öffnet. In dem Menü sollte sich der Menüpunkt „Create Project from Template“ befinden.

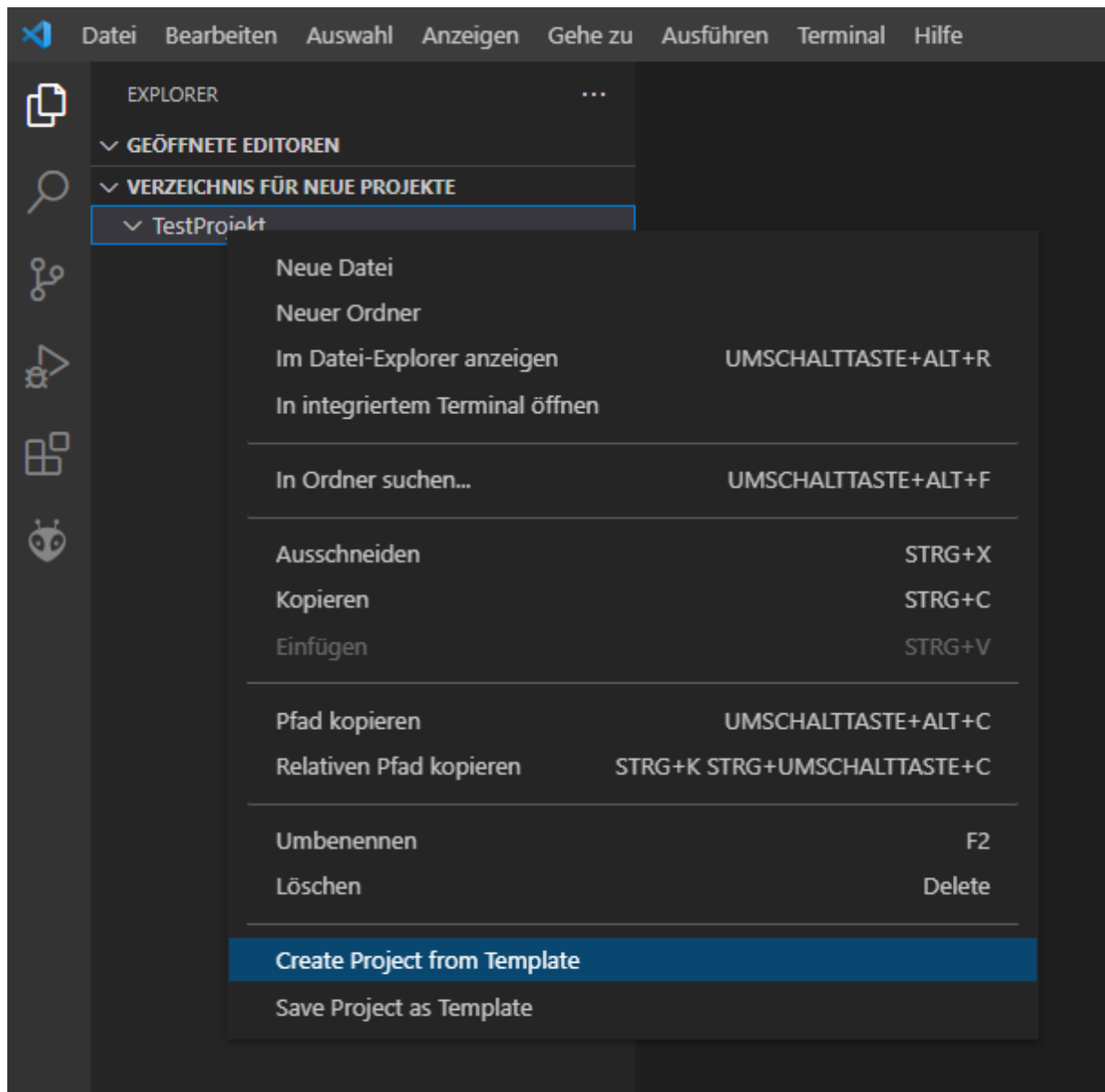


Abbildung 18: Projekt von einem Template erstellen in VSCode

Mit der Auswahl des Menüpunktes öffnet sich die Befehlspalette in VSCode mit einer Liste aller gespeicherten Template-Projekte. In dieser Liste muss das gewünschte Template-Projekt ausgewählt werden. Dadurch wird der Inhalt (alle Unterordner und Dateien) des ausgewählten Template-Projektes in den leeren Ordner kopiert. Damit besitzt das neue Projekt am Ende des Vorgangs die gleiche Verzeichnisstruktur, wie das Template-Projekt.

Darüber hinaus können in den Dateien eines Template-Projektes auch variable Platzhalter eingesetzt werden. Die Schreibweise sieht wie folgt aus:

```
Author: #{author}  
Title: #{title}
```

Abbildung 19: Art der Integration für Platzhalter der VSCode Extension

Wenn eine Datei aus einem Template mit Platzhaltern erstellt wird, wird man bei der Erstellung zu einer Eingabe eines Wertes aufgefordert. Platzhalter können außerdem auch in Dateinamen verwendet werden [28].

Auch die Extension „Project Templates“ ist für die Lösung des Problems dieser Arbeit nicht ausreichend. Es ist mit ihr zwar möglich ein neues Projekt anhand einer Projekt-Vorlage zu erstellen, allerdings sind die Platzhalter, die man in die Datei-Vorlagen einbinden kann, nur auf die Ersetzung durch Zeichenketten beschränkt. Es ist nicht möglich, dass Platzhalter eine Liste von vordefinierten Werten übergeben werden kann, aus der bei der Projekterstellung ein Wert ausgewählt werden kann. Des Weiteren können Platzhalter keine booleschen Werte annehmen, aus denen hervorgeht, ob manche Dateien der Projekt-Vorlage, nicht in dem erstellten Projekt vorhanden sein sollen, diese also nicht miterstellt werden. Außerdem fehlt auch bei dieser Extension die Funktion ein Projekt mit deren Vorlage zu vergleichen und zu aktualisieren, falls die Vorlage nach der Erstellung des Projektes verändert / erweitert wurde.

2.3 Zusammenfassung

In diesem Kapitel wird erklärt, wieso die vorgestellten Lösungen dieses Kapitels nicht ausreichen, um die Probleme dieser Bachelorarbeit zu lösen.

Mit Versionsverwaltungen kann nur eine Kopie einer Projekt-Schablone erstellt werden. Um daraus ein neues Projekt zu erstellen, muss die Kopie anschließend manuell bearbeitet werden. Diese Vorgehensweise ist zeitaufwändig und fehleranfällig.

In Eclipse muss zuerst ein neuer Projekt-Wizard programmiert werden, um ein Projekt aus einer neuen Projekt-Schablone erstellen zu können. Dazu wird zu einem Vorwissen in der Programmierung und zum anderen Vorwissen über die Architektur von Eclipse benötigt.

CLion bietet keine Möglichkeit an, eine neue Projekt-Schablone einzubinden. Allerdings können in CLion Dateien aus eigenen Datei-Vorlagen erstellt werden. Ein Projekt aus einzelnen Dateien aufzubauen ist allerdings mühsam und fehleranfällig.

VSCoide mit der installierten Extension „Project Templates“ ist die beste vorgestellte Lösung. Allerdings ist diese Lösung auch nicht ausreichend, da für die Platzhalter, die den Dateien hinzugefügt werden, könne, nur Zeichenketten übergeben werden können.

Das wichtigste ist jedoch, dass keine der vorgestellten Lösungen eine Möglichkeit beinhaltet, mit der bestehende Projekte mit der Projekt-Schablone, aus der diese entstanden sind, verglichen werden und bei unterschieden aktualisiert werden können.

3. Lösungsansatz

Um das beschriebene Problem in Kapitel 1 zu lösen, wurde im Rahmen dieser Thesis ein spezielles DevOps Projekt-Konfigurator Werkzeug, mit zwei Hauptfunktionen entwickelt. Die Hauptfunktionen des Tools erlauben es Projekte, mithilfe einer grafischen Benutzeroberfläche, zu erstellen und zu aktualisieren. Die Projekterstellung sowie -aktualisierung erfolgt dabei auf Basis eines intern gespeicherten Template-Projektes. Die Verzeichnisstruktur des Templates besteht aus bestimmten Unterordnen und Dateien, die für jedes neue Projekt erforderlich sind. Der Inhalt der Dateien besteht meist aus fixen Daten, in manchen befinden sich allerdings Platzhalter, die bei der Erstellung ersetzt werden müssen.

Um ein neues Projekt mit dem Projekt-Konfigurator zu erstellen, muss ein spezielles Formular in der GUI ausgefüllt werden. In das Formular müssen folgende Werte eingegeben werden:

- Der lange Projektname – einzugeben in ein Texteingabefeld
- Der kurze Projektname – einzugeben in ein Texteingabefeld
- Die Projektbeschreibung – einzugeben in ein Texteingabefeld
- Der Modultyp – durch Auswahl eines Wertes in einer DropDown-Liste
- Der Modulname – einzugeben in ein Texteingabefeld.
- Der Lizenztyp – durch Auswahl eines Wertes in einer DropDown-Liste
- Die Autoren – einzugeben in eine Liste
- Eine Auswahl an vordefinierten Konfigurationsdateien – durch markieren der entsprechenden Checkboxes

Die Unterteilung des Projektnamens in einen kurzen und einen langen Namen ist aus verschiedenen Gründen notwendig. Der lange Projektname ist der vollständige Name des Projektes und wird deswegen nach der Erstellung des Projektes als Titel in verschiedenen Dateien benutzt. Allerdings ist die Länge des langen Projektnamens auf 255 Zeichen limitiert. In das entsprechende Texteingabefeld des Formulars können aus diesem Grund auch nicht mehr Zeichen eingegeben werden.

Der kurze Projektname wird dagegen einerseits als Name für den Projektordner und andererseits als Name für die Bereitstellungspakete des Softwareprojektes verwendet. Aus diesem Grund muss dieser Name URL-Konform sein, was bedeutet, dass der

dieser Name keine Zeichen beinhalten darf, die in einer URL nicht enthalten sein dürfen.

Nachdem das Formular mit den notwendigen Informationen ausgefüllt und bestätigt wurde, erstellt das Tool, mit Hilfe der eingegebenen Informationen und des Templates, automatisch ein neues Projekt. Die eingegebenen Informationen werden bei dem Erstellungsvorgang dafür verwendet, um die Platzhalter in den Template-Dateien, mit den entsprechenden Metadaten des Projekts zu ersetzen.

Die Aktualisierung bestehender Projekte ist die wichtigste Funktion des Projekt-Konfigurators. Mithilfe der GUI muss dafür ein bereits erstelltes Projekt im Dateisystem ausgewählt werden. Nachdem ein Projekt ausgewählt wurde, überprüft das Tool die Aktualität des Projekts, indem es das Projekt mit dem entsprechenden Template vergleicht. Wenn das Template, nach der Erstellung des ausgewählten Projektes verändert wurde, werden auf der GUI alle Unterordner und Dateien des Projekts aufgelistet, die aktualisiert werden müssen. Zusätzlich dazu beinhaltet die Auflistung auch alle Unterordner und Dateien, die dem Template neu hinzugefügt wurden und nicht in dem realen Projekt vorhanden sind. Alle Unterordner und Dateien, die sich in der Liste befinden können, anschließend automatisch durch einen Tastendruck auf der GUI aktualisiert werden.

4. Softwareentwurf

Für die Umsetzung des Lösungsansatzes, wurde eine Software mit einer grafischen Benutzeroberfläche benötigt. Die GUI ist dabei die Schnittstelle, über die die Entwickler mit dem Programmcode der Software und dem Computer interagierten. Dafür mussten als erstes die notwendigen Funktionen bestimmt werden, die zur Lösung der Problemstellung, wie in Kapitel 1.1 erläutert, unabdingbar sind. Demnach mussten die GUI am Ende dieser Bachelorarbeit folgende drei Hauptfunktionen ausführen können:

- Erstellung von Projekten auf Basis einer Vorlage
- Aktualisierung von Projekten auf Basis einer Vorlage
- Suchungsmechanismus von bestehenden Projekten

Mit letzterer können existierende Projekte gesucht und ausgewählt werden, um diese daraufhin aktualisieren zu können. Zusätzlich zu den Hauptfunktionen wurden noch zwei weitere Zusatzfunktionalitäten bestimmt, die ebenfalls in den Funktionsumfang der GUI integriert werden sollten:

- Erweiterung der Projekt Module
- Ausführung von Dienstprogrammen eines Projektes

In diesem Kapitel wird zuerst die verwendete Programmiersprache und das Framework für die Entwicklung der GUI begründet. Anschließend wird erklärt, wie das Programm aus der Sicht der Benutzerfreundlichkeit aussieht, indem die grafische Benutzeroberfläche mittels Skizzen analysiert wird, und wie die Software des Programms strukturiert ist, indem die Software-Architektur besprochen wird.

4.1 Die Programmiersprache und das GUI Framework

Für die Implementierung der Software wurde eine Programmiersprache und ein GUI-Framework benötigt. Im Folgenden wird erklärt, welche Sprache für die Implementierung gewählt wurde und auf Basis dessen, die Entscheidung des GUI-Frameworks begründet.

Die Software wurde mit der Programmiersprache Python geschrieben. Diese Auswahl wurde aus drei Gründen getroffen:

1. Python ist eine einfach zu benutzen
2. Python funktioniert auf jedem Betriebssystem gleich

3. Python verfügt über eine hervorragende Dokumentation und Bibliotheksunterstützung

Für die Umsetzung der GUI wurde zusätzlich ein GUI-Toolkit benötigt. Zur Auswahl standen PyQt, ein plattformübergreifendes GUI-Toolkit des Frameworks Qt, und Tkinter, ein GUI-Toolkit, welches ausschließlich für Python entwickelt wurde. Nach einem umfangreichen Vergleich dieser beiden Toolkits, fiel die Entscheidung auf PyQt. Diese Entscheidung basiert auf mehreren Gründen:

1. PyQt mehrere Widgets an als Tkinter, wodurch die Auswahl der Widgets größer und flexibler ist.
2. PyQt ist das am häufigsten verwendeten UI-Framework für Python, wodurch es, verglichen mit Tkinter, mehr Dokumentationen und Hilfestellungen im Internet verfügbar sind.
3. Die Programmierung mit PyQt basiert auf dem Konzept von Signalen und Slots, wodurch die Kommunikation zwischen Objekten bzw. Widgets hergestellt wird. Dadurch bietet PyQt eine hohe Flexibilität in der Programmierung.
4. PyQts Standarddesign der Widgets ist um einiges moderner als die der Tkinter Widgets. Darüber hinaus können die Widgets in PyQt mithilfe von CSS umgestaltet werden. Diese Funktion bietet Tkinter nicht.

Zusammengefasst bietet PyQt eine größere Auswahl an Widgets, die mit dem Einsatz von CSS umgestaltet werden können, sowie eine Vielzahl an Dokumentationen. Außerdem ist die Programmierung der GUI mit PyQt, aufgrund des Konzepts von Signalen und Slots um einiges flexibler. Aus diesen Gründen wurde PyQt als GUI Framework ausgewählt.

4.2 GUI Skizzen

Entsprechend den Haupt- und Zusatzfunktionen musste die Benutzeroberfläche aus fünf verschiedenen Fenstern (eng.: Screens) aufgebaut sein. Jeder Screen sollte dabei für eine bestimmte Haupt- oder Nebenfunktionalität zuständig sein. Um die Funktionen ausführen zu können, musste jeder Screen aus einer Reihe von bestimmten Steuerelementen, die als Widgets bezeichnet werden, aufgebaut sein. Ein Widget ist dabei ein Interaktionselement der grafischen Benutzeroberfläche, wie bspw.: Ein Button, ein Textfeld, eine Dropdown-Liste oder ein Listenfeld. Durch die Widgets können Entwickler mit der GUI interagieren und unter anderem bestimmte Prozesse

starten, Werte eingeben, Werte auswählen oder Daten konfigurieren. Die Widgets jedes Screens wurden hinsichtlich ihrer auszuführenden Funktion ausgewählt. Diese Auswahl folgte an zweiter Stelle, nach der Bestimmung aller GUI-Funktionalitäten. Das Ergebnis war eine Liste von Widgets für jeden Screen. Mit den ausgewählten Widgets wurden anschließend verschiedene Skizzen bzw. Sketches jedes Screens gezeichnet, um den Aufbau jedes Fensters zu visualisieren. Die Skizzen dienten außerdem als Vorlage für die Implementierung. Im weiteren Verlauf dieses Kapitels werden folgende Skizze gezeigt und erklärt:

- Das Hauptmenü, das für die Projektsuchfunktionalität zuständig ist und es dem Benutzer ermöglicht, die Fenster der GUI zu wechseln.
- Das Fenster für die Projekterstellung, mit dem anhand eingegebener Informationen ein neues Projekt erstellt werden kann.
- Das Fenster für die Projektaktualisierung, mit dem ein ausgewähltes Projekt aktualisiert werden kann.
- Das Fenster für die Erweiterung der Projektmodule, mit dem anhand eingegebener Informationen ein neues Projektmodul erstellt werden kann.
- Das Fenster für die Ausführung der Dienstprogramme, mit dem die Dienstprogramme eines Projektes auf Basis eingegebener Befehle ausgeführt werden können

4.2.1 Das Hauptmenü

Das Hauptmenü, welches in der Abbildung 20 gezeigt wird, ist der Startbildschirm der GUI. Durch das Hauptmenü lassen sich bestehende Projekte suchen und finden und es kann von hier auf alle anderen Fenster der GUI gewechselt werden.

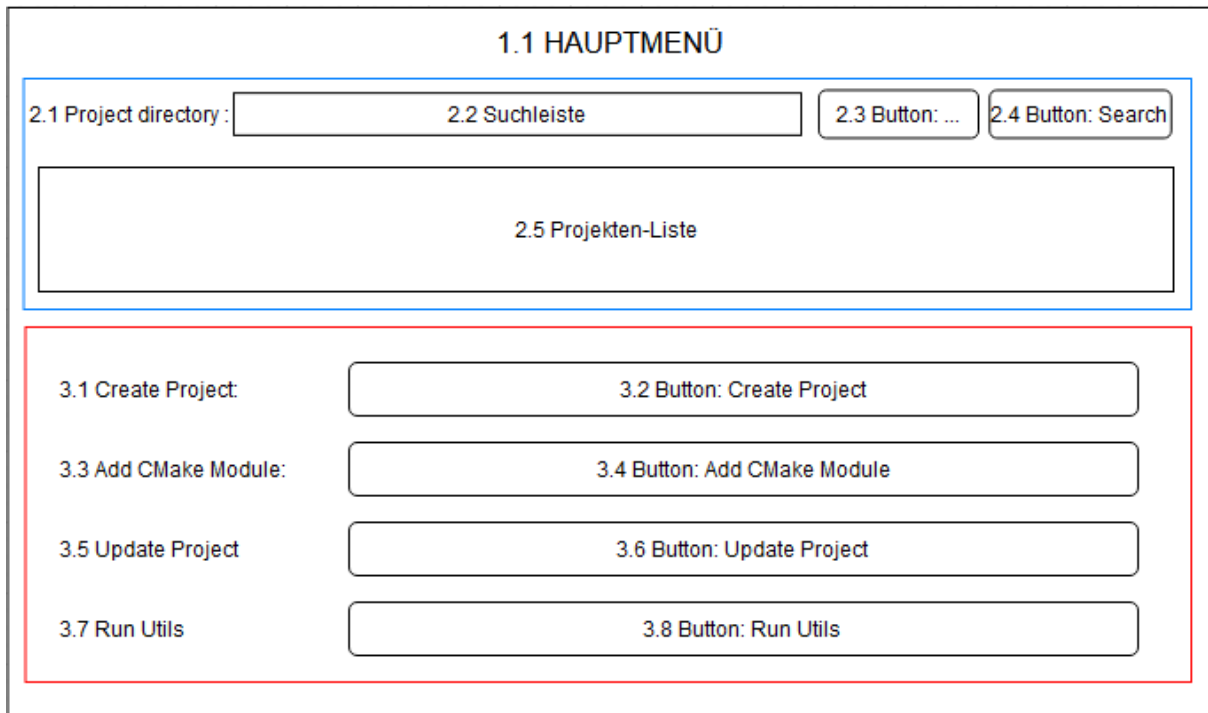


Abbildung 20: Skizze für den Screen des Hauptmenüs

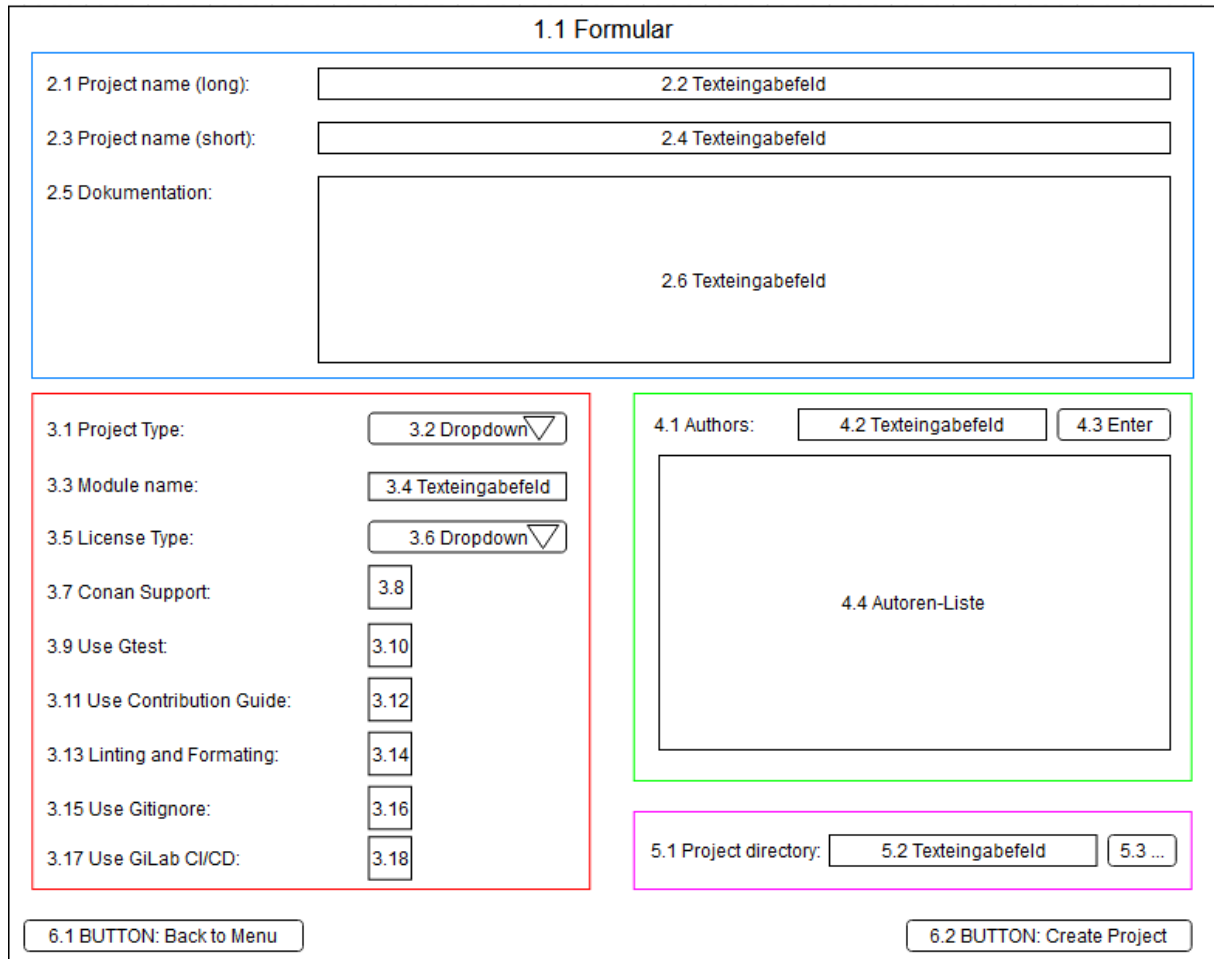
In der Abbildung 20 ist zusehen, dass das Hauptmenü aus zwei verschiedenen Teilen, in den Farben blau und rot gekennzeichnet, aufgebaut ist. Jeder Teil ist dabei für eine spezielle Funktion zuständig.

Der erste Teil ist für den Suchungsmechanismus von bestehen Projekten verantwortlich. In der Suchleiste, gekennzeichnet durch die Suchleiste 2.2 in der Abbildung 20, kann dafür der Pfad zu dem Root-Ordner eingegeben werden, in dem ein oder mehrere Projekte liegen. Um den Pfad nicht manuell eingeben zu müssen, kann alternativ dazu der „...“-Button, gekennzeichnet durch den Button 2.3 in der Abbildung 20, benutzt werden, um einen Ordner-Dialog zu öffnen, mit dem der Root-Ordner ausgewählt werden kann. Der Pfad des Root-Ordners wird daraufhin automatisch in die Suchleiste übertragen. Anschließend kann durch das Drücken des „Search“-Buttons, gekennzeichnet durch den Button 2.4 in der Abbildung 20, jedes Projekt, dass in dem Root-Ordner liegt, in der Projekten-Liste, gekennzeichnet durch die Liste 2.5 in der Abbildung 20, aufgelistet werden.

Der zweite Teil des Hauptmenüs ist für das Wechseln auf die anderen GUI-Fenster zuständig. Jeder der vier Button, gekennzeichnet durch die Buttons 3.2, 3.4, 3.6, 3.8 der Abbildung 20, wechselt dabei auf ein anderes Fenster. Der „Create Project“-Button, gekennzeichnet durch den Button 3.2 in der Abbildung 20, wechselt auf das Fenster für die Projekterstellung, dass später in Kapitel 5.2.2 erklärt wird. Der „Add CMake Module“-Button, gekennzeichnet durch den Button 3.4 in der Abbildung 20, wechselt auf das Fenster für die Erweiterung der Projektmodule, dass später in Kapitel 5.2.4 erklärt wird. Der „Update Project“-Button, gekennzeichnet durch den Button 3.6 in der Abbildung 20, wechselt auf das Fenster für die Projektaktualisierung, dass später in Kapitel 5.2.3 erklärt wird. Der „Run Utils“-Button, gekennzeichnet durch den Button 3.8 in der Abbildung 20, wechselt auf das Fenster für die Ausführung der Dienstprogramme, dass später in Kapitel 5.2.5 erklärt wird. Die Buttons 3.4, 3.6, 3.8 sind allerdings standardmäßig inaktiv, was bedeutet, dass diese nicht gedrückt werden können. Erst wenn ein Projekt in der Projekten-Liste, gekennzeichnet durch die Liste 2.5 in der Abbildung 20, ausgewählt wird, werden diese drei Buttons aktiv und wieder klickbar. Der Grund dafür ist, dass für die Weiterarbeit in den Fenstern, auf die mit Hilfe der drei Buttons gewechselt werden kann, ein ausgewähltes Projekt benötigt wird.

4.2.2 Fenster für die Projekterstellung

Auf diesem Fenster werden alle Informationen gesammelt, die zur Erstellung eines neuen Projekts erforderlich sind. Es funktioniert ähnlich wie ein Formular. Das Fenster wird dabei in der GUI angezeigt, wenn der „Create Project“-Button, gekennzeichnet durch den Button 3.2 in der Abbildung 20, des Hauptmenüs angeklickt wird.



1.1 Formular

2.1 Project name (long): 2.2 Texteingabefeld

2.3 Project name (short): 2.4 Texteingabefeld

2.5 Dokumentation: 2.6 Texteingabefeld

3.1 Project Type: 3.2 Dropdown

3.3 Module name: 3.4 Texteingabefeld

3.5 License Type: 3.6 Dropdown

3.7 Conan Support: 3.8

3.9 Use Gtest: 3.10

3.11 Use Contribution Guide: 3.12

3.13 Linting and Formatting: 3.14

3.15 Use Gitignore: 3.16

3.17 Use GiLab CI/CD: 3.18

4.1 Authors: 4.2 Texteingabefeld 4.3 Enter

4.4 Autoren-Liste

5.1 Project directory: 5.2 Texteingabefeld 5.3 ...

6.1 BUTTON: Back to Menu 6.2 BUTTON: Create Project

Abbildung 21: Skizze für den Screen der Projekterstellung

In der Abbildung 21 ist zu sehen, dass das Fenster aus vier verschiedenen Teilen, gekennzeichnet in blau, rot, grün und pink, aufgebaut ist.

Der erste Teil ist zuständig für den Projektnamen und die Projektbeschreibung. Der Projektnamen ist dabei in eine lange und eine kurze Version aufgeteilt. In dem ersten Texteingabefeld, gekennzeichnet durch das Eingabefeld 2.2 in der Abbildung 21, muss der lange Projektnamen eingegeben werden und in dem zweiten Texteingabefeld, gekennzeichnet durch das Eingabefeld 2.4 in der Abbildung 21, der kurze Projektnamen. In dem letzten Texteingabefeld des Teils, gekennzeichnet durch das

Eingabefeld 2.6 in der Abbildung 21, muss die Projektbeschreibung eingegeben werden.

Der zweite Teil ist zuständig für das Projektmodul, die verwendeten Lizenzen und die externen verwendeten Tools. Für das Projektmodul muss zuerst ein Projekttyp ausgewählt werden. Dieser ist über die erste Dropdown-Liste, gekennzeichnet durch die DropDown-Liste 3.2 in der Abbildung 21, auszuwählen. Anschließend muss ein Modulname vergeben werden. Dieser kann mithilfe des Texteingabefeldes, gekennzeichnet durch das Eingabefeld 3.4 in der Abbildung 21, eingegeben werden. Als nächstes muss ein Lizenztyp über die zweite Dropdown-Liste, gekennzeichnet durch die DropDown-Liste 3.4 in der Abbildung 21, ausgewählt werden. Als letztes müssen alle vordefinierten Konfigurationsdateien für die CI/CD-Tools ausgewählt werden, die für das Projekt verwendet werden. Zur Auswahl stehen sechs Tool, die durch die Checkboxes, gekennzeichnet durch die Checkboxes 3.6, 3.8, 3.10, 3.12, 3.14, 3.16 in der Abbildung 21, ausgewählt werden können.

Der dritte Teil ist für die Autoren des Projektes zuständig. In dem Texteingabefeld, gekennzeichnet durch das Eingabefeld 4.2 in der Abbildung 21, kann dafür der Name eines Autors eingegeben werden. Mit dem Drücken des „Enter“-Buttons, gekennzeichnet durch den Button 4.3 in der Abbildung 21, wird der eingegebene Name bestätigt und in die Autoren-Liste, gekennzeichnet durch die Liste 4.4 in der Abbildung 21, eingetragen. Dieser Vorgang kann für jeden Autor wiederholt werden. Das Ergebnis ist eine Liste bei dem jeder Listeneintrag ein Name eines Autors des Projekts ist.

Der vierte Teil des Fensters ist für den Speicherort des erstellten Projekts verantwortlich. In dem Texteingabefeld, gekennzeichnet durch das Eingabefeld 5.2 in der Abbildung 21, kann dazu der Pfad eingegeben werden, in dem das Projekt nach der Erstellung gespeichert werden soll. Alternativ dazu, kann, wie auch in Kapitel 5.2.1 erklärt, der „...“-Button, gekennzeichnet durch den Button 5.3 in der Abbildung 21, gedrückt werden, um ein Ordner-Dialog zu öffnen. In diesem Dialog kann anschließend der entsprechende Pfad ausgewählt und bestätigt werden. Durch die Bestätigung wird der Pfad automatisch in das Texteingabefeld eingetragen, gekennzeichnet durch das Eingabefeld 5.2 in der Abbildung 21.

Als letztes befinden sich unten in dem Fenster noch zwei weitere Buttons, gekennzeichnet durch die Buttons 6.1, 6.2 in der Abbildung 21. Der „Back to Menu“-Button dient lediglich dazu die Fenster zu wechseln, um zurück auf das Hauptmenü, dass in Kapitel 5.2.1 erklärt wurde, zu gelangen. Durch den „Create Project“-Button werden alle getätigten Eingaben in dem Fenster bestätigt und das neue Projekt erstellt. Nachdem das Projekt erstellt wurde, wechselt die GUI wieder auf das Hauptmenü-Fenster.

4.2.3 Fenster für die Projektaktualisierung

Mit diesem Screen kann ein Projekt, dass aus der Projekten-Liste des Hauptmenüs ausgewählt wurde, mit der Projekt-Schablone verglichen und aktualisiert werden. Das Fenster wird dabei in der GUI angezeigt, wenn der „Update Project“-Button, gekennzeichnet durch den Button 3.6 in der Abbildung 20, im Hauptmenü angeklickt wird.

1.1 Update Project

2.1 Project Settings

2.2 Project Name (Long): 2.3 Texteingabefeld

2.4 Project Name (Short): 2.5 Texteingabefeld

2.6 License Type: 2.7 Dropdown

2.8 Conan Support: ☐ 2.9

2.10 Use Gtest: ☐ 2.11

2.12 Use Contribution Guide: ☐ 2.13

2.14 Linting and Formatting: ☐ 2.15

2.16 Use GItignore: ☐ 2.17

2.18 Use GItLab CI/CD: ☐ 2.19

3.1 Updatable files

3.2 Liste: Aktualisierbare Dateien und Ordner

4.1.1 Button: Take Yours

4.2.1 Button: Accept Changes

4.2.2 Button: Reset Changes

4.3.1 Button: Take Theres

4.1.2 Liste: Projekt

4.2.3 Liste: Merged

4.3.2 Liste: Template

5.1 Button: Back to Menu

5.2 Button: Update Project

Abbildung 22: Skizze für den Screen der Projektaktualisierung

Die Abbildung 22, zeigt die Skizze für das Fenster der Projektaktualisierung. Das Fenster ist dabei aus drei verschiedenen Teilen aufgebaut, gekennzeichnet als blau, rot und grün.

Der erste Teil ist für die Veranschaulichung der getätigten Projekteinstellungen zuständig. Die gezeigten Werte sind dabei der lange und kurze Projektname, der ausgewählte Lizenztyp, sowie die ausgewählten Konfigurationsdateien. Die Werte werden bei dem Aufruf des Fensters automatisch eingetragen. Die Eingabefelder, gekennzeichnet durch die Texteingabefelder 2.3 und 2.5 in der Abbildung, stellen den Namen des Projektes dar. Dabei steht in dem ersten Eingabefeld der lange und in dem zweiten Eingabefeld der kurze Projektname. Der verwendete Lizenztyp des Projektes wird anhand der Dropdown-Liste, gekennzeichnet durch die Dropdown-Liste 2.7 der Abbildung wiedergegeben. Der Lizenztyp ist dabei als ersten Wert der Dropdown-Liste eingestellt. Die Checkboxes 2.9, 2.11, 2.13, 2.15, 2.17, 2.19 der Abbildung, spiegeln die Auswahl der Konfigurationsdateien bei der Projekterstellung wider. Dementsprechend sind alle Checkboxes, die bei der Projekterstellung markiert wurden, auch in diesem Fenster markiert. Diese Einstellungen können allerdings nach Bedarf verändert werden. Das bedeutet, der lange und kurze Projektname können in den Texteingabefelder umbenannt werden, in der Dropdown-Liste kann ein anderer Lizenztyp ausgewählt werden und die Checkboxes könne nach Belieben markiert und entmarkiert werden. Alle Änderungen in diesem Teil werden am Ende bei der Projektaktualisierung für das ausgewählte Projekt angewendet.

Der zweite Teil des Fensters ist für die Visualisierung aller Dateien und Ordner zuständig, die für das ausgewählte Projekt aktualisiert werden können. Zusätzlich beinhaltet die Visualisierung auch alle Dateien und Ordner, die der Projekt-Schablone neu hinzugefügt wurden und somit nicht in dem ausgewählten Projekt enthalten sind. Für die Visualisierung werden die Pfade zu allen aktualisierbaren und neuen Dateien und Ordner des ausgewählten Projektes, in einer Liste mit einer Baum-Ansicht, gekennzeichnet durch die Liste 3.2 in der Abbildung, aufgelistet. Die Pfade zu den entsprechenden Dateien oder Ordnern sind dabei anklickbar.

Der dritte Teil des Fensters ist wiederum selbst aus drei verschiedenen Unterteilen aufgebaut. Der erste Unterteil besteht aus dem „Take Yours“-Button und der Projekt-Inhalt-Liste, gekennzeichnet durch den Button 4.1.1 und der Liste 4.1.2 in der Abbildung. Der zweite Unterteil besteht aus dem „Accept Changes“-Button, dem

„Reset Changes“- Button und der zusammengesetzten (eng.: merged) Liste, gekennzeichnet durch den Button 4.2.1, den Button 4.2.2 und der Liste 4.2.3 in der Abbildung. Der dritte Unterteil besteht aus dem „Take Yours“-Button und der Template-Inhalt-Liste, gekennzeichnet durch den Button 4.3.1 und der Liste 4.3.2 in der Abbildung.

Die Unterteile werden nach dem Aufruf des Fensters standardmäßig nicht in dem Fenster angezeigt. Das bedeutet der Platz, an dem die Unterteile in der Abbildung gezeigt werden, ist nach dem Aufruf des Fensters standardmäßig leer. Sie werden erst in der GUI angezeigt, wenn der Pfad einer aktualisierbaren oder neuen Datei in der Liste des zweiten Teils angeklickt wird. Dabei zeigt die Projekt-Inhalt-Liste des ersten Unterteils, den Inhalt der Datei, die sich im ausgewählten Projekt befindet. Falls es sich hierbei um eine neue Datei handelt, also eine Datei die nicht im ausgewählten Projekt, sondern nur in der Projekt-Schablone existiert, wird in der Projekt-Inhalt-Liste eine leere Liste angezeigt. Die Template-Inhalt-Liste des dritten Unterteils zeigt den Inhalt der Datei, die sich in der Projekt-Schablone befindet. Die Merged-Liste des zweiten Unterteils, zeigt den zusammengesetzten Inhalt beider Dateien. Der Inhalt dieser Liste ist dabei konfigurierbar. Das heißt, ausgewählte Zeilen der Liste können per Doppelklick mit der Maus gelöscht werden. Des Weiteren können einzelne Zeilen der Projekt-Inhalt-Liste und der Template-Inhalt-Liste per Drag-and-Drop in die Merged-Liste hinzugefügt werden. Wenn bspw. nur der Inhalt der Template-Datei benötigt wird, kann dazu der „Take-There“-Button gedrückt werden. Dadurch wird der Inhalt der Merged-Liste mit dem Inhalt aus der Template-Inhalt-Liste überschrieben. Entgegengesetzt kann der „Take Yours“-Button gedrückt werden, falls nur der Inhalt der Projekt-Datei benötigt wird. Dadurch wird der Inhalt der Merged-Liste mit dem Inhalt aus der Projekt-Inhalt-Liste überschrieben.

Nachdem die Konfiguration der Merged-Liste abgeschlossen ist, kann der „Accept Changes“-Button gedrückt werden, um den Inhalt der Merged-Liste zu speichern und die Änderungen der ausgewählten Datei, aus der Liste des zweiten Teils, zuzuordnen. Dabei kann die Merged-Liste auch leer sein, weil bspw. eine neue Datei in der Liste des zweiten Teils ausgewählt wurde und anschließend der „Take Yours“-Button gedrückt wurde. Mit dem „Reset Changes“-Button können die gespeicherten Änderungen für die Datei wieder gelöscht werden.

Wenn für alle aktualisierbaren und neuen Dateien eine Änderung gespeichert worden ist, kann das Projekt mit dem „Update Project“-Button aktualisiert werden. Mit dem Klick auf diesen Button wird ein Prozess gestartet, bei dem alle aktualisierbaren Dateien aus der Liste des zweiten Teils in dem ausgewählten Projekt geöffnet werden und mit dem gespeicherten Inhalt überschrieben werden. Alle neuen Dateien aus der Liste werden dem ausgewählten Projekt, basierend auf den gespeicherten Inhalten, entweder hinzugefügt oder nicht. Wenn der gespeicherte Inhalt leer ist, wird dem Projekt die neue Datei nicht hinzugefügt.

Mit dem „Back to Menu“-Button, gekennzeichnet durch den Button 3.1 in der Abbildung, wechselt die GUI das Fenster zum Hauptmenü.

4.2.4 Fenster für die Erweiterung der Projektmodule

Dieses Fenster ähnelt dem Fenster in Kapitel 5.2.2 für die Projekterstellung, da er für die Erfassung aller erforderlichen Informationen zur Erstellung eines neuen Moduls zuständig ist und daher auch wie ein Formular funktioniert. Das Fenster wird dabei in der GUI angezeigt, wenn der „Add Cmake“-Button, gekennzeichnet durch den Button 3.4 in der Abbildung 20, des Hauptmenüs angeklickt wird.

1.1 Add Cmake Modul

2.1 Select Module Type: 2.2 Dropdown

2.3 Module Name: 2.4 Texteingabefeld

2.5 Select local dependency: ☐ 2.6.1 Name der Abhängigkeit ☐ 2.6.2 Private ☐ 2.6.3 Private

2.6 Dependency Liste

2.7 External Dependency: 2.8 Texteingabefeld 2.9 Button: Add

3.1 Button: Back to Menu 3.2 Button: Add Cmake

Abbildung 23: Skizze für den Screen der Projektmodulerweiterung

In der Abbildung 23 ist zusehen, dass das Fenster aus einem Teil, gekennzeichnet in blau, aufgebaut ist.

In diesem Teil werden alle notwendigen Informationen über das Projektmodul gesammelt. Jedes Modul hat einen Typ, der angegeben werden muss, zum Beispiel ein Bibliotheksmodul, ein ausführbares Modul, ein Schnittstellenmodul und so weiter. Dieser Typ kann mit der Auswahl eines Wertes aus der Dropdown-Liste, gekennzeichnet durch die DropDown-Liste 2.2 in der Abbildung 23, bestimmt werden. Als nächstes muss der Name des Moduls festgelegt werden. Dazu muss ein Modulname in das Texteingabefeld, gekennzeichnet durch das Eingabefeld 2.4 in der

Abbildung 23, eingegeben werden. Anschließend kann eine Auswahl getroffen werden, welche lokalen Abhängigkeiten dem Projektmodul hinzugefügt werden sollen. Dazu werden in der Liste, gekennzeichnet durch die Liste 2.6 in der Abbildung 23, alle lokalen Abhängigkeiten aufgelistet, die für das neue Projektmodul des ausgewählten Projektes ausgewählt werden können. Die Abhängigkeiten werden der Liste dabei als Listenelemente hinzugefügt, gekennzeichnet durch den rot markierten Teil in der Abbildung 23. Um eine lokale Abhängigkeit zu bestimmen, muss diese in der Liste, durch markieren der Checkbox, gekennzeichnet durch die Checkbox 1.1 in der Abbildung 23 bestätigt werden und durch das Klicken auf eines der zwei Radio-Buttons, gekennzeichnet durch die Radio-Button 1.3 und 1.5 in der Abbildung 23, angegeben werden, ob diese als private oder öffentliche Abhängigkeit hinzugefügt werden soll. Um das neue Projektmodul mit den eingegebenen Informationen zu erstellen, muss der „Add Cmake“-Button, gekennzeichnet durch den Button 3.2 in der Abbildung 23, gedrückt werden. Das Projektmodul wird dabei direkt in dem ausgewählten Projekt erstellt.

Mit dem „Back to Menu“-Button, gekennzeichnet durch den Button 3.1 in der Abbildung 23, wechselt die GUI das Fenster zum Hauptmenü.

4.2.5 Fenster für die Ausführung der Dienstprogramme

Durch dieses Fenster können verschiedene Dienstprogramme (eng.: utilities) eines Projektes, mit benutzerdefinierten Befehlen, ausgeführt werden. Das Fenster wird dabei in der GUI angezeigt, wenn der „Run Utils“-Button, gekennzeichnet durch den Button 3.8 in der Abbildung 20, des Hauptmenüs angeklickt wird.

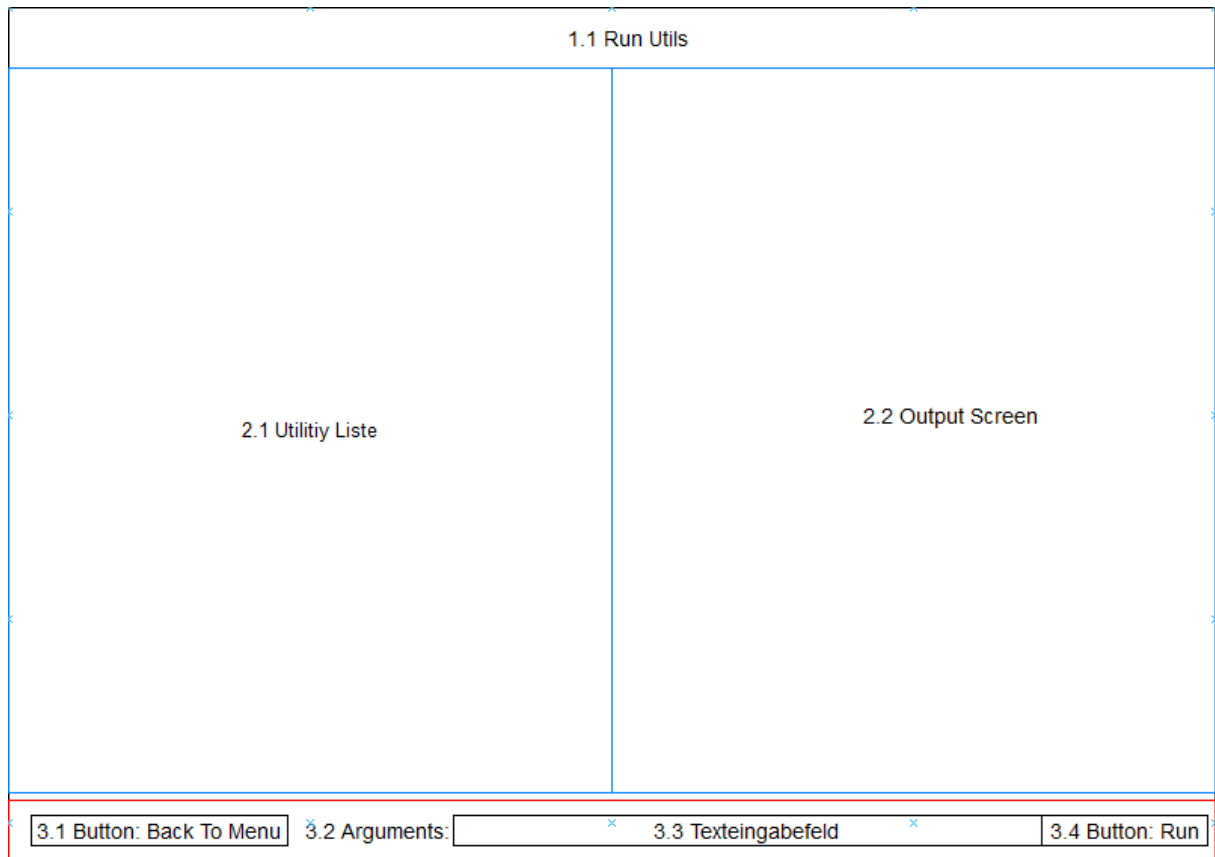


Abbildung 24: Skizze für den Screen der Dienstprogrammausführung

In der Abbildung 24 ist zusehen, dass das Fenster aus zwei Teilen, gekennzeichnet in blau und rot, aufgebaut ist.

Der erste Teil besteht wiederum selbst aus Unterteilen – die linke Seite, gekennzeichnet durch die Liste 2.1 in der Abbildung 24, und die rechte Seite, gekennzeichnet durch die Liste 2.2 in der Abbildung 24. Die linke Seite ist eine Liste, in der alle ausführbaren Dienstprogramme des ausgewählten Projektes aufgelistet werden. Die Dienstprogramme in der Liste können dabei angeklickt bzw. ausgewählt werden. Nachdem das passiert ist, wird das ausgewählte Dienstprogramm mit dem Befehl „--help“ ausgeführt. Die resultierende Ausgabe wird anschließend in dem Text Browser auf der rechten Seite ausgegeben. Eine weitere Funktion bietet der Text

Browser nicht. Er dient lediglich als Bildschirm für die Ausgaben der ausgeführten Dienstprogramme.

Der zweite Teil des Fensters, beinhaltet zwei Buttons, gekennzeichnet durch die Buttons 3.1 und 3.4 in der Abbildung 24, und ein Texteingabefeld, gekennzeichnet durch das Eingabefeld 3.3 in der Abbildung 24. Mit dem „Back to Menu“-Button, gekennzeichnet durch den Button 3.1 in der Abbildung 24, wechselt, wie in den Kapiteln davor, die GUI das Fenster zum Hauptmenü. In das Texteingabefeld, gekennzeichnet durch das Eingabefeld 3.3 in der Abbildung 24, können benutzerdefinierte Befehle für einen ausgewähltes Dienstprogram aus der Liste des ersten Teils eingegeben werden. Mit dem „Run“-Button, gekennzeichnet durch den Button 3.4 in der Abbildung 24, kann das ausgewählte Dienstprogramm mit den eingegeben benutzerdefinierten Befehlen ausgeführt werden. Die resultierende Befehlsausgabe wird anschließend auf dem Text Browser ausgegeben.

4.3 Softwarearchitektur

Mithilfe der Skizzen konnte das Aussehen, der Aufbau und der gesamte Funktionsumfang der GUI dargestellt werden. Anhand dieser Informationen wurde die Architektur der Software, in Form von zwei Klassendiagrammen, konstruiert. Das erste Klassendiagramm ist das Basis-Diagramm, mit dem die Grundarchitektur der Software verdeutlicht wird. Das zweite Klassendiagramm ist das vollständige Diagramm, in der alle Klassen und Beziehungen abgebildet sind.

4.3.1 Basis-Architektur

Die Abbildung 25 zeigt, die Basis-Architektur der Software, aus der hervor geht, dass die Software aus den folgenden vier Basis-Klassen aufgebaut ist: *App*, *Frame*, *Screen* und *SubScreen*.

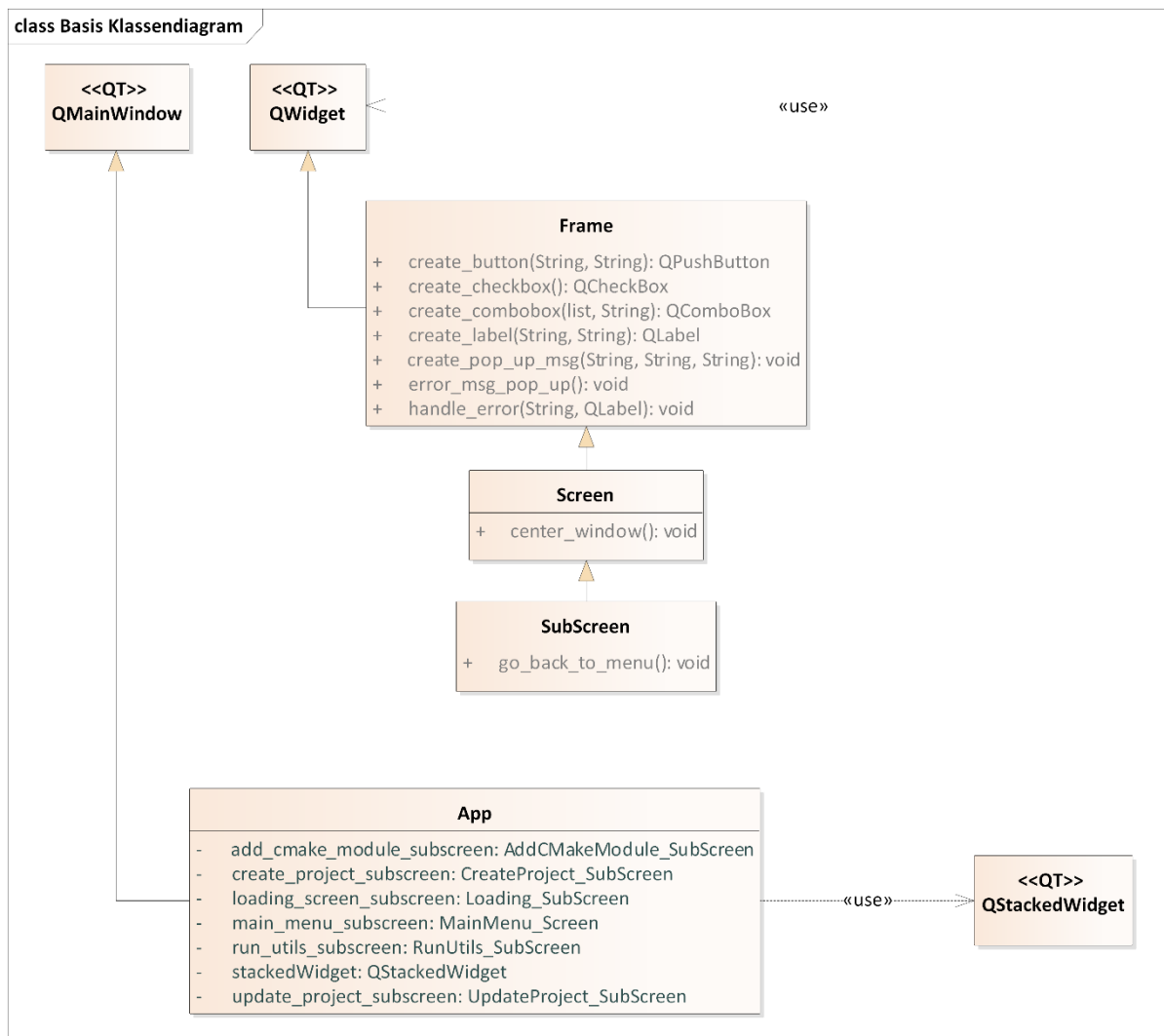


Abbildung 25: Basis Klassendiagramm

Die Klasse *App* ist dabei das Hauptfenster der Anwendung, weswegen es auch von der „Qt“-Framework spezifischen Klasse *QMainWindow* erbt. Das Qt Main Window stellt dabei einen Rahmen zum Erstellen der Benutzerschnittstelle einer Anwendung bereit. Es enthält alle Klassen, die für die Verwaltung, also das Minimieren, Verkleinern und Schließen, des Hauptfensters nötig sind. Die Klasse *QMainWindow* besitzt ein eigenes Layout, dass folgendermaßen aussieht:

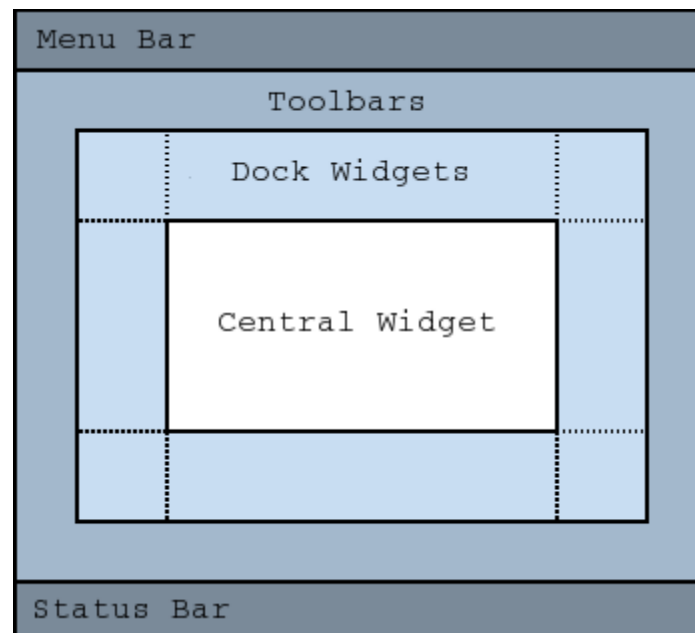


Abbildung 26: Layout der Klasse *QMainWindow* [29]

Diesem Layout kann eine Menüleiste, Werkzeugleiste, Dock-Widgets und eine Statusleiste hinzugefügt werden. Für die Klasse *App* wurden allerdings keine dieser Optionen verwendet. Außerdem besitzt das Layout einen zentralen Bereich, der von jeder Art von Widget belegt werden kann. Dieses zentrale Widget muss für ein *QMainWindow* gesetzt werden, selbst wenn es nur ein Platzhalter ist. Ohne ein diesen kann ein *QMainWindow* nicht existieren [30].

Für die Klasse *App* ist das zentrale Widget die „Qt“-Framework spezifische Klasse *QStackedWidget*, welches einen Stack (dt.: Stapel) von Widgets bereitstellt, bei dem jeweils nur ein Widget sichtbar ist. Die Widgets in dem Stack der Klasse *QStackedWidget* besitzen dabei einen Index, der aussagt, an welcher Stelle die jeweiligen Widgets sich in dem Stack befinden. Wenn sich mindestens ein Widget in dem Stack befindet, wird standardmäßig das erste Widget in dem Stack, also das Widget mit dem Index 0, von der Klasse *QStackedWidget* angezeigt. Allerdings besitzt die Klasse eine integrierte Funktion, mit der sich das angezeigt Widget wechseln

lässt [31]. Dementsprechend kann die Klasse *QStackedWidget* mit einem Buch verglichen werden, wobei die Klasse *QStackedWidget* das Buch ist und der Stack mit den Widgets die einzelnen Seiten in diesem Buch. Das Umblättern der Seiten ist bei diesem Vergleich die Funktion, mit der sich angezeigte Widgets wechseln lassen.

Um das *QStackedWidget* als zentrales Widget nutzen zu können, erstellt die Klasse *App* eine Instanz der Klasse *QStackedWidget* und fügt diese sich selbst als zentrales Widget hinzu. Die Widgets, die als Elemente der *QStackedWidget*-Instanz gebaut werden, werden in dem vorherigen Kapitel 5.1 erklärt. Diese Elemente sind dabei die Kinder, der Klasse *Screen* (dt.: Fenster) oder der Klasse *SubScreen* (dt.: Unterfenster). Das Hauptmenü, welches in Kapitel 5.2.1 gezeigt wurde, ist ein Kindelement der Klasse *Screen*, mit dem Namen: *MainMenu_Screen*. Alle anderen Kindelemente der *QStackedWidget*-Instanz sind spezialisierte Instanzen der *SubScreen*-Klasse. Diese lauten wie folgt: *CreateProject_SubScreen*, dargestellt in dem Kapitel 5.2.2, *UpdateProject_SubScreen*, dargestellt in dem Kapitel 5.2.3, *AddCmakeModule_SubScreen*, dargestellt in dem Kapitel 5.2.4, *RunUtils_SubScreen*, dargestellt in dem Kapitel 5.2.5.

Der Grund für die Unterteilung in *Screens* und *SubScreens* ist, dass das Hauptmenü, durch dessen Buttons, den Index der *QStackedWidget*-Instanz auf den Index eines *SubScreens* wechseln kann, sodass dieser von der *QStackedWidget*-Instanz angezeigt wird. Diese Funktionalität dürfen die *SubScreens* nicht besitzen, da diese nicht auf andere *SubScreens* wechseln dürfen. Die einzige Möglichkeit wie die *SubScreens* mit der *QStackedWidget*-Instanz interagieren dürfen, ist durch den „Back to Menu“-Button. Dadurch wird der Index der *QStackedWidget*-Instanz auf den Index des *MainMenu_Screens* gewechselt, sodass die GUI wieder das Hauptmenü anzeigt. Diese Funktionalität dürfen wiederum die *Screens* nicht besitzen, um auszuschließen, dass die *Screens* immer wieder auf sich selbst wechseln.

Des Weiteren ist in der Abbildung 25 zusehen, dass die *Screens* und *SubScreens* von der Klasse *Frame* (dt.: Rahmen) erben, die wiederum von der „Qt“-Framework spezifischen Klasse *QWidget* erbt. Das bedeutet, dass alle *Frames*, *Screens* und *SubScreens* *QWidgets* sind. Deswegen können die *Screens* und *SubScreens* auch zu dem Stack der *QStackedWidget*-Instanz hinzugefügt werden. Die Kinderelemente der Klasse *Frame* werden dazu benutzt, um die farblich markierten Teile (inklusive aller Widgets) der Skizzen des Kapitels 5.1 in den *Screens* / *Subscreens* zu implementieren.

Insgesamt gibt es zwölf Kinderelemente der Klasse *Frame*. Die *Screens* / *SubScreens* erstellen dazu Instanzen der Kinderelemente und fügen diese ihrem Layout hinzu. Die Instanzen beinhalten dabei alle Widgets, die die farblich markierten Teile in den Skizzen des Kapitels 5.1 auch besitzen.

Die Erbreihenfolge der Klasse *Frame*, *Screen* und *SubScreen* wurde so gewählt, weil die Klasse *Frame* verschiedene Funktionen bereitstellt, um benutzerdefinierte Widgets zu erstellen. Diese Funktionen werden von den *Screens* und *SubScreens* gleichermaßen benötigt. Weitere Funktionen beinhaltet die Klasse *Frame* nicht, da diese nur zum Aufbau der verschiedenen *Frames* dienen soll. Deswegen bauen die Klassen *Screen* und *SubScreen* auf der Klasse *Frame* auf.

Zusammengefasst, besteht die Anwendung aus verschiedenen *Screens* und *SubScreens*, die aus verschiedenen *Frames* aufgebaut sind. Diese *Screens* und *SubScreens* befinden sich innerhalb eines *QStackedWidget*, der zur gleichen Zeit immer nur ein Widget seines Stacks zeigt. Das *QStackedWidget* ist dabei zentrale Widget einer Klasse *App*, welche außerdem das Hauptfenster der Anwendung ist. Die Klasse *App* wird bei der Ausführung des Projekt-Konfigurators gezeigt.

4.3.2 Vollständige Architektur

In diesem Kapitel werden die restlichen Klassen der Software, mithilfe verschiedener Klassendiagramme gezeigt und erklärt. Die Klassendiagramme befinden sich dabei aufgrund ihrer Bildgröße im Anhang dieser Bachelorarbeit. Der Anhang ist am Ende dieser Arbeit zu finden.

Der Anhang A zeigt das Klassendiagramm für das Hauptmenü der GUI. Um das Hauptmenü und all dessen Funktionen zu erstellen, wird die Klasse *MainMenu_Screen* verwendet. Dazu erstellt, wie in dem letzten Kapitel 5.3.1 erklärt, die Klasse *App* eine Instanz der Klasse *MainMenu_Screen* und fügt diese der Instanz der Klasse *QStackedWidget* hinzu. Wie in Kapitel 5.2.1 erklärt, besteht das Layout des Hauptmenüs dabei aus zwei Teilen. Der erste Teil, gekennzeichnet durch die Farbe Blau in der Abbildung 20 des Kapitels 5.2.1, wird mithilfe der Klasse *SearchProject_Frame* erstellt, einem Kindelement der Klasse *Frame*. Dazu erstellt *MainMenu_Screen* eine Instanz der Klasse *SearchProject_Frame* und fügt diese Instanz dem eigenen Layout hinzu. Die *SearchProject_Frame*-Instanz enthält dabei alle notwendigen Widgets, die auch in der Abbildung 20 des Kapitels 5.2.1 zu sehen

sind. Der zweite Teil, gekennzeichnet durch die Farbe Rot in der Abbildung 20 des Kapitels 5.2.1, sowie alle anderen Widgets des Hauptmenüs werden direkt in dem Konstruktor der Klasse *MainMenu_Screen* erstellt.

Der Anhang B zeigt das Klassendiagramm für das Fenster der Projekterstellung in der GUI. Um das Fenster für die Projekterstellung, wie in dem vorherigen Kapitel 5.2.2 erklärt, zu erstellen, wird die Klasse *CreateProject_SubScreen* verwendet. Dazu erstellt die Klasse *App* eine Instanz der Klasse *MainMenu_Screen* und fügt diese der Instanz der Klasse *QStackedWidget* hinzu. Wie in der Abbildung 21 des Kapitels 5.2.2 dargestellt, ist das Fenster für die Projekterstellung aus vier Teilen, gekennzeichnet durch die Farben Blau, Rot, Grün und Pink, aufgebaut. Der erste Teil wird mithilfe der Klasse *ProjectNameAndReadme_Frame* erstellt, der zweite Teil mithilfe der Klasse *FeaturesCheckbox_Frame*, der dritte Teil mithilfe der Klasse *AddAuthor_Frame* und der letzte Teil mithilfe der Klasse *SavingDirectory_Frame*. Alle vier Klassen sind dabei Kinderelemente der Klasse *Frame*. Um die Klassen für die farblich markierten Teile nutzen zu können, erstellt die Klasse *CreateProject_SubScreen* Instanzen der vier Frame-Klassen und fügt diese nacheinander dem eigenen Layout hinzu. Die restlichen Widgets, die in der Abbildung 21 des Kapitels 5.2.2 zu sehen sind, werden direkt in dem Konstruktor der Klasse *CreateProject_SubScreen* erstellt.

Der Anhang C zeigt das Klassendiagramm für das Fenster der Projektaktualisierung in der GUI. Um das Fenster für die Projektaktualisierung, wie in dem vorherigen Kapitel 5.2.3 erklärt, zu erstellen, wird die Klasse *UpdateProject_SubScreen* verwendet. Dazu erstellt, wie in dem letzten Kapitel 5.2.1 erklärt, die Klasse *App* eine Instanz der Klasse *UpdateProject_SubScreen* und fügt diese der Instanz der Klasse *QStackedWidget* hinzu. Wie in der Abbildung 22 des Kapitels 5.2.3 dargestellt, ist das Fenster für die Projektaktualisierung aus drei Teilen, gekennzeichnet durch die Farben Blau, Rot und Grün, aufgebaut. Der erste Teil wird mithilfe der Klasse *ProjectConfig_Frame* erstellt, der zweite Teil mithilfe der Klasse *UpdatableFiles_Frame* und der dritte Teil mithilfe der Klasse *DiffOutput_Frame*. Alle vier Klassen sind dabei Kinderelemente der Klasse *Frame*. Die Klassen *ProjectConfig_Frame* und *UpdatableFiles_Frame* sind dabei Kinderelemente der Klasse *Frame*, die Klasse *DiffOutput_Frame* ist dagegen ein Kindelement der Qt spezifischen Klasse *QSplitter*. Um die Klassen für die farblich markierten Teile nutzen zu können, erstellt die Klasse *UpdateProject_SubScreen* Instanzen der vier Frame-Klassen und fügt diese nacheinander dem eigenen Layout

hinzu. Die drei Instanzen enthält dabei alle notwendigen Widgets, die auch in der Abbildung 22 des Kapitels 5.2.3 zu sehen sind. Die Klasse *DiffOutput_Frame* benötigt dafür allerdings zwei weitere Klassen, um die drei Unterteile, beschrieben in Kapitel 5.2.3, des grün markierten Teils in der Abbildung 22 zu erstellen. Dafür erstellt die Klasse *DiffOutput_Frame* zwei Instanzen der Klasse *Source_Frame* und eine Instanz der Klasse *Recipient_Frame* und fügt diese sich selbst als Splitter-Elemente hinzu. Die zwei *Source_Frame*-Instanzen sind dabei die beiden äußeren Splitter-Elemente des grün markierten Teils der Abbildung 22 in dem Kapitel 5.2.3. Das mittlere Splitter-Element, desselben grün markierten Teils, wird mithilfe der *Recipient_Frame*-Instanz erstellt. Jedes der *Source_-/Recipient_Frame*-Instanzen sind Kinderelemente der Klasse *Frame* und besitzen alle notwendigen Widgets, die Abbildung 22 des Kapitels 5.2.3 zu sehen sind. Die restlichen Widgets außerhalb der farblich markierten Teile, die in der Abbildung 22 des Kapitels 5.2.3 zu sehen sind, werden direkt in dem Konstruktor der Klasse *UpdateProject_SubScreen* erstellt.

Der Anhang D zeigt das Klassendiagramm für das Fenster der Projekterweiterung in der GUI. Um das Fenster für die Erweiterung der Projektmodule, wie in dem vorherigen Kapitel 5.2.4 erklärt, zu erstellen, wird die Klasse *AddCmakeModule_SubScreen* verwendet. Dazu erstellt, wie in dem letzten Kapitel 5.3.1 erklärt, die Klasse *App* eine Instanz der Klasse *AddCmakeModule_SubScreen* und fügt diese der Instanz der Klasse *QStackedWidget* hinzu. Wie in der Abbildung 23 des Kapitels 5.2.4 dargestellt, ist das Fenster für die Erweiterung der Projektmodule aus einem Teil, gekennzeichnet durch die Farbe Blau, aufgebaut. Dieser Teil wird mithilfe der Klasse *AddCmakeModule_Frame* erstellt, einem Kindelement der Klasse *Frame*. Dazu erstellt *AddCmakeModule_SubScreen* eine Instanz der Klasse *AddCmakeModule_Frame* und fügt diese Instanz dem eigenen Layout hinzu. Die *AddCmakeModule_Frame*-Instanz enthält dabei alle notwendigen Widgets, die auch in der Abbildung 23 des Kapitels 5.2.4 zu sehen sind. Darunter befindet sich auch die Liste, gekennzeichnet durch die List 2.6 in der Abbildung 23 des Kapitels 5.2.4, in der alle lokalen Abhängigkeiten des ausgewählten Projektes aufgelistet sind. Wie durch den rot markierten Teil in der Abbildung 23 des Kapitels 5.2.4 zusehen, weisen die Listenelemente dieser Liste, mit der die lokalen Abhängigkeiten eines ausgewählten Projektes visualisiert werden, ein spezielles Layout auf. Deswegen werden die Listenelemente mithilfe der Klasse *ListItemWidget_Frame* erstellt, welche auch ein Kindelement der Klasse *Frame* ist. Dazu erstellt die Klasse *AddCmakeModule_Frame*

beliebig viele Instanzen der Klasse *ListItemWidget_Frame* und fügt diese der Abhängigkeiten-Liste als Listenelemente hinzu. Die restlichen Widgets außerhalb des blau markierten Teils, die in der Abbildung 23 des Kapitels 5.2.4 zu sehen sind, werden direkt in dem Konstruktor der Klasse *AddCmakeModule_SubScreen* erstellt.

Der Anhang E zeigt das Klassendiagramm für das Fenster der Dienstprogrammausführung in der GUI. Um das Fenster für die Ausführung der Dienstprogramme eines Projektes, wie in dem vorherigen Kapitel 5.2.5 erklärt, zu erstellen, wird die Klasse *RunUtils_SubScreen* verwendet. Dazu erstellt, wie in dem letzten Kapitel 5.3.1 erklärt, die Klasse *App* eine Instanz der Klasse *RunUtils_SubScreen* und fügt diese der Instanz der Klasse *QStackedWidget* hinzu. Wie in der Abbildung 24 des Kapitels 5.2.5 dargestellt, ist das Fenster für die Ausführung der Dienstprogramme eines Projektes aus zwei Teilen, gekennzeichnet durch die Farben Blau und Rot, aufgebaut. Der erste Teil wird mithilfe der Klasse *Utility_Frame* erstellt, einem Kindelement der Klasse *Frame*. Dazu erstellt *RunUtils_SubScreen* eine Instanz der Klasse *Utility_Frame* und fügt diese Instanz dem eigenen Layout hinzu. Die *Utility_Frame*-Instanz enthält dabei alle notwendigen Widgets, die auch in der Abbildung 24 des Kapitels 5.2.5 zu sehen sind. Der zweite Teil, gekennzeichnet durch die Farbe Rot in der Abbildung 24 des Kapitels 5.2.5, sowie alle anderen Widgets dieses Fensters werden direkt in dem Konstruktor der Klasse *RunUtils_SubScreen* erstellt.

Der Anhang F zeigt das Klassendiagramm für den Ladebildschirm, der in der GUI angezeigt wird, bevor die GUI auf das Fenster für die Projektaktualisierung wechselt. Der Grund für den Ladebildschirm ist, dass das Fenster für die Projektaktualisierung erst mit entsprechenden Daten konfiguriert werden muss, bevor es angezeigt werden kann. Dieser Konfigurationsprozess kann eine Weile dauern.

Wenn in dem Hauptmenü der GUI der „Update Project“-Button, gekennzeichnet durch den Button 3.6 in der Abbildung 20 des Kapitels 5.2.1, gedrückt wird, wechselt die GUI nicht direkt auf das Fenster für die Projektaktualisierung, dass in dem vorherigen Kapitel 5.2.3 beschrieben wurde, sondern erst auf einen Ladebildschirm und im Anschluss auf das Fenster für die Projektaktualisierung. Der Grund dafür ist, dass nach dem Klick auf den „Update Project“-Button im Hintergrund ein Prozess gestartet wird, bei dem das ausgewählte Projekt aus der Projekten-Liste, gekennzeichnet durch die Liste 2.5 in der Abbildung 20 des Kapitels 5.2.1, mit der Projekt-Schablone verglichen

wird. Während dem Vergleich werden alle Dateien die aktualisierbar sind in der Liste, gekennzeichnet durch die Liste 3.2 der Abbildung 22 des Kapitels 5.2.3, des Fensters für die Projektaktualisierung aufgelistet. Dieser Prozess benötigt Zeit, weswegen die GUI, solange dieser Prozess läuft, auf einen Ladebildschirm wechselt. Nachdem der Vergleich zu Ende ist und alle aktualisierbaren Dateien in die Liste eingetragen wurden, wechselt die GUI von dem Ladebildschirm auf das Fenster für die Projektaktualisierung.

Die Klasse *Loading_Screen* wird dazu verwendet, um den Ladebildschirm der GUI zu erstellen. Dazu erstellt, wie in dem letzten Kapitel 5.3.1 erklärt, die Klasse *App* eine Instanz der Klasse *Loading_Screen* und fügt diese der Instanz der Klasse *QStackedWidget* hinzu. *Loading_Screen* ist dabei ein Kindelement der Klasse *Screen*. Wenn der Prozess, der in dem vorherigen Abschnitt erklärt wurde, gestartet wird, wird der Index des Stacks der *QStackedWidget*-Instanz auf den Index des *Loading_Screens* gewechselt, sodass die GUI den Ladebildschirm anzeigt.

Der beschriebene Prozess, bei dem das ausgewählte Projekt mit der Projekt-Schablone verglichen wird, wird nebenbei in einem neuen Thread gestartet. Um den neuen Thread zu erzeugen, wird die Klasse *QThread* benötigt. Eine Instanz dieser Klasse wird in der Klasse *MainMenu_Screen* erzeugt, nachdem der „Update Project“-Button, gekennzeichnet durch den Button 3.6 in der Abbildung 20 des Kapitels 5.2.1, gedrückt wird. Anschließend erstellt die Klasse *MainMenu_Screen* eine Instanz der Klasse *Job*, die bei ihrer Erstellung eine beliebige Anzahl von Funktionen übergeben bekommt. Nach der Erstellung wird die *Job*-Instanz der *QThread*-Instanz zugewiesen, wodurch alle Aufgaben bzw. Funktionen, die der *Job*-Instanz übergeben wurde, ausgeführt werden. Solange diese Funktionen ausgeführt werden, ist in der GUI der Ladebildschirm zusehen. Nachdem die Funktionen beendet wurden, schließt die Klasse *MainMenu_Screen* den Thread und wechselt das Fenster der GUI zu dem Fenster der Projektaktualisierung, welches in Kapitel 5.2.3 beschrieben wurde.

4.4 Zusammenfassung

In diesem Kapitel wurde erklärt, wie die Software des Projekt-Konfigurators designt wurde.

Dazu wurde als erstes der Aufbau der GUI erklärt, indem das Aussehen und die Funktionen jedes GUI-Fensters mithilfe von Skizzen erläutert wurden.

Als zweites wurde das Basis-Klassendiagramm erklärt, wodurch der grundlegende Aufbau der GUI deutlich wurde. Dadurch wurde außerdem wurde ersichtlich, wieso die Klassen Frame, Screen und SubScreen existieren und wieso diese benötigt werden.

Als letztes wurde das gesamte Klassendiagramm und damit der Aufbau jedes Screens und SubScreens erklärt.

5. Implementierung der Software

In diesem Kapitel wird erklärt, wie die gesamte Software implementiert wurde. Dazu wird als erstes die intern gespeicherte Projekt-Schablone gezeigt und erklärt. Als nächstes wird beschrieben wie die Fenster inklusive ihrer Funktionen, die in Kapitel 5.1 erklärt wurden, implementiert wurden. Dabei werden alle Funktionen mithilfe verschiedener Flussdiagramme, detailliert erklärt.

5.1 Interne Projekt-Schablone

Für die Projekterstellung und die Projektaktualisierung wird eine Projekt-Schablone benötigt, auf dessen Basis Projekte erstellt bzw. aktualisiert werden können. Die Schablone ist dabei intern in dem Projekt-Konfigurator gespeichert und besitzt eine vorgegebene Verzeichnisstruktur.

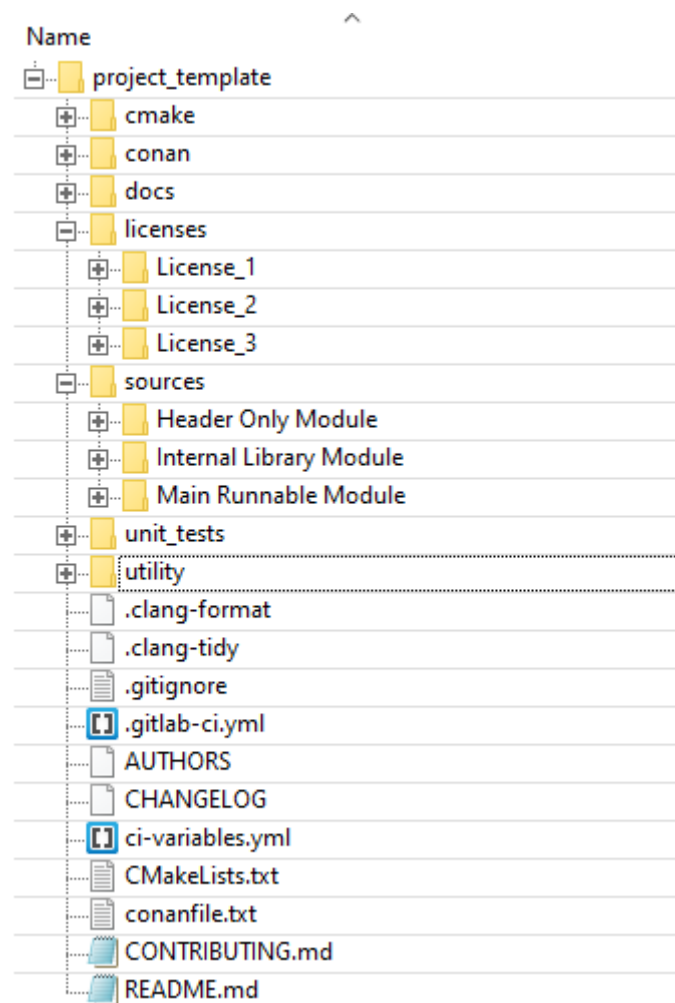


Abbildung 27: Struktur der Projekt-Schablone

Die Abbildung 27 zeigt die Struktur der Projekt-Schablone. Jedes Projekt, das mit dem Projekt-Konfigurator erstellt wird, wird auf Basis dieser Schablone erstellt. Allerdings

werden für die Erstellung einige Informationen, welche in Kapitel 4 beschrieben wurden, über das reale Projekt benötigt. Diese Informationen werden durch das Fenster für die Projekterstellung, welches in Kapitel 5.2.2 beschrieben wurde, gesammelt. Im Folgenden wird erklärt, welche Dateien der Schablone bei der Projekterstellung, mit den eingegebenen Informationen über das Projekt, konfiguriert werden:

Der lange Projektname, welcher in das Texteingabefeld 2.2 in der Abbildung 21 eingetragen werden muss, und die Projektbeschreibung, welche in das Texteingabefeld 2.6 in der Abbildung 21 eingetragen werden muss, werden für die README Datei, gekennzeichnet durch die „README.md“-Datei in der Abbildung 27, verwendet.

Der kurze Projektname, welcher in das Texteingabefeld 2.4 in der Abbildung 21 eingetragen werden muss, wird für den Namen des erstellten Projektes verwendet. Das bedeutet, dass der Ordner, der bei der Erstellung im Dateisystem erstellt wird, nach dem kurzen Projektnamen benannt ist. Außerdem wird der kurze Projektname, nach dem Build, als Name für das Bereitstellungspaket des Projektes verwendet.

Die Liste der Projekttypen, gekennzeichnet durch die DropDown-Liste 3.2 in der Abbildung 21, setzt sich aus den Namen der Unterordner zusammen, die sich in dem Quellen Order, gekennzeichnet durch den „Sources“-Ordner in der Abbildung 27, befinden. In jedem dieser Unterordner befindet sich eine „CMakeLists.txt“-Datei. Bei der Projekterstellung wird der ausgewählte Projekttyp, also der entsprechende Unterordner der Schablone, dem realen Projekt als Projektmodul hinzugefügt. Das Projektmodul ist dabei ein Unterordner in dem realen Projekt, welcher nach dem Modulnamen, aus dem Texteingabefeld 2.4 in der Abbildung 21, benannt ist. Dieser Modulname wird außerdem für die „CMakeLists.txt“-Datei.

Die Liste der Lizenztypen, gekennzeichnet durch die DropDown-Liste 3.6 in der Abbildung 21, setzt sich aus den Namen der Unterordner zusammen, die sich in dem Lizenzen Order, gekennzeichnet durch den „licenses“-Ordner in der Abbildung 27, befinden. In jedem dieser Unterordner befindet sich eine Textdatei der entsprechenden Lizenz. Bei der Projekterstellung wird die ausgewählte Lizenz, also die Textdatei des entsprechenden Unterordners der Schablone, dem realen Projekt hinzugefügt.

Die Dateien „clang-format“, „clang-tidy“, „gitignore“, „gitlab-ci.yml“, „ci-variables.yml“, „conanfile.txt“ und „CONTRIBUTING.md“ aus der Projekt-Schablone, die in Abbildung 27 dargestellt ist, sind vordefinierte Konfigurationsdateien. Diese werden bei der Projekterstellung nur dann für das neue Projekt miterstellt, wenn die entsprechenden Checkboxes, gekennzeichnet durch die Checkboxes 3.8, 3.10, 3.12, 3.14, 3.16 und 3.18 in der Abbildung 21, markiert wurden. Wenn die Checkboxes nicht markiert wurden, werden diese Konfigurationsdateien nicht in das neue Projekt übertragen.

Die Liste der Autoren, gekennzeichnet durch die Liste 4.4 in der Abbildung 21, wird bei der Projekterstellung in die Autoren Datei, gekennzeichnet durch die „AUTHORS“-Datei in der Abbildung 27, übertragen. In der Projekt-Schablone ist diese Datei leer.

Der Änderungsprotokoll Datei, gekennzeichnet durch die „CHANGELOG“-Datei in der Abbildung 27, wird bei der Projekterstellung das derzeitige Datum hinzugefügt. In der Projekt-Schablone ist diese Datei leer.

5.2 Implementierung der Software

Nachdem die Skizzen und das Klassendiagramm fertiggestellt, die Programmiersprache, sowie das Framework für die GUI ausgewählt und die Projekt-Schablone erstellt wurde, konnte die Software implementiert werden. Im weiteren Verlauf dieses Kapitels wird die Implementierung jedes Fensters beschrieben, sowie alle Funktionen der GUI, die in Kapitel 5.1 oberflächlich beschrieben wurden, mithilfe von Flussdiagrammen, detailliert erklärt.

5.2.1 Das Hauptmenü

In diesem Kapitel wird erklärt wie das Hauptmenü, welches in dem Kapitel 5.2.1 erklärt wurde, in die Software implementiert wurde und wie die Funktionen des Hauptmenüs im Detail funktionierten.

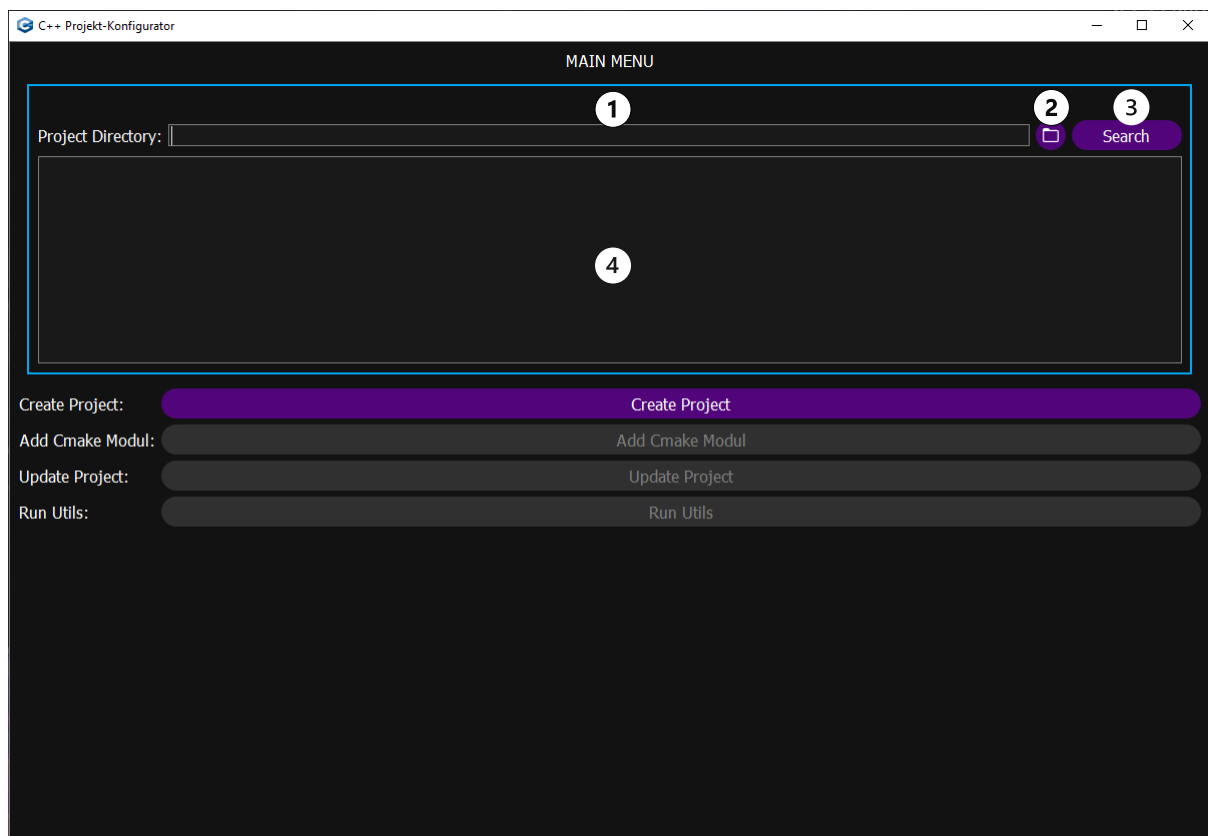


Abbildung 28: Implementierung des Hauptmenüs

Die Abbildung 28 zeigt ein Screenshot des Hauptmenüs. Dabei ist zusehen, dass das Hauptmenü nach Vorlage der Abbildung 20 des Kapitels 5.2.1 implementiert wurde, da der Aufbau des Fensters und die Benennung der Widgets unverändert von der Abbildung 20 übernommen wurden. Im Folgenden wird mithilfe eines Flussdiagramms der Suchmechanismus für existierende Projekte erklärt.

Um den Suchmechanismus zu starten, sollte als erstes der Pfad eines Ordners, in dem sich existierende Projekte befinden, in das Texteingabefeld, gekennzeichnet durch die Nummer 1 in der Abbildung 28, eingetragen werden. Dies kann manuell durchgeführt werden oder durch Benutzung eines Ordner-Dialogs. Dieser wird durch den Klick auf den Button mit dem Ordner-Icon, gekennzeichnet durch die Nummer 2 in der Abbildung 28, geöffnet. Intern wird dadurch die Methode *getExistingDirectory()* der Qt spezifischen Klasse *QFileDialog* aufgerufen, durch die sich ein klassischer Ordner- bzw. Datei-Dialog öffnet, mit dem der entsprechende Pfad ausgewählt und bestätigt werden kann.

Nachdem der Pfad eingegeben wurde, kann der Suchmechanismus durch den „Search“-Button, gekennzeichnet durch die Nummer 3 der Abbildung 28, gestartet werden.

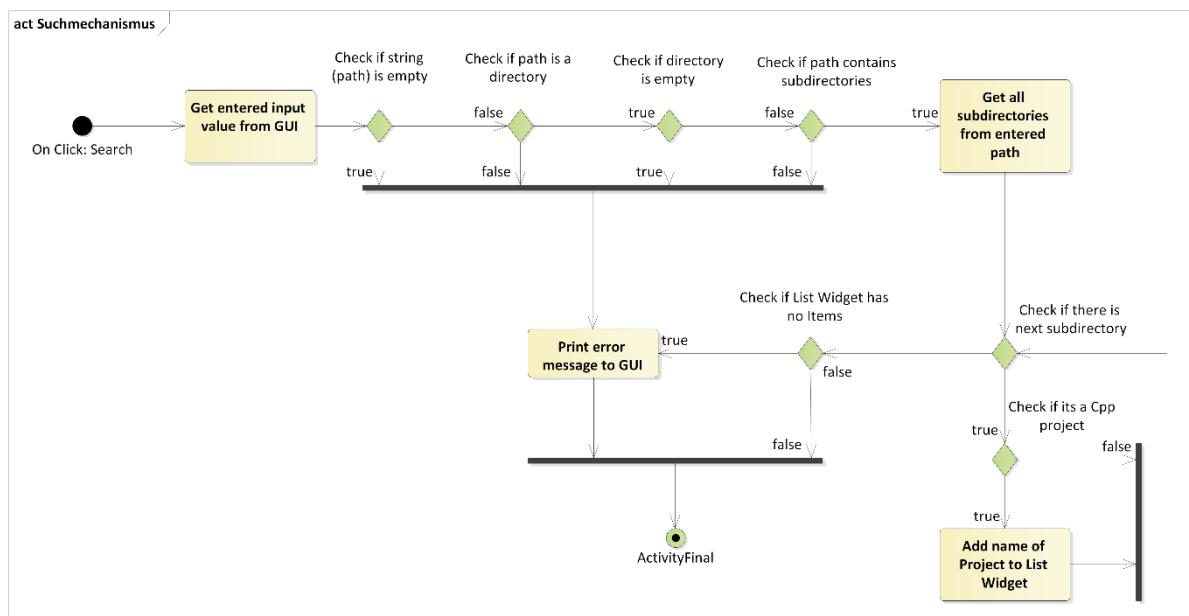


Abbildung 29: Flussdiagramm für den Suchmechanismus

In der Abbildung 29 ist der Ablauf des Suchmechanismus dargestellt. Dieser kann in drei Teilschritten erklärt werden:

Im ersten Teilschritt, der mit der Operation "Get entered input value from GUI" beginnt und mit "Check if path contains subdirectories" in der Abbildung 29 endet, wird geprüft, ob der eingegebene Wert, der in das Texteingabefeld, gekennzeichnet durch die Nummer 1 in der Abbildung 28, eingegeben wurde, ein Pfad zu einem Ordner, mit weiteren Unterordnern ist. Dazu wird der Wert intern als String gespeichert und anschließend auf vier Kriterien überprüft. Bei der ersten Prüfung wird kontrolliert, ob der eingegebene Wert ein leerer String ist, also ob der Nutzer etwas in das

Texteingabefeld eingegeben hat. Ist der String nicht leer, wird bei der zweiten Prüfung kontrolliert, ob der String ein Pfad zu einem Ordner ist. Wenn dies der Fall ist, wird als drittes kontrolliert, ob der Ordner, zu dem der eingegebene Pfad führt, leer ist. Ist der Ordner nicht leer, wird als letztes geprüft, ob der Ordner weitere Unterorder enthält. Wenn eines dieser Überprüfungen fehlschlägt, d.h. der String entweder leer ist, kein Pfad zu einem Ordner ist, der Ordner leer ist oder der Ordner keine weiteren Unterordner enthält, erscheint auf dem Hauptmenü der GUI eine Error-Nachricht mit dem Grund des Fehlschlags. Nachdem dies passiert ist, wird der weitere Verlauf des Suchmechanismus gestoppt.

Wenn die Überprüfungen erfolgreich waren, der eingegeben Wert also ein Pfad zu einem Ordner mit weiteren Unterordnern ist, wird der zweite Teilschritt des Suchungsmechanismus ausgelöst. In dem zweiten Teilschritt, der mit der Operation "Get all subdirectories from entered path" beginnt und mit "Add name of Project to List Widget" in der Abbildung 29 endet, wird überprüft, ob eines der Unterordner des Hauptordners, ein existierendes C++-Projekt ist, welches dadurch in das Listen-Widget, gekennzeichnet durch die Nummer 4 in der Abbildung 28, angezeigt wird. Dazu wird intern zuerst eine Liste aller Unterordner des Hauptordners generiert. Diese Liste besteht dabei aus den Namen der Unterordner, welche der Liste als Strings hinzugefügt wurden. Anschließend wird in einer doppelten *for-Schleife* überprüft, ob die Unterordner der Liste C++-Projekte sind. Dazu wird in der ersten *for-Schleife* die Liste, mit den Namen der Unterordner, durchiteriert. Bei jeder Iteration wird der Pfad zu dem jeweiligen Unterordner gebildet, um nachfolgend den Inhalt dieses Unterordners in einer zweiten Liste zu speichern. Die zweite Liste besteht dabei aus den Namen der Unterordner und Dateien, des jeweiligen Unterordners der ersten Liste. Nachdem die zweite Liste erstellt wurde, wird diese in der zweiten *for-Schleife* ebenfalls durch iteriert, wobei überprüft wird, ob sich in dieser Liste eine JSON-Datei, mit den Metadaten des Projektes, befindet. Falls dies der Falls ist, bedeutet das, dass der Unterordner aus der ersten Liste ein C++-Projekt ist. Dadurch wird der Name dieses Unterordners dem Listen-Widget im Hauptmenü, gekennzeichnet durch die Nummer 4 in der Abbildung 28, hinzugefügt. Dazu wird eine Instanz der Qt spezifischen Klasse *QListWidgetItem* erstellt, welchem bei der Instanziierung der Name des Unterordners als String übergeben wird. Dadurch wird ein Listenelement für das Listen-Widget erstellt, welches den Namen des Unterordners anzeigt. Die *QListWidgetItem*-Instanz wird anschließend dem Listen-Widget hinzugefügt. Falls sich

in der zweiten Liste keine JSON-Datei, mit den Metadaten des Projektes, befindet, überprüft das Programm den nächsten Unterorder der ersten Liste.

Während dem letzten Teilschritt, der sich auf die Operation "Check if List Widget has no Items" in der Abbildung 29 bezieht, wird geprüft, ob sich *QListWidgetItem*-Instanzen in dem Listen-Widget befinden. Wenn dies zutrifft, ist der Prozess des Suchmechanismus zu Ende und der Benutzer sieht dadurch mindestens ein existierendes C++-Projekt in dem Listen-Widget. Wenn das Listen-Widget leer ist, erscheint eine Error-Nachricht in der GUI mit der Nachricht, dass der eingegebene Pfad in dem Texteingabefeld, gekennzeichnet durch die Nummer 2 in der Abbildung 28, kein C++-Projekt enthält.

5.2.2 Fenster für die Projekterstellung

In diesem Kapitel wird erklärt wie das Fenster für die Projekterstellung, welches in dem Kapitel 5.2.2 erklärt wurde, in die Software implementiert wurde und wie die Projekterstellung im Detail abläuft.

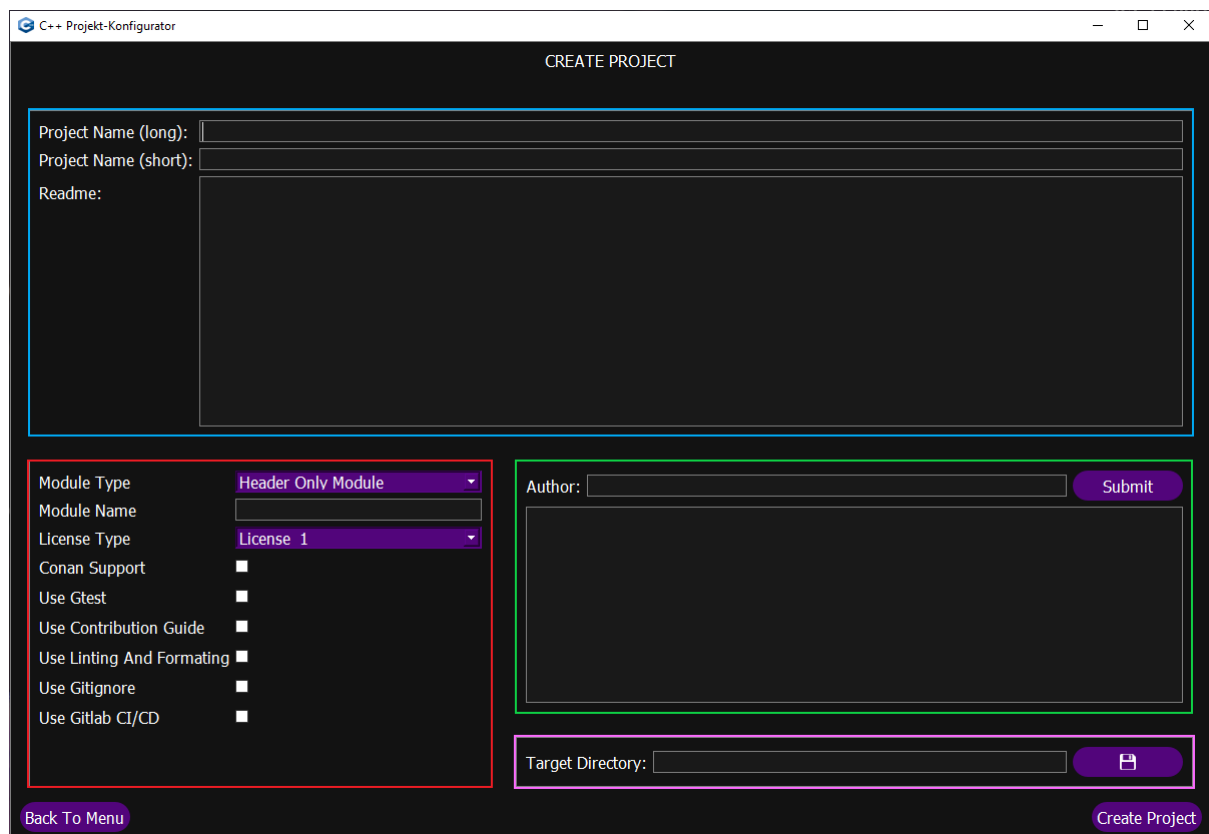


Abbildung 30: Implementierung des Fensters für die Projekterstellung

Die Abbildung 30 zeigt ein Screenshot des Fensters für die Projekterstellung. Dabei ist zusehen, dass dieses Fenster nach Vorlage der Abbildung 21 des Kapitels 5.2.2

implementiert wurde, da der Aufbau des Fensters und die Benennung der Widgets unverändert von der Abbildung 21 übernommen wurden. Im Folgenden wird mithilfe von verschiedenen Flussdiagrammen erklärt, wie ein Projekt durch den Projekt-Konfigurator erstellt wird.

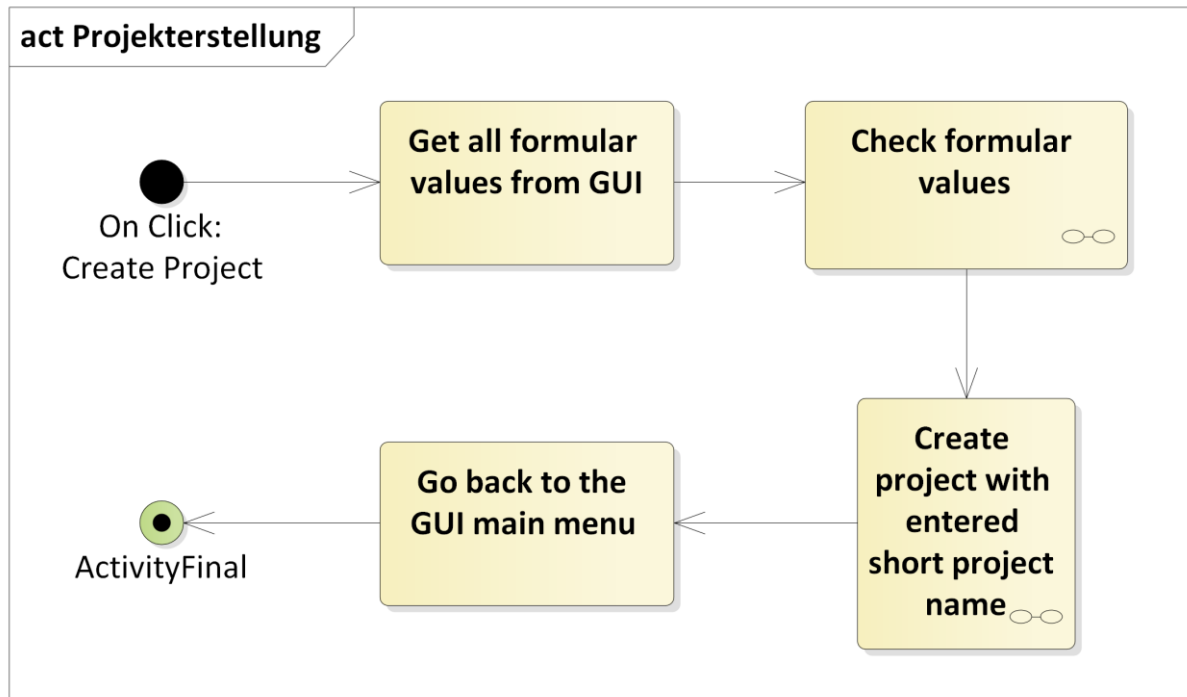


Abbildung 31: Flussdiagramm für die Projekterstellung

In der Abbildung 31 ist der Ablauf der Projekterstellung dargestellt. Dieser kann in vier Teilschritten erklärt werden:

Mit dem Klick auf den „Create Project“-Button, zusehen in der Abbildung 21, wird intern der Prozess für die Projekterstellung gestartet. Im ersten Teilschritt dieses Prozesses, gekennzeichnet durch die Operation „Get all formular values from GUI“ in der Abbildung 31 bezieht, speichert das Programm folgende Werte aus dem Formular für die Projekterstellung zusehen in der Abbildung 30, als Strings:

- Der lange Projektname – einzugeben in dem „Project Name (long)“-Texteingabefeld, gekennzeichnet als erstes Texteingabefeld in dem blau markierten Teil der Abbildung 30
- Der kurze Projektname – einzugeben in dem „Project Name (short)“-Texteingabefeld, gekennzeichnet als zweites Texteingabefeld in dem blau markierten Teil der Abbildung 30

- Die Projektbeschreibung – einzugeben in dem „Readme“-Texteingabefeld, gekennzeichnet als drittes Texteingabefeld in dem blau markierten Teil der Abbildung 30
- Der Modulname – einzugeben in dem „Module Name“-Texteingabefeld, gekennzeichnet als Texteingabefeld in dem rot markierten Teil der Abbildung 30
- Der Speicherort des Projektes – einzugeben in dem „Target Directory“-Texteingabefeld, gekennzeichnet als Texteingabefeld in dem pink markierten Teil der Abbildung 30

In dem zweiten Teilschritt, der sich auf die Operation „Check formular values“ in der Abbildung 31 bezieht, werden die gespeicherten Werte des ersten Teilschritts überprüft. Diese Überprüfung wird mithilfe eines zusätzlichen Hilfsdiagrammes erklärt:

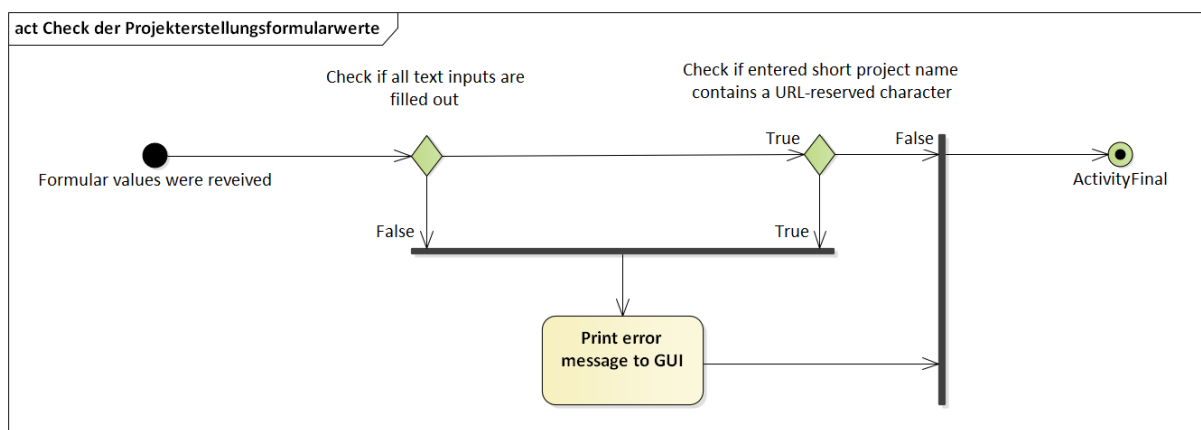


Abbildung 32: Flussdiagramm für den Check der Projekterstellungsformularwerte

In der Abbildung 32 ist der Überprüfungsprozess der eingegebenen Werte bzw. der Ablauf des zweiten Teilschritts des gesamten Projekterstellungsprozesses, dargestellt in der Abbildung 31, zusehen. Dabei wird als erstes, zu sehen als Operation „Check if all text inputs are filled out“ in der Abbildung 32, überprüft, ob die gespeicherten Strings aus dem ersten Teilschritt leere Strings sind, bzw. ob in einem Texteingabefeld kein Wert eingegeben wurde. Diese Überprüfung erfolgt nacheinander für jeden String. Ist ein String leer, wird eine entsprechende Error-Nachricht in der GUI ausgegeben und der gesamte Prozess der Projekterstellung gestoppt.

Wenn die Strings nicht leer sind, bzw. in jedes Texteingabefeld ein Wert eingegeben wurde, wird überprüft, ob der kurze Projektname ein unzulässiges Zeichen enthält, zusehen als Operation „Check if entered short project name contains a URL-reserved

character“ in der Abbildung 32. Diese Zeichen beziehen sich dabei auf die reservierten Zeichen einer URL. Dazu erstellt das Programm eine Liste mit allen reservierten Zeichen einer URL, wobei die Zeichen als Strings in der Liste gespeichert sind. Anschließend wird kontrolliert, ob sich eines dieser Zeichen in der Liste in dem String des kurzen Projektnamens befindet. Falls dies der Fall ist, wird eine entsprechende Error-Nachricht in der GUI ausgegeben und der gesamte Prozess der Projekterstellung gestoppt.

Nachdem alle Werte überprüft wurden, startet der dritte Teilschritt, der sich auf die Operation „Create project with entered short project name“ in der Abbildung 31 bezieht. In diesem Teilschritt wird das neue Projekt erstellt und dabei alle Dateien, die in dem letzten Kapitel 6.1 erklärt wurden, konfiguriert. Dieser Prozess wird mithilfe eines weiteren Hilfsdiagramm erklärt:

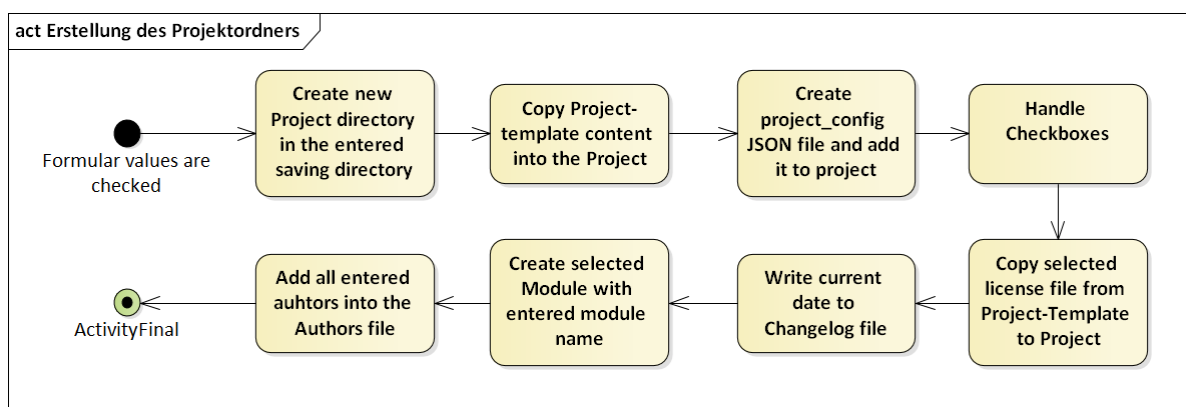


Abbildung 33: Flussdiagramm für Erstellung des Projektordners

In der Abbildung 33 ist der Ablauf für die Erstellung des neuen Projektes und die Konfiguration der entsprechenden Dateien bzw. der dritte Teilschritt des gesamten Projekterstellungsprozesses, dargestellt in der Abbildung 31, zusehen. Dabei wird als erstes, zusehen als Operation „Create new Project directory in the entered saving directory“ in der Abbildung 33, ein neues Projekt bzw. ein neuer Ordner in dem eingegebenen Speicherort erstellt. Der Speicherort wurde dafür als Pfad in das „Target Directory“-Texteingabefeld, gekennzeichnet als Texteingabefeld des pink markierten Teils in der Abbildung 30, eingeben. Das neue Projekt bzw. der neue Ordner besitzt dabei den Namen, der als Wert in das „Project Name (short)“-Texteingabefeld, gekennzeichnet als zweites Texteingabefeld des blau markierten Teils in der Abbildung 30, eingeben wurde.

Als nächstes, zusehen als Operation „Copy Project-template content into the Project“ in der Abbildung 33, kopiert das Programm den Inhalt der Projekt-Schablone, die in dem Kapitel 6.1 erklärt wurde, in das neue erstellte Projekt. Nachfolgend wird sowohl der Lizenzen-Unterordner, gekennzeichnet durch den „licenses“-Unterordner in der Abbildung 27 des Kapitels 6.1, als auch der Inhalt des Quellen-Unterordner, gekennzeichnet durch des „sources“-Unterordner in der Abbildung 27 des Kapitels 6.1, aus dem neuen Projekt gelöscht. Der Quellen-Unterordner bleibt allerdings in dem neuen Projekt bestehen.

Danach, gekennzeichnet durch die Operation „Create project_config JSON file and add it to project“ in der Abbildung 33, erstellt das Programm eine JSON-Datei, in der alle Metadaten des Projektes gespeichert sind, und fügt diese dem Projekt hinzu. Die JSON-Datei wird mithilfe eines Python Dictionary erstellt, das gleich wie die JSON-Datei aufgebaut ist. Die Abbildung 34 zeigt eine beispielhafte JSON-Datei für ein Test-Projekt. Die booleschen Werte sind dabei die Status der Checkboxen, die in dem rot markierten Teil in der Abbildung 30 dargestellt sind.

```
{
  "Projektname_lang": "Test_lang",
  "Projektname_kurz": "Test_kurz",
  "Readme": "Test_description",
  "Module_typ": "Test_Module",
  "Lizenz_typ": "License_3",
  "conan_support": true,
  "gtest": true,
  "contribution_guide": true,
  "linting_and_formatting": true,
  "gitignore": true,
  "gitlab_ci_cd": true
}
```

Abbildung 34: Projekt-Konfiguration JSON-Datei

Mithilfe des Python Dictionary wird als nächstes, zusehen als Operation „Handle Checkboxes“ in der Abbildung 33, der Status der Checkboxen, gekennzeichnet durch die Checkboxen in dem rot markierten Teil in der Abbildung 30, überprüft. Wie in Kapitel 6.1 erklärt, werden dem neuen Projekt, durch Markieren der Checkboxen, ein oder mehrere vordefinierte Konfigurationsdateien hinzugefügt. Durch die Kopie der Projekt-Schablone besitzt das neue Projekt allerdings schon alle Konfigurationsdateien – auch diejenigen, die in dem neuen Projekt nicht benötigt werden. Deshalb wird bei der Überprüfung nur nach unmarkierten Checkboxen gesucht. Die Überprüfung erfolgt dabei nacheinander für jede Checkbox. Falls eine

Checkbox unmarkiert ist, der Status der Checkbox also *False* ist, werden die entsprechenden Konfigurationsdateien aus dem neuen Projekt gelöscht.

Im Anschluss daran, zusehen als Operation „Copy selected license file from Project-Template to Project“ in der Abbildung 33, überprüft das Programm welcher Wert in der „License Type“-DropDown-Liste, gekennzeichnet durch die zweite DropDown-Liste in dem rot markierten Teil in der Abbildung 30, ausgewählt wurde. Dieser Wert entspricht dem gleichnamigen Lizenztyp-Ordner, gekennzeichnet durch die „License_1“- / „License_2“- und „License_3“-Ordner in der Abbildung 27 des Kapitels 6.1, in dem Lizenzen-Ordner, gekennzeichnet durch den „licenses“-Unterordner in der Abbildung 27 des Kapitels 6.1, der Projekt-Schablone. Der Inhalt dieses Lizenztyp-Ordners wird anschließend in das neue Projekt kopiert.

Als nächstes, zusehen als Operation „Write current date to Changelog file“ in der Abbildung 33, öffnet das Programm die Änderungsprotokoll-Datei, gekennzeichnet durch die „CHANGELOG“-Datei in der Abbildung 27 des Kapitels 6.1, des neuen Projektes, fügt ihr das Datum der Projekterstellung hinzu und schließt die Datei anschließend wieder.

Danach, zusehen als Operation „Create selected Module with entered module name“ in der Abbildung 33, erstellt das Programm ein neues Projektmodul in dem neuen Projekt. Dazu wird zuerst geprüft welcher Wert in der Projekttyp-DropDown-Liste, gekennzeichnet durch die „Project Type“-DropDown-Liste in Abbildung 30, ausgewählt wurde. Dieser Wert entspricht dem gleichnamigen Projekttyp-Ordner, gekennzeichnet durch die „Header Only Module“- / „Internal Library Module“- und „Main Runnable Module“-Ordner in der Abbildung 27 des Kapitels 6.1, in dem Quellen-Ordner, gekennzeichnet durch den „sources“-Unterordner in der Abbildung 27 des Kapitels 6.1, der Projekt-Schablone. Der entsprechende Projekttyp-Ordner wird anschließend, inklusive Inhalt, in den „sources“-Ordner des neuen Projektes kopiert.

Als letztes, zusehen als Operation „Add all entered authors into the Authors file“ in der Abbildung 33, öffnet das Programm die Autoren-Datei, gekennzeichnet durch die „AUTHORS“-Datei in der Abbildung 27 des Kapitels 6.1, fügt ihr alle Autorennamen, die in dem Listen-Widget des grün markierten Teils der Abbildung 30 aufgelistet sind, hinzu und schließt die Datei anschließend wieder. Dadurch endet der dritte Teilschritt

des gesamten Projekterstellungsprozesses, welcher in der Abbildung 31 dargestellt ist.

In dem vierten Teilschritt, der sich auf die Operation „Go back to the GUI main menu“ in der Abbildung 31 bezieht, wechselt die GUI das Fenster wieder zurück auf das Hauptmenü, wodurch der Prozess der Projekterstellung beendet wird.

5.2.3 Fenster für die Projektaktualisierung

In diesem Kapitel wird erklärt wie das Fenster für die Projektaktualisierung, welches in dem Kapitel 5.2.3 erklärt wurde, in die Software implementiert wurde und wie die Projektaktualisierung im Detail abläuft.

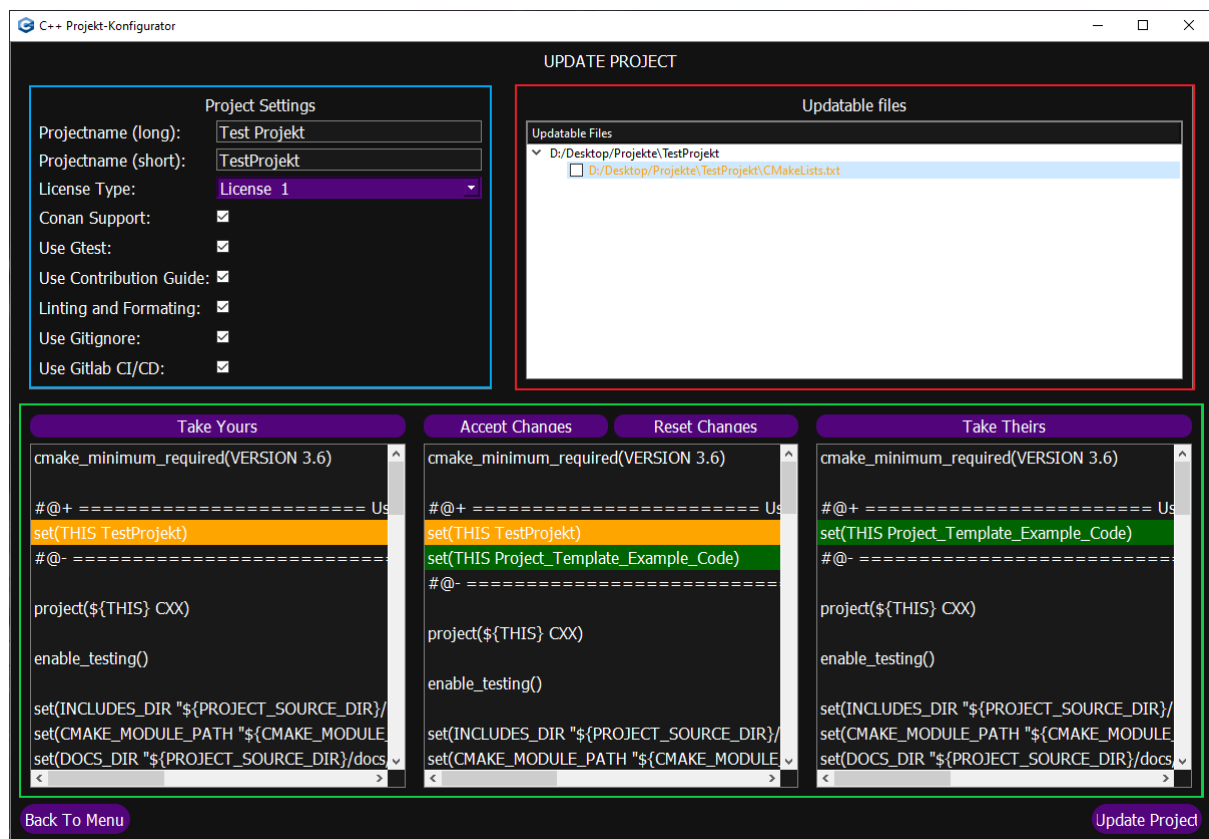


Abbildung 35: Implementierung des Fensters für die Projektaktualisierung

Die Abbildung 35 zeigt ein Screenshot des Fensters für die Projektaktualisierung. Dabei ist zusehen, dass dieses Fenster nach Vorlage der Abbildung 22 des Kapitels 5.2.3 implementiert wurde, da der Aufbau des Fensters und die Benennung der Widgets unverändert von der Abbildung 22 übernommen wurden. Im Folgenden wird mithilfe von verschiedenen Flussdiagrammen erklärt, wie ein Projekt durch den Projekt-Konfigurator aktualisiert wird.

Um ein Projekt aktualisieren zu können, muss das Programm, nach dem Klick auf den „Update Project“-Button, gekennzeichnet durch den „Update Project“-Button in der Abbildung 28 des Kapitels 6.2.1, im Hauptmenü, die Projekteinstellungen, gekennzeichnet durch den blau markierten Teil in der Abbildung 35, konfigurieren und die aktualisierbaren Dateien des Projektes der „Updatables Files“-Liste, gekennzeichnet durch den rot markierten Teil in der Abbildung 35, hinzufügen. Erst nachdem die beiden Teile des Fensters entsprechend konfiguriert wurden, wird das Fenster für die Projektaktualisierung in der GUI gezeigt.

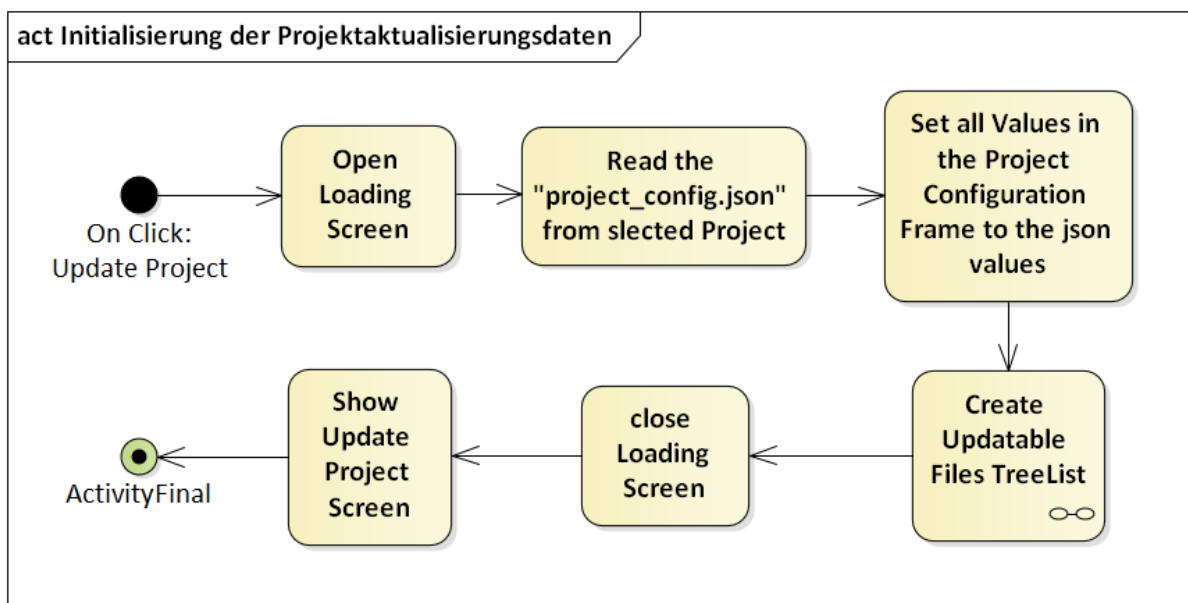


Abbildung 36: Flussdiagramm der Initialisierung der Projektaktualisierungsdaten

In der Abbildung 36 ist der Ablauf dargestellt, durch den der blau und rot markierte Teil der Abbildung 35, bei dem Wechsel auf dieses Fenster, konfiguriert werden. Dieser Prozess kann in sechs Teilschritten erklärt werden:

Im ersten Teilschritt, der sich auf die Operation „Open Loading Screen“ in der Abbildung 36 bezieht, wechselt die GUI das Fenster auf den Ladebildschirm und startet anschließend einen neuen Thread, in dem die vier nachfolgenden Teilschritte ausgeführt werden.

Im zweiten Teilschritt, der sich auf die Operation „Read the „project__config.json“ from selected Project“ in der Abbildung 36 bezieht, liest das Programm die JSON-Datei ein, in der die Metadaten des Projektes stehen. Dazu generiert das Programm den Pfad zu der entsprechenden Datei, öffnet diese, liest diese und speichert Inhalt der Datei intern als Python Dictionary.

In dem dritten Teilschritt, der sich auf die Operation „Set all Values in the Project Configuration Frame to the json values“ in der Abbildung 36 bezieht, setzt das Programm in dem Teil für die Projekteinstellungen des Fensters, gekennzeichnet durch den blau markierten Teil in der Abbildung 35, alle Werte auf die entsprechenden Werte der JSON-Datei bzw. dem Python Dictionary.

Nachdem die Werte der Projekteinstellungen gesetzt wurden, startet der vierte Teilschritt, der sich auf die Operation „Create Updatable Files TreeList“ in der Abbildung 36 bezieht. In diesem Teilschritt wird der „Updatable Files“-Liste, gekennzeichnet durch den roten Teil in der Abbildung 35, die Pfade aller aktualisierbaren Dateien des Projektes als Listenelemente hinzugefügt. Dieser Prozess wird mithilfe eines zusätzlichen Hilfsdiagramm erklärt:

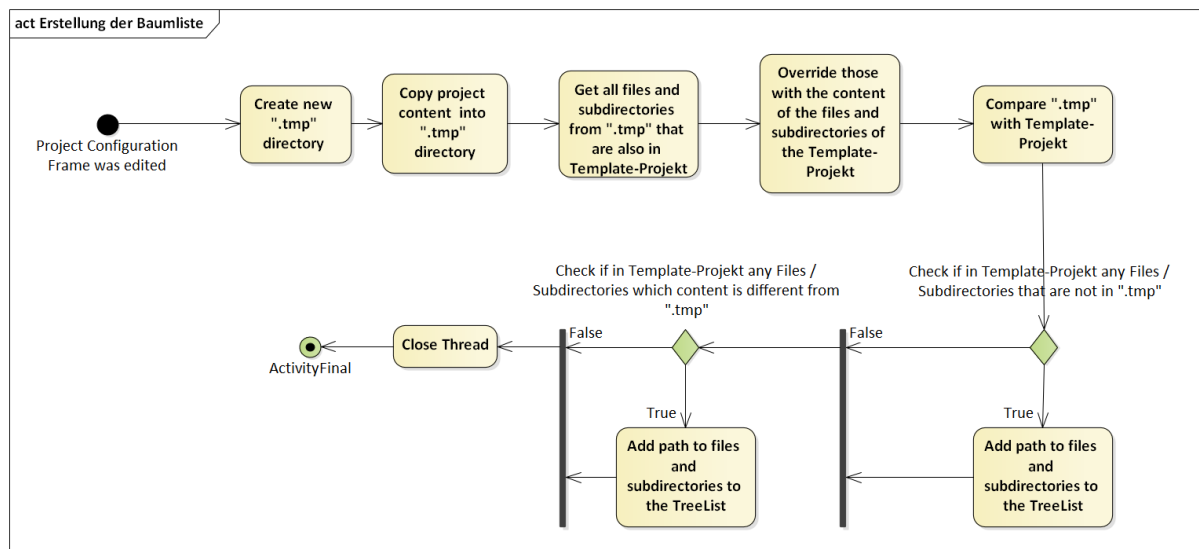


Abbildung 37: Flussdiagramm für die Erstellung der Baumliste

In der Abbildung 37 ist der Ablauf dargestellt, durch den der „Updatable Files“-Liste die Pfade aller aktualisierbaren Dateien als Listenelemente hinzugefügt werden. Als erstes, zusehen als Operation „Create new „.tmp“ directory“ in der Abbildung 37, erstellt das Programm einen temporären „.tmp“-Ordner.

Nach dessen Erstellung wird der Inhalt des ausgewählten Projektes in den „.tmp“-Ordner kopiert, zusehen als „Copy project content into „.tmp“ directory“ in der Abbildung 37.

Anschließend, zusehen als Operation „Get all files and subdirectories from „.tmp“ that are also in Template-Projekt“ in der Abbildung 37, speichert das Programm jeden Pfad

der Dateien und Unterordner des „.tmp“-Ordners, die sich auch in der Projekt-Schablone befinden, intern in einer Liste.

Im Anschluss, zusehen als Operation „Override those with the content of the files and subdirectories of the Template-Projekt“ in der Abbildung 37, daran überschreibt das Programm den Inhalt der Dateien und Unterordner aus der Liste, mit dem Inhalt der gleichen Dateien und Unterordner aus der Projekt-Schablone.

Als nächstes, zusehen als Operation „Compare „.tmp“ with Template-Projekt“ in der Abbildung 37, vergleicht das Programm den „.tmp“-Ordner mit der Projekt-Schablone. Dabei wird als erstes überprüft, zusehen als Operation „Check if in Template-Projekt any Files / Subdirectories that are not in „.tmp““ in der Abbildung 37, ob sich in der Projekt-Schablone Dateien oder Unterordner befinden, die sich nicht in dem „.tmp“-Ordner vorhanden sind. Falls das der Fall ist, wurden diese Dateien oder Unterordner nach der Erstellung des Projektes in die Projekt-Schablone hinzugefügt. Deswegen werden die Pfade zu diesen Dateien oder Unterordnern der „Updatables Files“-Liste, gekennzeichnet durch den rot markierten Teil in der Abbildung 35, als grüne Listenelemente hinzugefügt. Dazu wird eine *QListWidgetItem*-Instanz, mit dem Pfad zu der entsprechenden Datei oder zu dem entsprechenden Unterordner, als dessen Namen, erstellt, der angezeigte Text dieser Instanz auf die Farbe Grün gesetzt und dem Listen-Widget hinzugefügt. Falls es keine Datei oder kein Unterordner gibt, welches der Projekt-Schablone nachträglich hinzugefügt wurde, wird dem Listen-Widget kein Pfad hinzugefügt.

Als zweites wird, während dem Vergleich zwischen dem „.tmp“-Ordner und der Projekt-Schablone überprüft, zusehen als Operation „Check if in Template-Projekt any Files / Subdirectories which content is different from „.tmp““ in der Abbildung 37, ob es Dateien oder Unterordner gibt, die sich sowohl in dem „.tmp“-Ordner, als auch in der Projekt-Schablone befinden und deren Inhalt sich unterscheidet. Falls das der Fall ist, wurden diese Dateien oder Unterordner nach der Erstellung des Projektes in der Projekt-Schablone verändert bzw. erweitert. Deswegen werden die Pfade zu diesen Dateien oder Unterordnern der „Updatables Files“-Liste, gekennzeichnet durch den rot markierten Teil in der Abbildung 35, als orangene Listenelemente hinzugefügt. Dazu wird eine *QListWidgetItem*-Instanz, mit dem Pfad zu der entsprechenden Datei oder zu dem entsprechenden Unterordner, als dessen Namen, erstellt, der angezeigte Text dieser Instanz auf die Farbe Orange gesetzt und der „Updatables Files“-Liste

hinzugefügt. In dem rot markierten Teil in der Abbildung 35 ist beispielhaft ein solches Listenelement zusehen. Falls es keine Datei oder kein Unterordner gibt, welches nachträglich in der Projekt-Schablone verändert bzw. erweitert wurde, wird dem Listen-Widget kein Pfad hinzugefügt.

Nachdem die Pfade aller aktualisierbaren Dateien des Projektes, der „Updatables Files“-Liste als Listenelemente hinzugefügt wurden, wird im fünften Teilschritt, der sich auf die Operation „Close Loading Screen“ in der Abbildung 36 bezieht, der Ladebildschirm geschlossen und der neue Thread, in dem die letzten Teilschritte ausgeführt wurden, geschlossen.

Im letzten Teilschritt, der sich auf die Operation „Show Update Project Screen“ in der Abbildung 36 bezieht, wird das konfigurierte Fenster für die Projektaktualisierung in der GUI angezeigt, wodurch der Prozess für die Konfigurierung der ersten beiden Teile des Fensters beendet wird.

Wenn das ausgewählte Projekt veraltet ist, wurden dem Listen-Widget dieses Fensters verschiedene Pfade als Listenelemente hinzugefügt. Diese Listenelemente können nun von den Benutzern ausgewählt werden, um die entsprechende Datei mithilfe der drei Listen, dargestellt durch den grün markierten Teil in der Abbildung 35, zu konfigurieren. Damit die Benutzer die drei Listen nutzen kann, müssen diese zuerst von dem Splitter Programm erstellt werden. Das bedeutet, dass die drei Listen nach dem Aufruf des Fensters nicht existieren. Erst nach der Auswahl eines Listenelementes werden die drei Listen mit den Informationen des ausgewählten Elementes erstellt und angezeigt. Nachfolgend wird mithilfe eines Flussdiagramms erklärt, wie diese Listen erstellt werden.

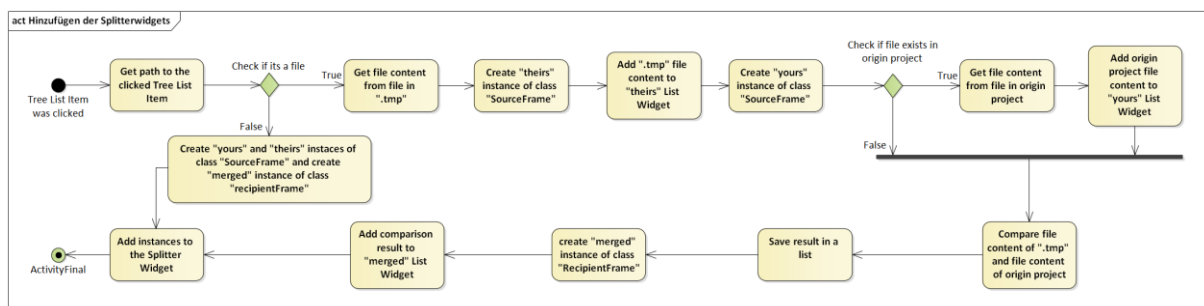


Abbildung 38: Flussdiagramm für das Hinzufügen der Splitterwidgets

In der Abbildung 38 ist der Ablauf dargestellt, durch den die Listen, gekennzeichnet durch den grün markierten Teil in der Abbildung 35, erstellt werden, nachdem ein

Listenelement in der „Updatables Files“-Liste, gekennzeichnet durch den rot markierten Teil in der Abbildung 35, ausgewählt wurde. Dieser Prozess kann in vier Teilschritten erklärt werden:

Im ersten Teilschritt, der mit der Operation "Get path to the clicked Tree List Item" beginnt und mit "Add „tmp“ file content to „theirs“ List Widget" in der Abbildung 38 endet, speichert das Programm den Pfad, der auf dem ausgewählten Listenelement angezeigt wird. Anschließend wird überprüft, ob es sich bei diesem Pfad um eine Datei handelt. Wenn dies zutrifft, öffnet das Programm diese Datei in dem „tmp“-Ordner und speichert den Inhalt dieser Datei in eine interne Liste. Danach erstellt das Programm eine „theirs“-Instanz der Klasse *SourceFrame*. Diese Instanz ist für den rechten Teil, gekennzeichnet durch den „Take Theirs“-Button und dem darunterliegenden Listen-Widget in dem grün markierten Teil in der Abbildung 35, zuständig. Nachdem die „theirs“-Instanz erstellt wurde, wird die Liste, in der der Inhalt der Datei gespeichert wurde, dem Listen-Widget dieser Instanz hinzugefügt. Dadurch wird in dem Listen-Widget der Inhalt der Datei, die sich in dem „tmp“-Ordners befindet, dargestellt.

Im zweiten Teilschritt, der mit der Operation "Create „yours“ instance of class „SourceFrame“" beginnt und mit "Add origin project file content to „yours“ List Widget" in der Abbildung 38 endet, erstellt das Programm eine „yours“-Instanz der Klasse *SourceFrame*. Diese Instanz ist für den linken Teil, gekennzeichnet durch den „Take Yours“-Button und dem darunterliegenden Listen-Widget in dem grün markierten Teil in der Abbildung 35, zuständig. Anschließend wird überprüft, ob die Datei des ausgewählten Listenelementes auch in dem ausgewählten Projekt existiert. Falls das zutrifft, öffnet das Programm die Datei in dem Projekt und speichert den Inhalt dieser Datei ebenfalls in eine interne Liste. Danach wird die Liste, in der der Inhalt der Datei gespeichert wurde, dem Listen-Widget dieser Instanz hinzugefügt. Dadurch wird in dem Listen-Widget der Inhalt der Datei, die sich in dem ausgewählten Projekt befindet, dargestellt. Falls die Datei des ausgewählten Listenelementes nicht in dem Projekt existiert, bleibt das Listen-Widget der „yours“-Instanz leer. Dadurch wird dem Benutzer signalisiert, dass die Datei nur in der Projekt-Schablone existiert.

Im dritten Teilschritt, der mit der Operation "Compare file content of „tmp“ and file content of origin project" beginnt und mit "Add comparison result to „merged“ List Widget" in der Abbildung 38 endet, vergleicht das Programm den Inhalt der beiden Listen, in denen der Inhalt der Dateien des „tmp“-Ordners und des

ausgewählten Projektes gespeichert ist. Die Liste mit dem Inhalt der Datei aus dem Projekt ist dabei leer, falls die Datei nicht in dem Projekt existiert. Das Ergebnis des Vergleichs wird in einer Liste gespeichert. Nach dem Vergleich erstellt das Programm eine „merged“-Instanz der Klasse „RecipientFrame“. Diese Instanz ist für den mittleren Teil, gekennzeichnet durch den „Accept Changes“-Button, den „Reset Changes“-Button und dem darunterliegenden Listen-Widget in dem grün markierten Teil in der Abbildung 35, zuständig. Nachdem die „merged“-Instanz erstellt wurde, wird die Liste, in der das Ergebnis des Vergleichs gespeichert wurde, dem Listen-Widget dieser Instanz hinzugefügt. Dadurch wird in dem Listen-Widget die Differenz der beiden Dateien dargestellt.

Falls der Pfad des ausgewählten Listenelements aus der „Updatables Files“-Liste, gekennzeichnet durch den rot markierten Teil in der Abbildung 35, keine Datei, sondern ein Ordner ist, werden die drei Instanzen mit leeren Listen-Widgets erstellt, da kein Datei-Inhalt dargestellt werden kann.

Im letzten Teilschritt, der sich auf die Operation „Add instances to the Splitter Widget“ in der Abbildung 38 bezieht, werden die drei erstellten Instanzen dem unteren Teil des Fensters, gekennzeichnet durch den grün markierten Teil in der Abbildung 35, hinzugefügt. Dieser untere Teil ist dabei ein spezielles Splitter-Objekt, welchem Widgets hinzugefügt werden können, die anschließend in der gleichen Reihenfolge, in der diese hinzugefügt worden, in einer horizontalen Ansicht angezeigt werden. Nachdem die Instanzen dem Splitter-Objekt hinzugefügt wurden, ist der gesamte Prozess, der in der Abbildung 38 dargestellt ist, beendet, wodurch die Benutzer die ausgewählte Datei konfigurieren können.

Wichtig dabei ist, dass die Konfigurationen nur auf das des Listen-Widgets der „merged“-Instanz angewendet werden können, da der Inhalt dieses Listen-Widgets den Inhalt der aktualisierten Datei des Projektes darstellt. Eine Datei kann auf drei Arten konfiguriert werden:

Die erste Art ist durch den „Take Yours“-Button. Durch Drücken dieses Buttons wird der Inhalt des Listen-Widgets der „merged“-Instanz, mit dem Inhalt des Listen-Widgets der „yours“-Instanz überschrieben. Das bedeutet, dass der Inhalt der Datei aus dem Projekt nicht verändert bzw. aktualisiert werden soll.

Die zweite Art ist durch den „Take Theirs“-Button. Durch Drücken dieses Buttons wird der Inhalt des Listen-Widgets der „merged“-Instanz, mit dem Inhalt des Listen-Widgets der „theirs“-Instanz überschrieben. Das bedeutet, dass der Inhalt der Datei aus dem Projekt mit dem Inhalt der gleichen Datei aus der Projekt-Schablone aktualisiert bzw. überschrieben werden soll.

Die letzte Art ist eine benutzerdefinierte Konfigurierung des Inhaltes des Listen-Widgets der „merged“-Instanz, durch Drag and Drop. Dabei können nur die farbigen Listenelemente aus allen Instanzen gezogen werden. Diese können anschließend nur in das Listen-Widget der „merged“-Instanz fallengelassen werden. Diese Funktionalität bietet dem Benutzer die Möglichkeit, einige benutzerdefinierte Änderungen in der Originaldatei beizubehalten, sie aber dennoch mit neuen Funktionen aus der Vorlage zu aktualisieren.

Nachdem die Konfiguration einer Datei beendet wurde, muss der Inhalt des Listen-Widgets der „merged“-Instanz gespeichert werden. Dazu muss der „Accept Changes“-Button, gekennzeichnet durch den gleichnamigen Button in dem grün markierten Teil in der Abbildung 35, gedrückt werden. Wenn dieser Button gedrückt wird, speichert das Programm den Inhalt des Listen-Widgets intern in ein Python Dictionary. Der Schlüssel des Dictionarys ist dabei der Pfad zu der Datei in dem „.tmp“-Ordner. Der Wert des Schlüssels ist eine Liste, in der der konfigurierte Inhalt des Listen-Widgets gespeichert ist. Immer wenn der „Accept Changes“-Button gedrückt wird, wird das Python Dictionary mit dem entsprechenden Pfad und Inhalt der ausgewählten Datei aus der „Updatables Files“-Liste, gekennzeichnet durch den rot markierten Teil in der Abbildung 35, erweitert. Außerdem wird durch Drücken des „Accept Changes“-Button, die Checkbox, des ausgewählten Listenelements der „Updatables Files“-Liste markiert. Nur wenn die Checkboxes aller Listenelemente markiert wurden, kann das Projekt im späteren Verlauf mit den vorgenommenen Änderungen aktualisiert werden.

Falls gespeicherte Änderungen einer Datei, zurückgesetzt werden müssen, kann dazu der „Reset Changes“-Button, gekennzeichnet durch den gleichnamigen Button in der Abbildung 35, benutzt werden. Durch den Klick auf den „Reset Changes“-Button löscht das Programm den Pfad und den Inhalt aus dem Python Dictionary, in dem die Änderungen einer Datei intern gespeichert werden und setzt den Inhalt des Listen-Widgets der „merged“-Instanz auf den Erstzustand zurück. Außerdem wird durch

Drücken des „Reset Changes“-Buttons, die Checkbox des ausgewählten Listenelements der „Updatables Files“-Liste wieder entmarkiert.

Nachdem für jedes Listenelement bzw. für jede Datei der „Updatables Files“-Liste eine Version gespeichert wurde, kann durch klicken des „Update Project“-Buttons, gekennzeichnet durch den gleichnamigen Button in der Abbildung 35, das ausgewählte Projekt aktualisiert werden.

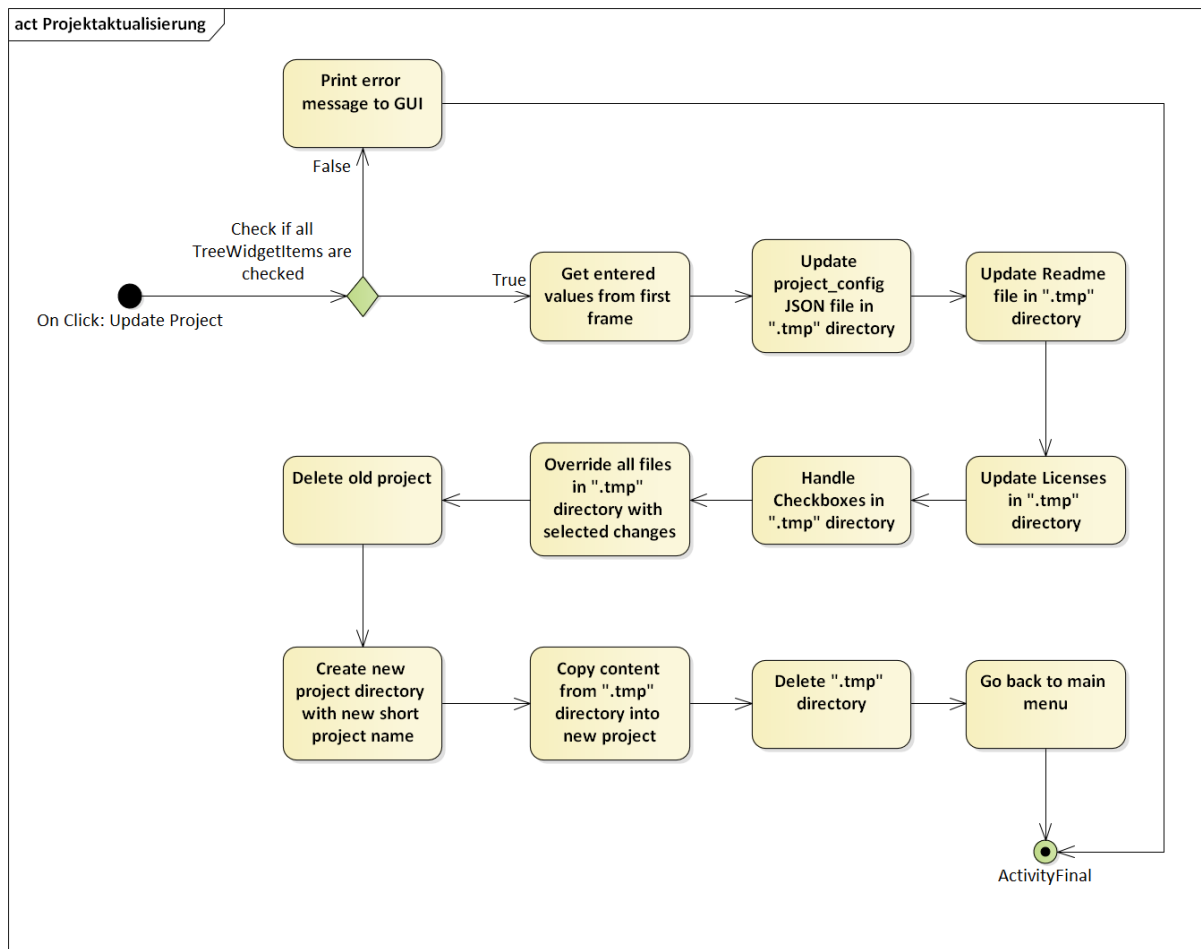


Abbildung 39: Flussdiagramm für die Projektaktualisierung

In der Abbildung 39 ist der Ablauf der Projektaktualisierung dargestellt. Dieser Prozess kann in neun Teilschritten erklärt werden:

Im ersten Teilschritt, der sich auf die Operation „Check if all TreeWidgetItems are checked“ in der Abbildung 39 bezieht, speichert das Programm alle Listenelemente der „Updatables Files“-Liste in eine interne Liste. Anschließend iteriert das Programm durch diese Liste und überprüft dabei, ob die Checkbox jedes Listenelements markiert ist. Die Markierung der Checkbox signalisiert dabei, dass eine Änderung für die entsprechende Datei des Listenelements, durch den „Accept Changes“-Button,

gespeichert wurde. Falls mindestens eine Checkbox nicht markiert wurde, wird eine entsprechende Error-Nachricht in der GUI ausgegeben und der gesamte Prozess der Projektaktualisierung wird gestoppt. Andernfalls startet der zweite Teilschritt dieses Prozesses.

Im zweiten Teilschritt, der sich auf die Operation „Get entered values from first frame“ in der Abbildung 39 bezieht, speichert das Programm intern folgende Werte aus dem ersten Teil, gekennzeichnet durch den blau markierten Teil in der Abbildung 35, des Fensters:

- Langer Projektname – einzugeben in dem „Projectname (long)“-Texteingabefeld, gekennzeichnet durch das erste Texteingabefeld in dem blau markierten Teil in der Abbildung 35
- Kurzer Projektname – einzugeben in dem „Projectname (short)“-Texteingabefeld, gekennzeichnet durch das zweite Texteingabefeld in dem blau markierten Teil in der Abbildung 35
- Der Lizenztyp – auszuwählen aus der „License Typ“-DropDown-Liste, gekennzeichnet durch die DropDown-Liste in dem blau markierten Teil in der Abbildung 35
- Die Werte aller Checkboxes, gekennzeichnet durch die Checkboxes in dem blau markierten Teil in der Abbildung 35

Im dritten Teilschritt, der sich auf die Operation „Update project_config JSON file in „.tmp“ directory“ in der Abbildung 39 bezieht, aktualisiert das Programm die JSON-Datei des „.tmp“-Ordners, in der die Metadaten des Projektes gespeichert sind. Dazu liest das Programm die JSON-Datei ein und erstellt daraus ein Python Dictionary mit demselben Aufbau. Die alten Werte in dem Dictionary werden im Anschluss mit den neuen Werten, die im zweiten Teilschritt gespeichert wurden, ersetzt. Anschließend wird die alte JSON-Datei des „.tmp“-Ordners mit einer neuen JSON-Datei, die mithilfe des aktualisierten Dictionarys erstellt wurde, ersetzt.

Im vierten Teilschritt, der sich auf die Operation „Update Readme file in „.tmp“ directory“ in der Abbildung 39 bezieht, aktualisiert das Programm die „README.md“-Datei des „.tmp“-Ordners. Dazu öffnet das Programm diese Datei und ersetzt den Titel der Datei mit dem eingegeben langen Projektnamen, gekennzeichnet durch das „Projectname (long)“-Texteingabefeld des blau markierten Teils in der Abbildung 35.

Im fünften Teilschritt, der sich auf die Operation „Update Licenses in „.tmp“ directory“ in der Abbildung 39 bezieht, aktualisiert das Programm die Lizenz-Datei des „.tmp“-Ordners. Dazu löscht das Programm die aktuelle Lizenz-Datei des „.tmp“-Ordners und fügt diesem Ordner anschließend die neue Lizenz-Datei hinzu. Die neue Lizenz-Datei ist dabei die Lizenz, die in der entsprechenden DropDown-Liste, gekennzeichnet durch die „License“-DropDown-Liste des blau markierten Teils in der Abbildung 35, ausgewählt wurde. Wie in dem letzten Kapitel 6.2.1 erklärt, steht der ausgewählte Wert dieser DropDown-Liste für einen Unterordner des „license“-Ordners in der Projekt-Schablone. Dieser Unterordner enthält eine Textdatei. Diese Textdatei ist die neue Lizenz-Datei, die dem „.tmp“-Ordner hinzugefügt wird.

Im sechsten Teilschritt, der sich auf die Operation „Handle Checkboxes in „.tmp“ directory“ in der Abbildung 39 bezieht, überprüft das Programm den Wert der Checkboxes, gekennzeichnet durch die Checkboxes des blau markierten Teils in der Abbildung 35. Wenn dabei neue Checkboxes markiert oder Markierungen von Checkboxes entfernt wurden, werden die entsprechenden vordefinierten Konfigurationsdateien dem „.tmp“-Ordner entweder hinzugefügt oder aus dem „.tmp“-Ordner gelöscht.

Im siebten Teilschritt, der sich auf die Operation „Override all files in „.tmp“ directory with selected changes“ in der Abbildung 39 bezieht, aktualisiert das Programm alle Dateien, die als Listenelemente in der „Updatables Files“-Liste aufgelistet wurden. Dazu iteriert das Programm durch das Python Dictionary, indem die Änderungen der Dateien, durch den „Accept Changes“-Button, gespeichert wurden. Bei jeder Iteration öffnet das Programm die Datei, dessen Pfad als Schlüssel gespeichert ist, und überschreibt den Inhalt dieser Datei mit der Liste, die als Wert zu dem Schlüssel gespeichert ist. Wie zuvor erklärt enthält die Liste dabei den konfigurierten Inhalt des Listen-Widgets der „merged“-Instanz.

Im achten Teilschritt, der mit der Operation "Delete old project" beginnt und mit "Delete „.tmp“ directory" in der Abbildung 39 endet, aktualisiert das Programm das ausgewählte Projekt mithilfe des „.tmp“-Ordners. Dazu löscht das Programm zuerst das alte Projekt und erstellt in demselben Verzeichnis ein neues leeres Projekt bzw. einen neuen leeren Ordner mit dem eingegeben kurzen Projektnamen, gekennzeichnet als „Projectname (short)“-Texteingabefeld in dem blau markierten Teil in der Abbildung 35, als dessen Name. Anschließend kopiert das Programm den Inhalt

des „tmp“-Ordners in das neue leere Projekt. Als letztes löscht das Programm den „tmp“-Ordner.

Im letzten Teilschritt, der sich auf die Operation „Go back to main menu“ in der Abbildung 39 bezieht, wechselt die GUI das Fenster wieder zurück auf das Hauptmenü, wodurch der Prozess für die Projektaktualisierung beendet wird.

5.2.4 Fenster für die Erweiterung der Projektmodule

In diesem Kapitel wird erklärt wie das Fenster für die Erweiterung der Projektmodule, welches in dem Kapitel 5.2.4 erklärt wurde, in die Software implementiert wurde und wie Erweiterung der Projektmodule im Detail abläuft. Die Abbildung 23 des Kapitels 5.2.4 diente dabei als Vorlage für die Implementierung.

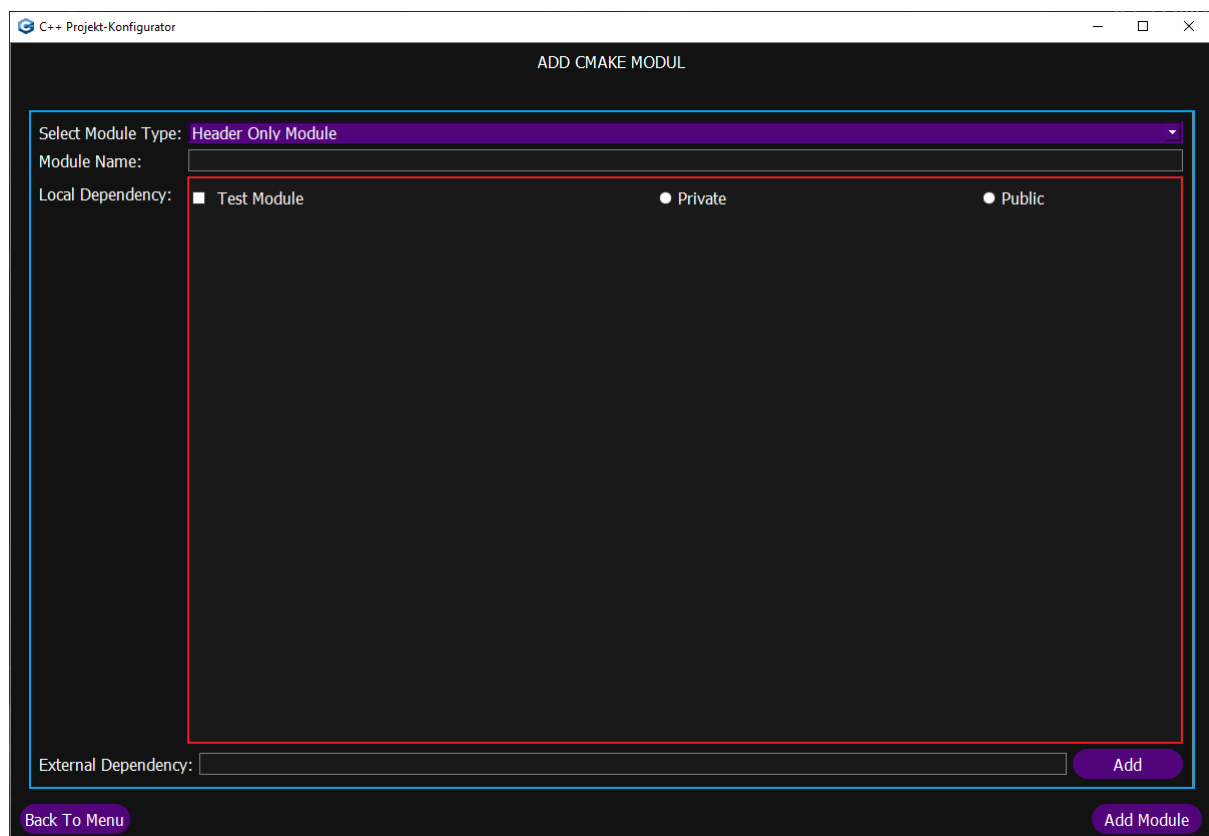


Abbildung 40: Implementierung des Fensters für die Projektmodulerweiterung

Die Abbildung 40 zeigt ein Screenshot des Fensters für die Erweiterung der Projektmodule. Dabei ist zusehen, dass dieses Fenster nach Vorlage der Abbildung 23 des Kapitels 5.2.4 implementiert wurde, da der Aufbau des Fensters und die Benennung der Widgets unverändert von der Abbildung 23 übernommen wurden. Im Folgenden wird mithilfe von verschiedenen Flussdiagrammen erklärt, wie ein

Projektmodul durch den Projekt-Konfigurator erstellt und dem ausgewählten Projekt hinzugefügt wird.

Um ein neues Projektmodul mit diesem Fenster erstellen zu können, fügt das Programm, nach dem Klick auf den „Add Cmake Modul“-Button im Hauptmenü, gekennzeichnet durch den „Add Cmake Modul“-Button in der Abbildung 20 des Kapitels 5.2.1, dem Listen-Widget, gekennzeichnet durch den rot markierten Teil der Abbildung 40, alle lokalen Abhängigkeiten, die für das neue Projektmodul genutzt werden können, als Listenelemente hinzu. Erst nach auflisten der Listenelemente wird das Fenster für die Erweiterung der Projektmodule in der GUI gezeigt.

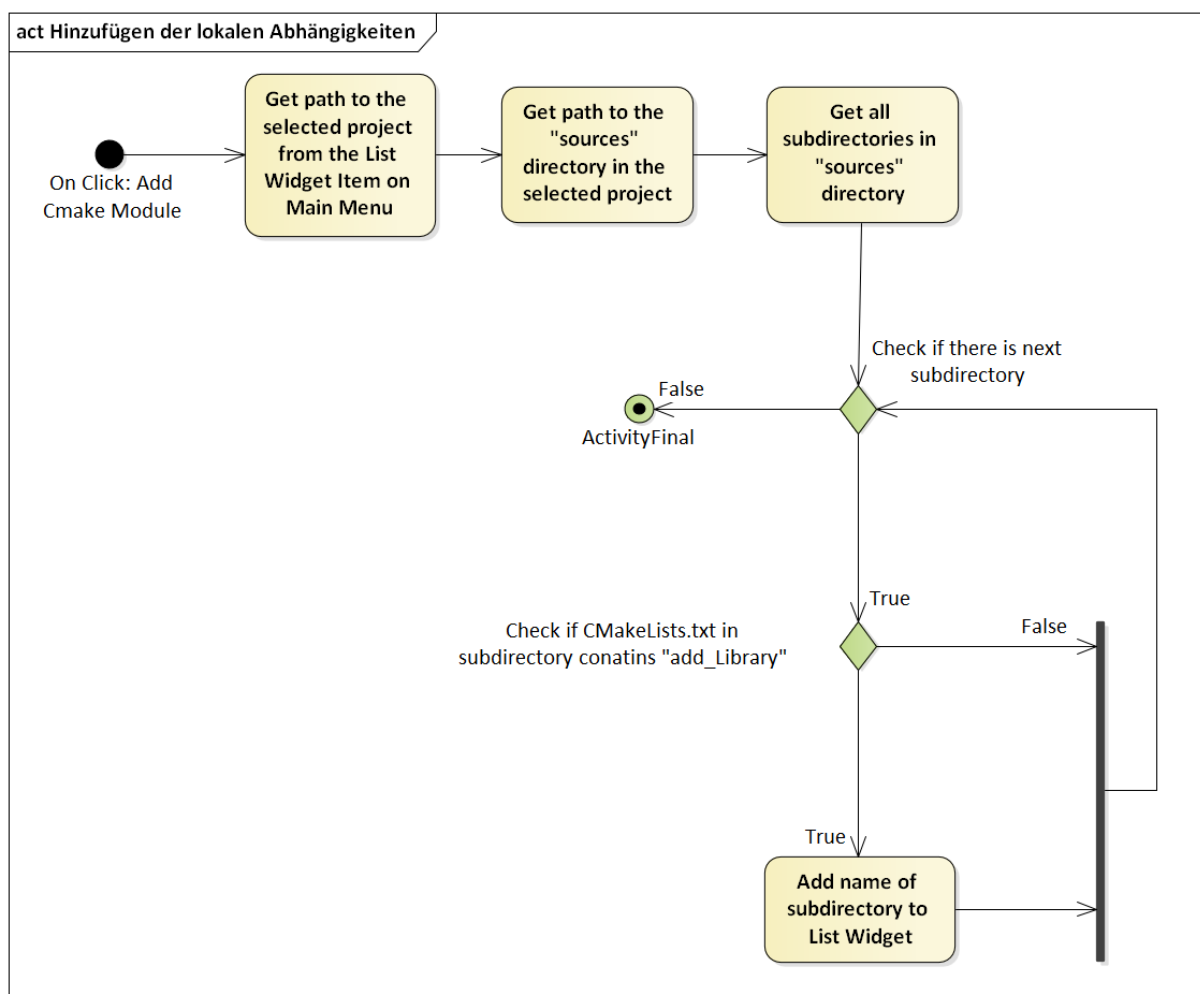


Abbildung 41: Flussdiagramm für das Hinzufügen der lokalen Abhängigkeiten

In der Abbildung 41 ist der Ablauf dargestellt, durch den die lokalen Abhängigkeiten dem Listen-Widget, gekennzeichnet durch den rot markierten Teil in der Abbildung 40, hinzugefügt werden. Dieser Prozess kann in drei Teilschritten erklärt werden:

Im ersten Teilschritt, der mit der Operation „Get path to the selected project from the List Widget Item on Main Menu“ beginnt und mit „Get path to the „sources“ directory in the selected project“ in der Abbildung 41 endet, generiert das Programm den Pfad zu dem Quellen-Unterordner, gekennzeichnet durch den „sources“-Ordner in der Abbildung 27 des Kapitels 6.1, des ausgewählten Projektes generiert.

Im zweiten Teilschritt, der sich auf die Operation „Get all subdirectories in „sources“ directory“ in der Abbildung 41 bezieht, speichert das Programm alle Namen der Ordner, die sich in dem Quellen-Unterordner befinden, in einer internen Liste. Die Namen werden dabei als Strings in der Liste gespeichert. Die Unterordner des Quellen-Ordners sind dabei alle internen Module des Projektes, die mit dem neu erstellten Projektmodul verknüpft werden können.

Im letzten Teilschritt, der mit der Operation "Check if there is next subdirectory" beginnt und mit "Add name of subdirectory to the List Widget" in der Abbildung 41 endet, fügt das Programm die lokalen Abhängigkeiten, die mit dem neuen Projektmodul verknüpft werden können, dem Listen-Widget des Fensters hinzu. Dazu iteriert das Programm durch die Liste mit den Namen der Ordner. Bei jeder Iteration wird die „CMakeLists.txt“-Datei des aktuellen Ordners geprüft. Befindet sich der Teilstring „add_Library“ in der „CMakeLists.txt“-Datei, wird der Name des aktuellen Ordners dem Listen-Widget, gekennzeichnet durch den rot markierten Teil in der Abbildung 40, hinzugefügt, um als lokale Abhängigkeit genutzt werden zu können. Dazu wird eine *QListWidgetItem*-Instanz mit dem Namen des Ordners erstellt und dem Listen-Widget hinzugefügt. Befindet sich dieser Teilstring nicht in der „CMakeLists.txt“-Datei, wird der Name des Ordners dem Listen-Widget nicht hinzugefügt und das Programm überprüft die nächste „CMakeLists.txt“-Datei des nächsten Ordners.

Nachdem alle Ordner in der intern gespeicherten Liste überprüft wurden, ist der Prozess, der in der Abbildung 41 dargestellt ist, beendet und das Fenster für die Erweiterung der Projektmodule wird in der GUI gezeigt. Dadurch kann der Benutzer nun das ausgewählte Projekt mit einem neuen Projektmodul erweitern. Der Prozess, der dafür intern abläuft, wird nachfolgend erklärt.

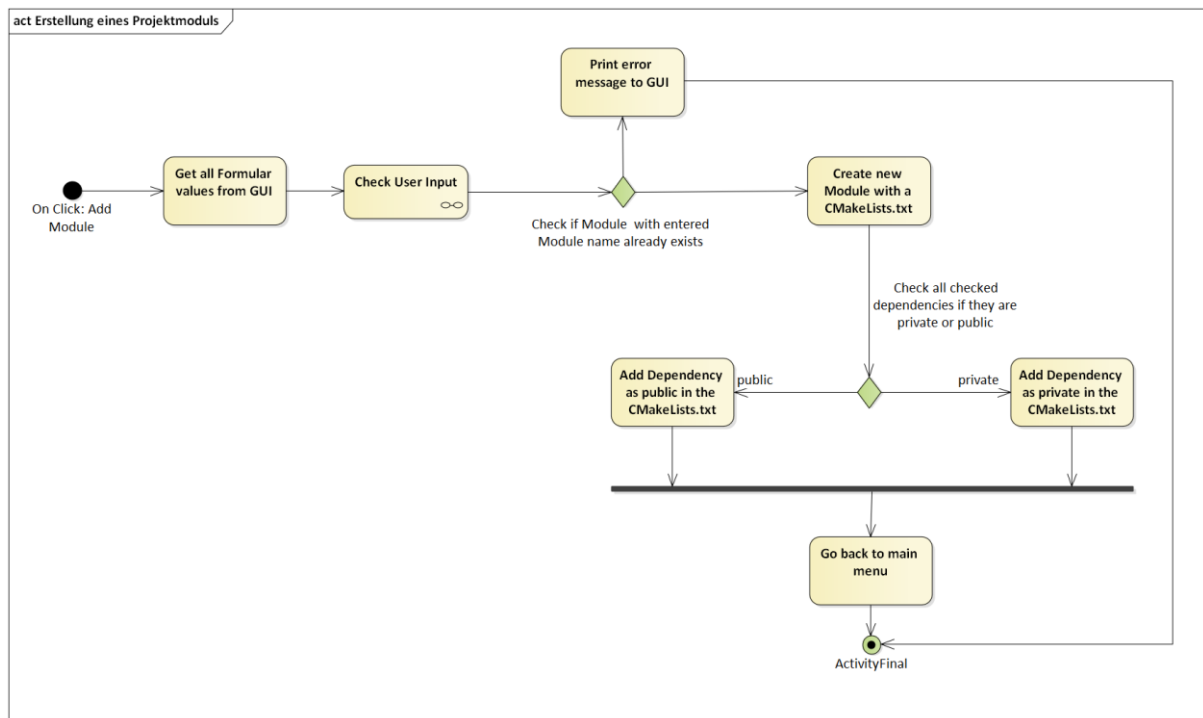


Abbildung 42: Flussdiagramm für die Erstellung eines Projektmoduls

In der Abbildung 42 ist der Ablauf dargestellt, durch den ein Projekt mit einem neu erstellten Projektmodul erweitert werden kann. Gestartet wird dieser Prozess mit dem Klick auf den „Add Module“-Button, gekennzeichnet durch den „Add Module“-Button in der Abbildung 40. Der Prozess kann ich fünf Teilschritten erklärt werden:

Im ersten Teilschritt, der sich auf die Operation „Get all Formular values from GUI“ in der Abbildung 42 bezieht, speichert das Programm folgende Werte, aus dem Formular für die Projekterstellung, zusehen in der Abbildung 40, als Strings:

- Der Modultyp – auszuwählen in der Modultyp-DropDown-Liste, gekennzeichnet als „Select Module Type“-DropDown-Liste in dem blau markierten Teil der Abbildung 40
- Der Modulname – einzugeben in dem „Module Name“-Texteingabefeld, gekennzeichnet durch das Texteingabefeld in dem blau markierten Teil in der Abbildung 40

Nachdem die Werte gespeichert wurden, startet der zweite Teilschritt, der sich auf die Operation „Check User Input“ in der Abbildung 42 bezieht, bei dem der Modulname auf verschiedene Kriterien überprüft wird. Dieser Prozess wird mithilfe eines Zusätzlichen Hilfsdiagramm erklärt:

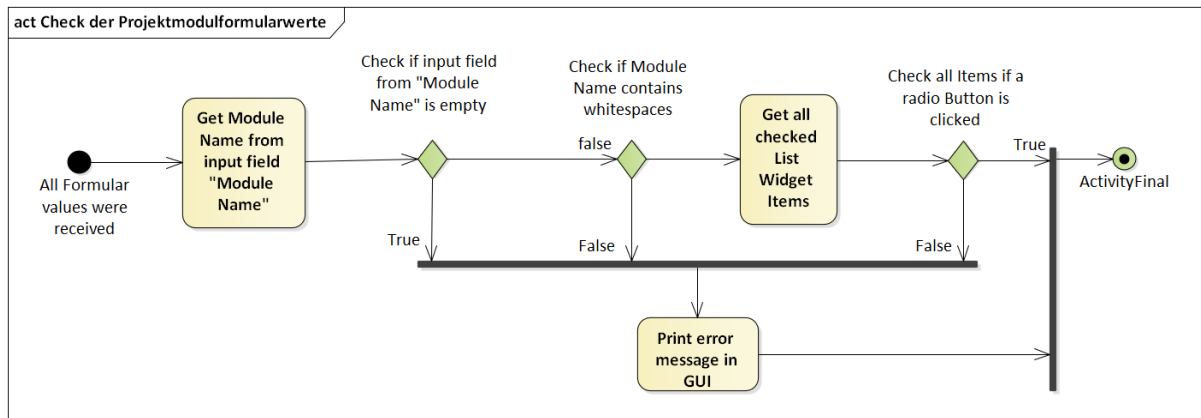


Abbildung 43: Flussdiagramm für den Check der Projektmodulformularwerte

In der Abbildung 43 ist der Ablauf dargestellt, durch den der eingegebene Modulname überprüft wird. Als erstes, zusehen als Operation „Check if input field from „Module Name“ is empty“ in der Abbildung 43, kontrolliert das Programm, ob der gespeicherte String des Modulnamens leer ist, also ob ein Wert in das entsprechende Texteingabefeld eingegeben wurde.

Anschließend, zusehen als Operation „Check if Module Name contains whitespaces“ in der Abbildung 43, wird überprüft, ob der String des Modulnamens Leerzeichen enthält. Wenn eines dieser Überprüfungen fehlschlägt, also der String entweder leer ist oder Leerzeichen enthält, wird eine entsprechende Error-Nachricht in der GUI ausgegeben und der gesamte Prozess der Projektmodulerweiterung gestoppt.

Andernfalls wird der Prozess fortgeführt, indem das Programm als nächstes, zusehen als Operation „Get all checked List Widget Items“ in der Abbildung 43, alle Namen der markierten Listenelemente, also die Namen der lokalen Abhängigkeiten, des Listen-Widgets, gekennzeichnet durch den rot markierten Teil in der Abbildung 40, intern in eine Liste speichert. Die Listenelemente zählen als markiert, wenn die Checkbox der Listenelemente markiert wurde.

Anschließend, zusehen als Operation „Check all items if a radio Button is clicked“ in der Abbildung 43, iteriert das Programm durch die Liste und prüft dabei für jedes Listenelement, also für jede lokale Abhängigkeit, ob ein Radio-Button des Listenelements ausgewählt wurde. Falls das nicht der Fall ist, wird ebenfalls eine entsprechende Error-Nachricht in der GUI ausgegeben und der gesamte Prozess der Projektmodulerweiterung gestoppt.

Wenn jedoch für jedes Listenelement ein radio-button ausgewählt wurde, startet der dritte Teilschritt des gesamten Projektmodulerweiterungsprozesses. In dem dritten Teilschritt, der sich auf die Operation „Check if Module with entered Module name already exists“ in der Abbildung 43 bezieht, überprüft das Programm, ob für das ausgewählte Projekt schon ein Projektmodul mit dem gleichen eingegebenen Modulnamen existiert. Wenn das der Fall ist, wird eine entsprechende Error-Nachricht in der GUI ausgegeben und der gesamte Prozess der Projektmodulerweiterung gestoppt.

Falls noch kein gleichnamiges Projektmodul existiert, startet der vierte Teilschritt. In dem vierten Teilschritt, der sich auf die Operation „Create new Module with CMakeLists.txt“ in der Abbildung 42 bezieht, erstellt das Programm das neue Projektmodul bzw. einen neuen Ordner, in dem Quellen-Unterordner, gekennzeichnet durch den „sources“-Ordner in der Abbildung 27 des Kapitels 6.1, des ausgewählten Projektes. Das Projektmodul wird dabei nach dem eingegeben Modulnamen benannt. In das neue Projektmodul wird anschließend eine Kopie der vordefinierten „CMakeLists.txt“-Datei aus dem ausgewählten Modultyp-Ordner der Projekt-Schablone hinzugefügt. Der Modultyp-Order wird dabei aus der „Select Module Type“-DropDown-Liste ausgewählt. Den Wert, der aus dieser DropDown-Liste ausgewählt wird, bezieht sich auf den gleichnamigen Unterordner des Quellen-ordners, gekennzeichnet durch die Unterordner des „sources“-Ordners in der Abbildung 27 des Kapitels 6.1, der Projekt-Schablone.

In dem letzten Teilschritt, der sich auf die Operation „Check all checked dependencies if they are private or public“ in der Abbildung 42 bezieht, wird die intern gespeicherte Liste mit den Namen der markierten Listenelemente bzw. den Namen der lokalen Abhängigkeiten nochmals durchiteriert. Dabei wird für jedes Listenelement überprüft, ob der „Private“-Radiobutton, gekennzeichnet durch den „Private“-Radiobutton in dem rot markierten Teil in der Abbildung 40, oder der „Public“-Radiobutton, gekennzeichnet durch den „Public“-Radiobutton in dem rot markierten Teil in der Abbildung 40, ausgewählt wurde. Die Überprüfung der Radiobuttons ist wichtig, da Cmake wissen muss, ob das die lokale Abhängigkeit offengelegt werden muss oder nicht. Wenn der „Private“-Radiobutton ausgewählt wurde, wird dieses Listenelement bzw. diese lokale Abhängigkeit dem Projektmodul als private lokale Abhängigkeit hinzugefügt. Dazu wird der Name der lokalen Abhängigkeit der „CMakeLists.txt“-Datei als private lokale

Abhängigkeit hinzugefügt. Andernfalls wird das Listenelement bzw. die lokale Abhängigkeit dem Projektmodul als öffentliche lokale Abhängigkeit hinzugefügt. Dazu wird der Name der lokalen Abhängigkeit der „CMakeLists.txt“-Datei als öffentliche lokale Abhängigkeit hinzugefügt.

Nachdem dem Projektmodul alle lokalen Abhängigkeiten hinzugefügt wurden, wechselt die GUI das Fenster wieder zurück auf das Hauptmenü, wodurch der Prozess für die Erweiterung der Projektmodule beendet wird.

5.2.5 Fenster für die Ausführung der Dienstprogramme

In diesem Kapitel wird erklärt, wie das Fenster für die Ausführung der Dienstprogramme, welches in dem Kapitel 5.2.5 erklärt wurde, in die Software implementiert wurde und wie die Ausführung der Dienstprogramme im Detail abläuft. Die Abbildung 24 des Kapitels 5.2.5 diente dabei als Vorlage für die Implementierung.



Abbildung 44: Implementierung des Fensters für die Dienstprogrammausführung

Die Abbildung 44 zeigt ein Screenshot des Fensters für die Ausführung der Dienstprogramme. Dabei ist zusehen, dass dieses Fenster nach Vorlage der Abbildung 24 des Kapitels 5.2.5 implementiert wurde, da der Aufbau des Fensters und die Benennung der Widgets unverändert von der Abbildung 24 übernommen wurden.

Im Folgenden wird mithilfe von verschiedenen Flussdiagrammen erklärt, wie Dienstprogramme durch den Projekt-Konfigurator ausgeführt werden.

Um die Dienstprogramme eines Projekts mit diesem Fenster ausführen zu können, fügt das Programm, nach dem Klick auf den „Run Utils“-Button, gekennzeichnet durch den „Run Utils“-Button in der Abbildung 20 des Kapitels 6.2.1, im Hauptmenü, dem Listen-Widget, gekennzeichnet durch die linke Hälfte des blau markierten Teils der Abbildung 44, alle Dienstprogramme als Listenelemente hinzu. Erst nach auflisten der Listenelemente wird das Fenster für die Ausführung der Dienstprogramme in der GUI gezeigt.

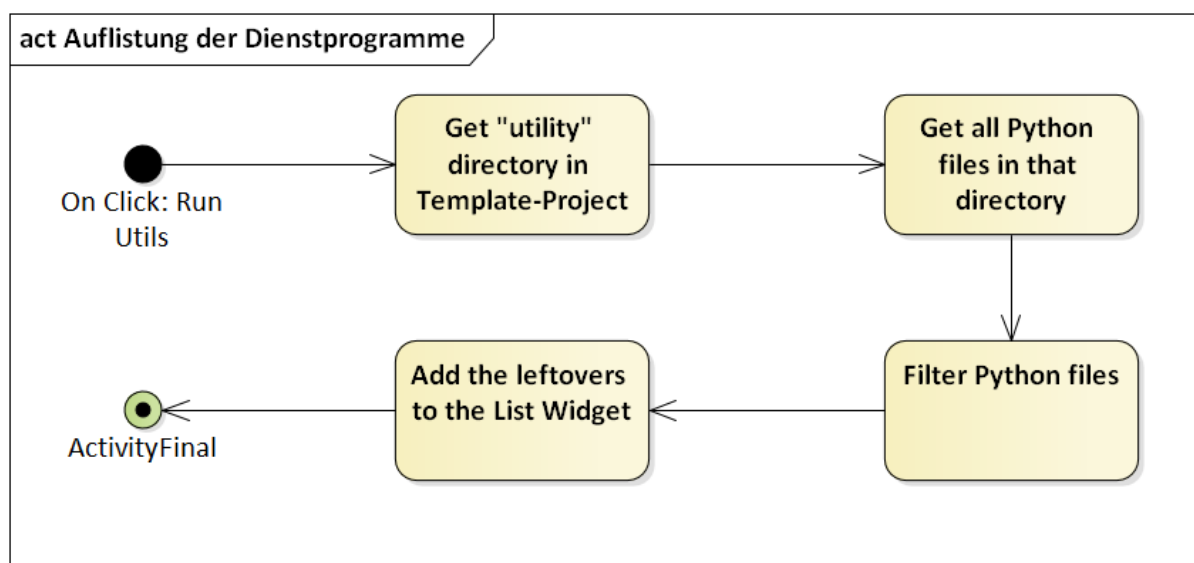


Abbildung 45: Flussdiagramm für die Auflistung der Dienstprogramme

In der Abbildung 45 ist der Ablauf dargestellt, durch den die Dienstprogramme eines Projektes in der Liste, gekennzeichnet durch die linke Hälfte des blau markierten Teils der Abbildung 44, des Fensters aufgelistet werden. Dieser Prozess kann in vier Teilschritten erklärt werden:

In dem ersten Teilschritt, gekennzeichnet durch die Operation „Get „utility“ directory in Template-Project“ in der Abbildung 45, erstellt das Programm den Pfad zu dem Dienstprogramm-Unterordner, gekennzeichnet durch den „utility“-Ordner in der Abbildung 27 des Kapitels 5.1, des ausgewählten Projektes.

Im zweiten Teilschritt, gekennzeichnet durch die Operation „Get all Python files in that directory“ in der Abbildung 45, speichert das Programm intern eine Liste, mit den

Namen der Python-Dateien des Dienstprogramm-Unterordners. Die Namen werden dabei als Strings in der Liste gespeichert.

In dem dritten Teilschritt, gekennzeichnet durch die Operation „Filter Python files“ in der Abbildung 45, wird diese Liste gefiltert, wodurch entsprechende Listenelemente bzw. Python-Datei-Namen aus der Liste entfernt werden. Dazu liest das Programm eine bestimmte JSON-Datei ein, in der die Python-Datei-Namen aufgelistet sind, die aus der Liste entfernt werden sollen. Die Namen aus der JSON-Datei werden nach dem Einlesen in einer zweiten Liste gespeichert. Anschließend werden alle Python-Datei-Namen, die sich in der zweiten Liste befinden aus der ersten Liste entfernt.

Als letzter Teilschritt, gekennzeichnet durch die Operation „Add the leftovers to the List Widgets“ in der Abbildung 45, wird jeder Python-Datei-Name, der nicht aus der ersten Liste gefiltert wurde, dem Listen-Widget, gekennzeichnet durch die linke Hälfte des blau markierten Teils der Abbildung 44, als Listenelement hinzugefügt. Dazu wird für jedes Listenelement eine *QListWidgetItem*-Instanz, mit dem entsprechenden Python-Datei-Namen, erstellt und dem Listen-Widget hinzugefügt.

Nachdem die Dienstprogramme dem Listen-Widget hinzugefügt wurden, zeigt die GUI das Fenster mit den aufgelisteten Dienstprogrammen an, wodurch die Benutzer diese auswählen und ausführen können. Dies kann auf zwei Arten passieren, die nachfolgend erklärt werden.

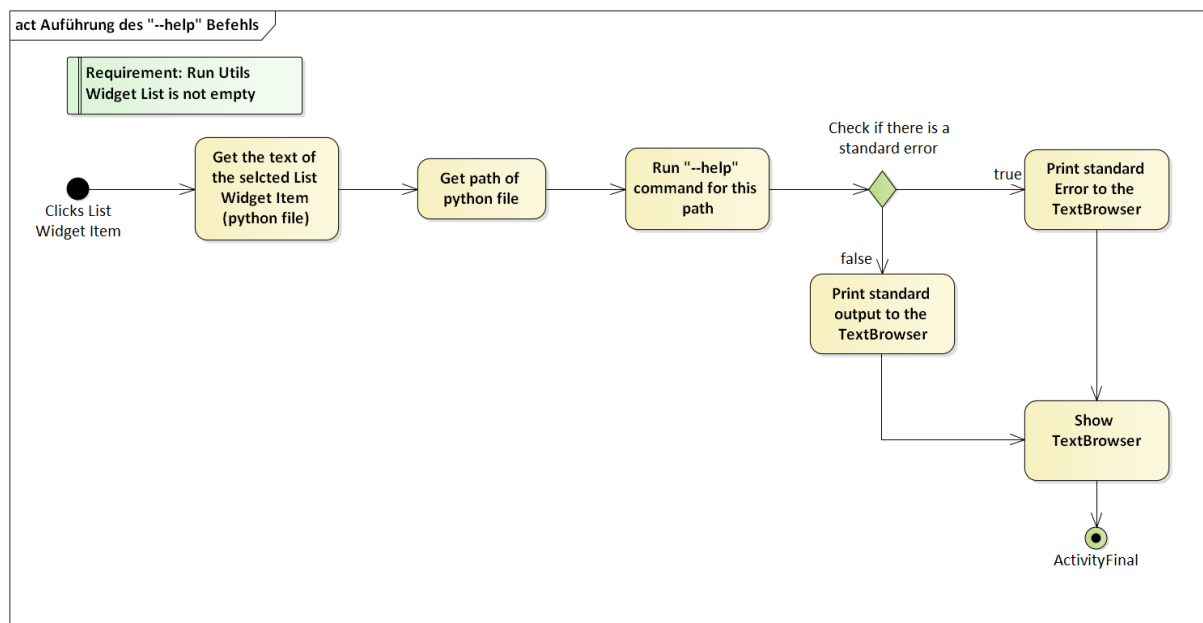


Abbildung 46: Flussdiagramm für die Ausführung des "--help"-Befehls

In der Abbildung 46 ist die erste Art und somit der Ablauf dargestellt, durch den der „--help“-Befehl für ein ausgewähltes Dienstprogramm ausgeführt werden kann. Dabei startet der Prozess, nachdem ein Listenelement aus dem Listen-Widget ausgewählt wird. Dieser Prozess kann in drei Teilschritten erklärt werden:

In dem ersten Teilschritt, der mit der Operation "Get the text of the selected List Widget Item (python file)" beginnt und mit "Get path of python file" in der Abbildung 46 endet, speichert das Programm den Namen des ausgewählten Listenelements und generiert anschließend den Pfad zu dem entsprechenden Dienstprogramm in dem ausgewählten Projekt.

Im zweiten Teilschritt, gekennzeichnet durch die Operation „Run „--help“ command for this path“ in der Abbildung 46, führt das Programm das ausgewählte Dienstprogramm mit dem „--help“-Befehl aus und speichert dabei den Standardausgang, sowie -error der Ausführung als Strings.

Im letzten Teilschritt, der mit der Operation "Check if there is a standard error" beginnt und mit "Show TextBrowser" in der Abbildung 46 endet, prüft das Programm, ob der String des Standarderrors leer ist. Ist das der Fall, wird der Standardausgang-String in dem Textbrowser, gekennzeichnet durch die rechte Hälfte in der Abbildung 44, angezeigt. Dazu wird der Text des Textbrowsers auf den Text des Standardausgang-Strings gesetzt. Falls der String des Standarderrors nicht leer ist, wird stattdessen der Standarderror-String in dem Textbrowser angezeigt.

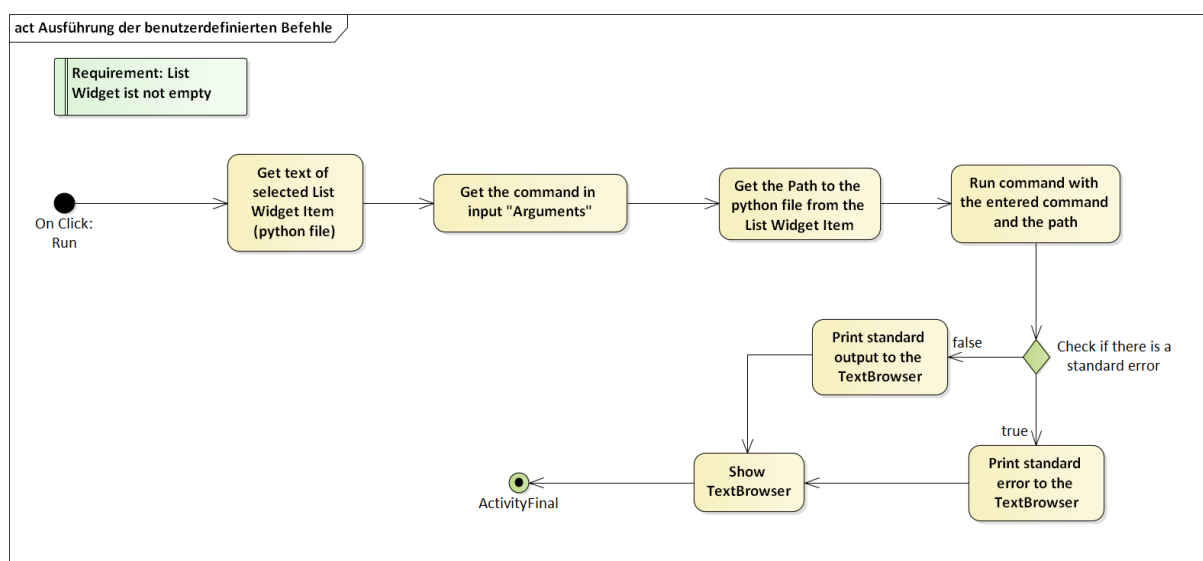


Abbildung 47: Flussdiagramm für die Ausführung der benutzerdefinierten Befehle

In der Abbildung 47 ist die zweite Art und somit der Ablauf dargestellt, durch den ein eingegebener Befehl für ein ausgewähltes Dienstprogramm ausgeführt werden kann. Dabei startet der Prozess, nachdem der „Run“-Button, gekennzeichnet durch den „Run“-Button des rot markierten Teils in der Abbildung 44, geklickt wird. Dieser Prozess kann in drei Teilschritten erklärt werden:

In dem ersten Teilschritt, der mit der Operation "Get text of selected List Widget Item (python file)" beginnt und mit "Get the Path to the python file from the List Widget Item" in der Abbildung 47 endet, speichert das Programm den Namen des ausgewählten Listenelements und generiert anschließend den Pfad zu dem entsprechenden Dienstprogramm in dem ausgewählten Projekt.

In dem zweiten Teilschritt, der mit der Operation "Get the command in input „Arguments“" beginnt und mit "Run command with the entered command and the path" in der Abbildung 47 endet, speichert das Programm den eingegebenen Befehl, der in das „Arguments“-Texteingabefeld, gekennzeichnet durch das Texteingabefeld des rot markierten Teils in der Abbildung 44, eingegeben wurde als String. Anschließend führt das Programm das ausgewählte Dienstprogramm mit dem gespeicherten Befehl aus und speichert dabei den Standardausgang, sowie -error der Ausführung als Strings.

In dem dritten Teilschritt, der mit der Operation "Check if there is a standard error" beginnt und mit "Show TextBrowser" in der Abbildung 47 endet, prüft das Programm, ob der String des Standarderrors leer ist. Ist das der Fall, wird der Standardausgang-String in dem Textbrowser, gekennzeichnet durch die rechte Hälfte in der Abbildung 44, angezeigt. Dazu wird der Text des Textbrowsers auf den Text des Standardausgang-Strings gesetzt. Falls der String des Standarderrors nicht leer ist, wird stattdessen der Standarderror-String in dem Textbrowser angezeigt.

5.2.6 Zusammenfassung

In dem Kapitel 6.2 wurde die Implementierung jedes GUI-Fensters gezeigt und dabei erklärt wie die Funktionen jedes Fensters softwareseitig ablaufen.

Das Hauptmenü ist für den Suchungsmechanismus für bestehende C++-Projekte zuständig. Dazu muss ein Pfad in einem Texteingabefeld eingegeben werden, in dem das Programm nach existierenden Projekten sucht. Wenn das Programm welche findet, werden diese in einer Liste auf dem Hauptmenü ausgegeben.

Das Fenster für die Projekterstellung ist für die Erstellung eines neuen Projektes zuständig. Dazu muss ein Formular in dem Fenster ausgefüllt werden, in dem alle notwendigen Information für die Erstellung des Projektes einzugeben sind. Anschließend erstellt das Programm eine Kopie der Projekt-Schablone in dem eingegeben Speicherort und führt anschließend, mithilfe der eingegebene Informationen, einige Konfigurationen an der Kopie aus, sodass am Ende des Prozesses ein neues Projekt entsteht.

Das Fenster für die Projektaktualisierung ist für die Aktualisierung eines Projektes zuständig. Dazu vergleicht das Programm ein ausgewähltes Projekt mit der intern gespeicherten Projekt-Schablone und fügt dabei den Namen jeder aktualisierbaren Datei einer Liste des Fensters hinzu. Für jede Datei aus der Liste kann anschließend entschieden werden, ob die Änderungen von der gleichnamigen Datei der Projekt-Schablone für die reale Datei übernommen werden sollen oder nicht. Dabei kann auch entschieden werden, dass nur einzelne Teile und nicht die gesamten Änderungen übernommen werden. Diese Konfigurationen können per Knopfdruck gespeichert werden. Wenn für jede Datei eine Konfiguration gespeichert wurde, kann das Projekt durch den „Update Project“-Button aktualisiert werden. Dadurch überschreibt das Programm den Inhalt der Dateien, mit den gespeicherten Konfigurationen für die Dateien.

Das Fenster für die Erweiterung der Projektmodule ist für die Erweiterung eines Projektes durch ein neues Projektmodul zuständig. Dazu muss ein, ähnlich wie bei der Projekterstellung, ein Formular in dem Fenster ausgefüllt werden, in dem alle notwendigen Informationen für das neue Projektmodul eingegeben werden müssen. Das Programm erstellt daraufhin ein neues Projektmodul mit der entsprechenden CMakeLists.txt-Datei. In die CMakeLists.txt-Datei werden dabei alle lokalen Abhängigkeiten des Moduls, die aus einer Liste in dem Formular ausgewählt wurden, eingetragen.

Das Fenster für die Ausführung der Dienstprogramme ist für die Ausführung der Dienstprogramme eines ausgewählten Projektes zuständig. Dazu werden als erstes alle ausführbaren Dienstprogramme des Projektes in einer Liste des Fensters aufgelistet. Dadurch können die Benutzer eines der Dienstprogramme auswählen und in einem Texteingabefeld die Befehle eingeben, mit denen das Dienstprogramm ausgeführt werden soll. Durch Drücken des „Run“-Buttons führt das Programm das

ausgewählte Dienstprogramm mit den eingegebenen Befehlen aus und gibt danach die Ausgabe der Ausführung in dem Textbrowser des Fensters aus.

5.3 Mögliche Erweiterungspunkte

In diesem Kapitel werden kurz einige mögliche Erweiterungspunkte des Projekt-Konfigurators erklärt.

Die erste mögliche Erweiterung bezieht sich auf die Erweiterung der Modultypen in dem Quellen-Ordner, gekennzeichnet durch die Unterordner des „sources“-Ordners in der Abbildung 27 des Kapitels 6.1, und der Lizenztypen in dem Lizenzen-Ordner, gekennzeichnet durch die Unterordner des „licenses“-Ordners in der Abbildung 27 des Kapitels 6.1, der Projekt-Schablone. Diese Erweiterung ist die einfachste, da dafür der Quellcode des Projekt-Konfigurators nicht verändert bzw. erweitert werden muss. Die Modultypen und Lizenztypen werden nämlich in den entsprechenden DropDown-Listen, gekennzeichnet durch die „Module Type“-DropDown-Liste und die „License Type“-DropDown-Liste in dem grün markierten Teil in der Abbildung 30 des Kapitels 6.2.2, angezeigt. Dabei ist egal wie viele Modul- bzw. Lizenztypen in der Projekt-Schablone enthalten sind.

Die zweite mögliche Erweiterung bezieht sich auf die Erweiterung der GUI-Fenster. Durch die Architektur der Software ist es entsprechend einfach neue Fenster für die GUI zu erstellen. Dafür muss lediglich ein neues Kindelement der Klasse *SubScreen* / *Screen* erstellt werden und dem *QStackedWidget* hinzugefügt werden.

Die dritte mögliche Erweiterung ist das Hinzufügen einer Möglichkeit, mit der die Benutzer eigene Projekt-Schablonen in den Projekt-Konfigurator integrieren können. Durch die Integration einer weiteren Schablone, können die Benutzer in dem Fenster für die Projekterstellung auswählen welche Projekt-Schablone für die Erstellung verwendet werden soll. Die gleiche Auswahl haben die Benutzer dementsprechend auch bei dem Fenster für die Projektaktualisierung, bei der ausgewählt werden kann welche Projekt-Schablone für die Aktualisierung verwendet werden soll.

Die vierte mögliche Erweiterung ist, dass die Projekte nicht mehr lokal in dem Dateisystem erstellt werden, sondern als online Repository auf einer dementsprechenden Webseite.

5.4 Zusammenfassung

Zusammengefasst wurde in diesem Kapitel 6 die Implementierung der Software erklärt. Dazu wurde:

Als erstes die intern gespeicherte Projekt-Schablone gezeigt und dabei erklärt welche Dateien der Schablone für die Projekterstellung konfiguriert werden müssen.

Als zweites wurden die implementierten Fenster der GUI gezeigt und dabei der interne Ablauf jeder Funktion der einzelnen Fenster erklärt.

Als drittes wurde ein Ausblick in die Zukunft gegeben, bei dem einige mögliche Erweiterungspunkte für die GUI genannt wurden.

6. Zusammenfassung

In diesem Kapitel wird das Problem und die daraus resultierende Lösung zusammengefasst.

Große Softwareprojekte werden während der Entwicklung meist aus modularen Teilen aufgebaut. Diese werden dabei selbst als eigene Softwareprojekte mit speziellen Tools und einer eigenen Struktur entwickelt. Die Struktur der Softwareprojekte wird in der Regel durch eine vorgegebene Schablone bestimmt. Die Schablone ist dabei ein Ordner mit vordefinierten Unterordnern und Dateien, die für jedes Softwareprojekt notwendig sind. Einige Dateien der Schablone enthalten allerdings Platzhalter, die für jedes Projekt individuell ersetzt werden müssen. Diese Ausgangssituation sorgt für zwei Probleme:

Das erste Problem bezieht sich auf die Erstellung eines neuen Projektes. Dafür gibt es bisher zwei Möglichkeiten. Die erste Möglichkeit ist die Verwendung einer Versionskontrolle wie bspw. Git. Dabei muss die Schablone aus einem online Repository heruntergeladen werden und anschließend lokal auf das neue Projekt angepasst werden. Das bedeutet die Schablone muss mit dieser Möglichkeit manuell verändert werden, was zeitaufwändig und fehleranfällig ist. Aus diesem Grund ist die erste Möglichkeit keine gute Lösung.

Die zweite Möglichkeit ist die Erstellung eines neuen Projektes mithilfe einer integrierten Entwicklungsumgebung. Bei dieser Möglichkeit wird die Schablone in die IDE integriert, wodurch anschließend neue Projekte auf Basis dieser Schablone erstellt werden können. Bei der Erstellung fordert die IDE dazu auf einen Wert, für jeden Platzhalter in der Schablone, einzugeben, um die Platzhalter anschließend automatisch ersetzen zu können. Diese Möglichkeit kann allerdings nur mit wenigen IDEs umgesetzt werden. Außerdem können bei den IDEs nur Zeichenketten für die Platzhalter übergeben werden. Für einige Platzhalter ist es jedoch erforderlich einen Wert aus einer Liste auswählen zu können. Aus diesen Gründen ist auch die zweite Möglichkeit keine gute Lösung.

Das zweite Problem bezieht sich auf die Aktualisierung eines neuen Projektes. Es ist nämlich kein unwahrscheinliches Szenario, dass sich die Projekt-Schablone im Laufe des Entwicklungsprozesses verändert. Durch die Veränderung der Schablone, müssen nachfolgend alle Projekte aktualisiert werden, die auf der Basis dieser

Schablone erstellt wurden. Für dieses Problem gibt es bisher nur eine Möglichkeit, und zwar dies manuell zu tun. Die manuelle Aktualisierung ist allerdings keine gute Lösung, da die Durchführung dieser Aufgabe mühsam, zeitaufwändig und fehleranfällig ist.

Die Lösung für die zwei Probleme ist der Projekt-Konfigurator, der im Rahmen dieser Bachelorarbeit entwickelt wurde. Der Projekt-Konfigurator ist eine Software mit einer grafischen Benutzeroberfläche. Die Benutzeroberfläche besteht dabei aus sechs Fenstern:

- Dem Hauptmenü
- Dem Fenster für die Projekterstellung
- Dem Fenster für die Projektaktualisierung
- Dem Fenster für die Erweiterung der Projektmodule (Zusatzfunktion)
- Dem Fenster für die Ausführung der Dienstprogramme (Zusatzfunktion)
- Dem Ladebildschirm

Das Hauptmenü ist das erste Fenster, dass nach öffnen des Projekt-Konfigurators gezeigt wird. Das Hauptmenü ist für zwei Funktionalitäten zuständig. Die erste Funktionalität ist der Suchungsmechanismus von existierenden C++-Projekten. Dazu muss in dem Hauptmenü ein Pfad von dem Benutzer eingegeben werden, in dem C++-Projekte liegen. In diesem Pfad sucht das Programm intern nach bestehenden Projekten und gibt diese anschließend in einer Liste auf dem Hauptmenü aus.

Die zweite Funktionalität bezieht sich auf die Weiterleitung auf die anderen vier Fenster (der Ladebildschirm hierbei ausgeschlossen) der GUI. Dazu beinhaltet das Hauptmenü vier Buttons. Nach dem Klick auf eines der Buttons wechselt die GUI das Fenster auf eines der vier weiteren Fenster.

Das Fenster für die Projekterstellung ist für die Erstellung eines neuen Projektes zuständig. Dazu muss in dem Fenster ein Formular ausgefüllt werden, in das alle notwendigen Informationen des Projektes eingetragen werden müssen. Mithilfe dieser Informationen erstellt das Programm automatisch ein neues Projekt auf Basis einer intern gespeicherten Projekt-Schablone. Durch dieses Fenster wird das erste Problem dieser Bachelorarbeit gelöst.

Das Fenster für die Projektaktualisierung ist für die Aktualisierung eines Projektes zuständig. Dazu vergleicht das Programm intern ein ausgewähltes Projekt mit der

Projekt-Schablone und prüft dabei welche Dateien in dem Projekt aktualisiert werden können. Jede aktualisierbare Datei wird nach dem Vergleich zu einer Liste des Fensters hinzugefügt. Dadurch können die Benutzer jede Datei aus der Liste auswählen und entscheiden, ob die Änderungen der gleichnamigen Datei in der Projekt-Schablone für die Datei des Projektes angewendet werden soll oder nicht. Nachdem für jede Datei eine Entscheidung getroffen wurde, kann das ausgewählte Projekt aktualisiert werden. Durch dieses Fenster wird das zweite Problem dieser Bachelorarbeit gelöst.

Das Fenster für die Erweiterung der Projektmodule ist für die Erweiterung eines Projektes durch ein Projektmodul zuständig. Dazu muss, ähnlich wie bei der Projekterstellung, ein Formular in diesem Fenster ausgefüllt werden, in das alle notwendigen Informationen für das neue Projektmodul eingegeben werden müssen. Mithilfe dieser Informationen erstellt das Programm automatisch ein neues Projektmodul für ein ausgewähltes Projekt.

Das Fenster für die Ausführung der Dienstprogramme ist für die Ausführung der Dienstprogramme eines ausgewählten Projektes zuständig. Alle ausführbaren Dienstprogramme eines Projektes werden dazu in einer Liste des Fensters angezeigt. Dadurch können die Benutzer ein Dienstprogramm aus der Liste auswählen und in einem Texteingabefeld die Befehle eingeben, mit denen das Dienstprogramm ausgeführt werden soll. Die Ausgabe der Ausführung wird anschließend in dem Textbrowser des Fensters angezeigt.

Vorraussetzung für das Fenster für die Projektaktualisierung, das Fenster für die Erweiterung der Projektmodule und das Fenster für die Ausführung der Dienstprogramme ist, dass der Suchungsmechanismus des Hauptmenüs durchgeführt und ein Projekt aus der resultierenden Liste ausgewählt wurde.

Der Ladebildschirm wird angezeigt, wenn ein Projekt mit der Projekt-Schablone verglichen wird. Nachdem der Vergleich beendet ist, wird der Ladebildschirm geschlossen und das Fenster für die Projektaktualisierung, mit einer Auflistung aller aktualisierbaren Dateien, in der GUI angezeigt.

Literatur

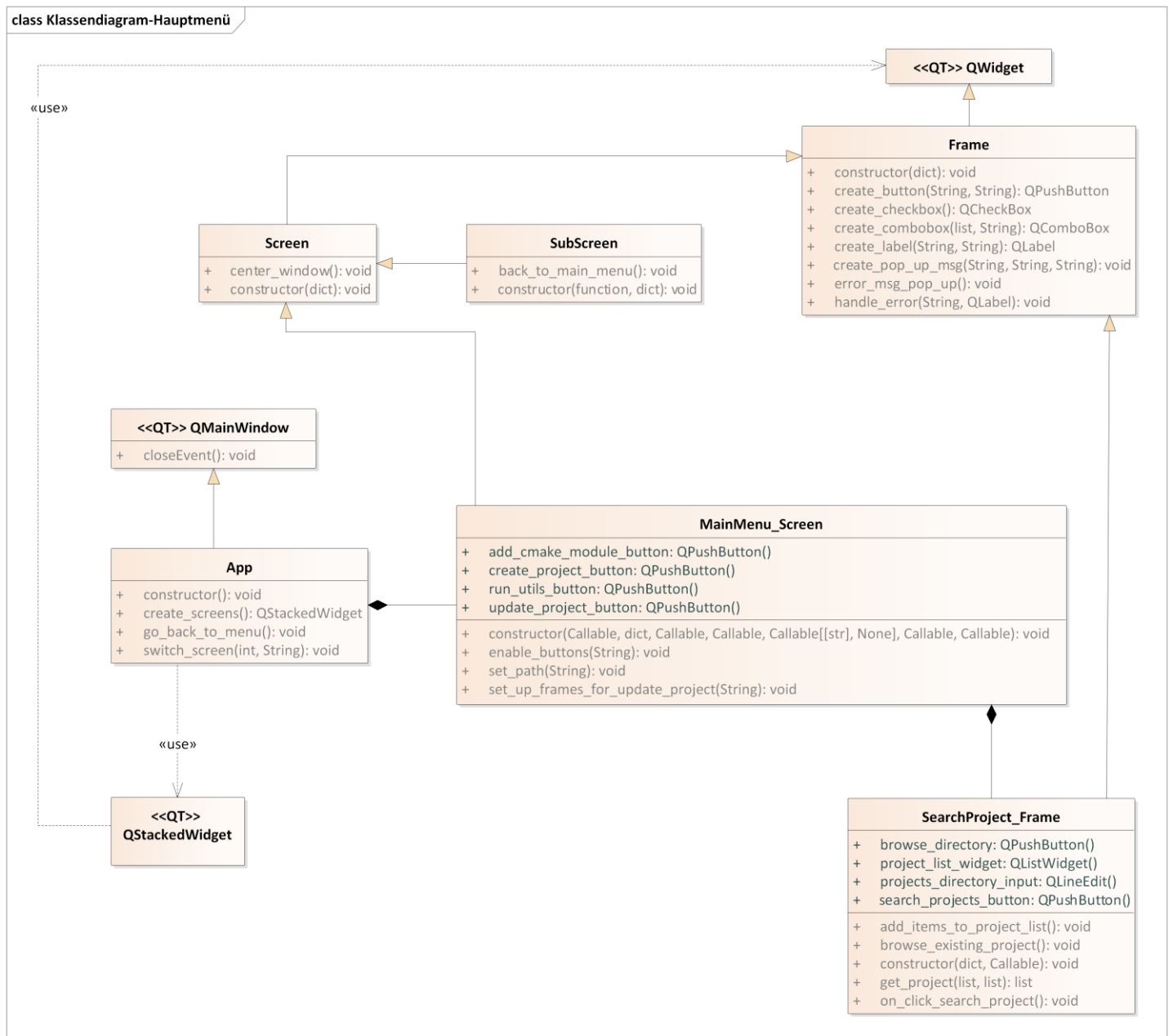
- [1] E. Wolff, *Continuous delivery: Der pragmatische Einstieg*, 2. Aufl. Heidelberg, Germany: dpunkt.verlag, 2016. [Online]. Verfügbar unter: <https://ebookcentral.proquest.com/lib/gbv/detail.action?docID=4471176>
- [2] M. Hüttermann, *DevOps for developers*. New York, NY: Apress, 2012. [Online]. Verfügbar unter: <https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=1156065>
- [3] R. Alt, *Innovationsorientiertes IT-Management mit DevOps: IT im Zeitalter von Digitalisierung und Software-defined Business*. Wiesbaden: Springer Gabler, 2017.
- [4] Splunk, *Was ist eine CI/CD-Pipeline?* [Online]. Verfügbar unter: https://www.splunk.com/de_de/data-insider/what-is-ci-cd-pipeline.html (Zugriff am: 5. Januar 2022.576Z).
- [5] J. Rossberg, *Agile Project Management with Azure DevOps: Concepts, Templates, and Metrics*. Berkeley, CA: Apress L. P, 2019. [Online]. Verfügbar unter: <https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=5771167>
- [6] S. Augsten, „Was ist ein Build?“, *Dev-Insider*, 27. Apr. 2018, 2018. [Online]. Verfügbar unter: <https://www.dev-insider.de/was-ist-ein-build-a-702737/>. Zugriff am: 24. November 2021.114Z.
- [7] Atlassian, *Was ist Versionskontrolle? | Atlassian Git Tutorial*. [Online]. Verfügbar unter: <https://www.atlassian.com/de/git/tutorials/what-is-version-control> (Zugriff am: 12. Februar 2022.052Z).
- [8] Stack Overflow, *Stack Overflow Developer Survey 2021*. [Online]. Verfügbar unter: <https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-other-tools> (Zugriff am: 12. Februar 2022.384Z).
- [9] J. L. Zuckarelli, *Programmieren lernen mit Python und JavaScript: Eine praxisorientierte Einführung für Einsteiger*, 1. Aufl. Wiesbaden: Springer Fachmedien Wiesbaden; Imprint Springer Vieweg, 2021.

- [10] Axel Bruns: *Die Geschichte des Computers - ebook - neobooks*. [Online]. Verfügbar unter: <https://link.springer.com/content/pdf/10.1007%2F978-1-4842-6901-5.pdf> (Zugriff am: 25. November 2021.268Z).
- [11] D. Schaefer, *Eclipse CDT | The Eclipse Foundation*. [Online]. Verfügbar unter: <https://www.eclipse.org/cdt/> (Zugriff am: 3. Januar 2022.849Z).
- [12] *C++ programmieren mit Eclipse CDT*. [Online]. Verfügbar unter: <https://www.edv-buchversand.de/productinfo.php?replace=false&cnt=productinfo&mode=2&type=2&id=dp-196&index=2&nr=0&art=Blick%20ins%20Buch&preload=false&page=1&view=fit&Toolbar=1&pagemode=none> (Zugriff am: 3. Januar 2022.834Z).
- [13] S. Bauer, „Eclipse für C/C++-Programmierer – Handbuch zu den Eclipse C/C++ Development Tools (CDT)“.
- [14] JetBrains, *Intelligente Programmierunterstützung und Codeanalyse – Features | CLion*. [Online]. Verfügbar unter: <https://www.jetbrains.com/de-de/clion/features/> (Zugriff am: 29. November 2021.450Z).
- [15] CLion Help, *Quick start guide | CLion*. [Online]. Verfügbar unter: <https://www.jetbrains.com/help/clion/project.html> (Zugriff am: 8. Dezember 2021.446Z).
- [16] CLion Help, *File templates | CLion*. [Online]. Verfügbar unter: <https://www.jetbrains.com/help/clion/project-models.html> (Zugriff am: 8. Dezember 2021.086Z).
- [17] CLion Help, *File templates | CLion*. [Online]. Verfügbar unter: <https://www.jetbrains.com/help/clion/quick-cmake-tutorial.html> (Zugriff am: 8. Dezember 2021.931Z).
- [18] CLion Help, *File templates | CLion*. [Online]. Verfügbar unter: <https://www.jetbrains.com/help/clion/gradle-support.html> (Zugriff am: 8. Dezember 2021.412Z).

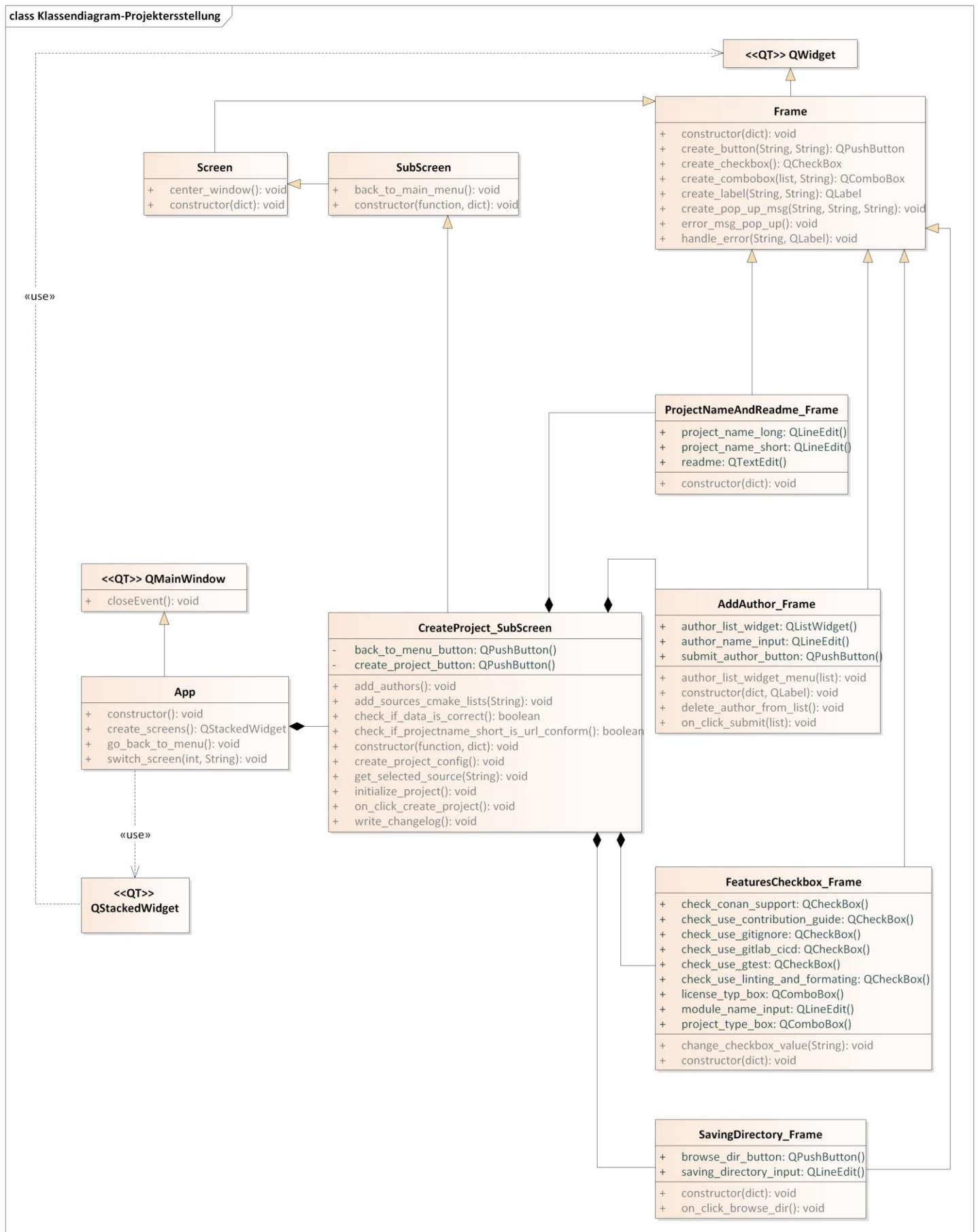
- [19] CLion Help, *File templates* | *CLion*. [Online]. Verfügbar unter:
<https://www.jetbrains.com/help/clion/makefiles-support.html> (Zugriff am: 8. Dezember 2021.280Z).
- [20] CLion Help, *File templates* | *CLion*. [Online]. Verfügbar unter:
<https://www.jetbrains.com/help/clion/compilation-database.html> (Zugriff am: 8. Dezember 2021.290Z).
- [21] CLion Help, *Open, reopen, and close projects* | *CLion*. [Online]. Verfügbar unter:
<https://www.jetbrains.com/help/clion/opening-reopening-and-closing-projects.html> (Zugriff am: 10. Dezember 2021.803Z).
- [22] CLion Help, *Quick CMake tutorial* | *CLion*. [Online]. Verfügbar unter:
<https://www.jetbrains.com/help/clion/creating-new-project-from-scratch.html>
(Zugriff am: 21. Dezember 2021.578Z).
- [23] CLion Help, *File templates* | *CLion*. [Online]. Verfügbar unter:
<https://www.jetbrains.com/help/clion/using-file-and-code-templates.html> (Zugriff am: 12. Januar 2022.637Z).
- [24] CLion Help, *Quick CMake tutorial* | *CLion*. [Online]. Verfügbar unter:
<https://www.jetbrains.com/help/clion/using-file-and-code-templates.html#create-new-template> (Zugriff am: 21. Dezember 2021.379Z).
- [25] A. Del Sole, *Visual Studio Code Distilled: Evolved Code Editing for Windows, macOS, and Linux*, 2. Aufl. Berkeley, CA: Apress; Imprint Apress, 2021.
- [26] *Managing Extensions in Visual Studio Code*. [Online]. Verfügbar unter:
<https://code.visualstudio.com/docs/editor/extension-marketplace> (Zugriff am: 14. Januar 2022.023Z).
- [27] *Visual Studio Code Tips and Tricks*. [Online]. Verfügbar unter:
https://code.visualstudio.com/docs/getstarted/tips-and-tricks#_files-and-folders
(Zugriff am: 29. November 2021.615Z).
- [28] *Project Templates - Visual Studio Marketplace*. [Online]. Verfügbar unter:
<https://marketplace.visualstudio.com/items?itemName=cantonios.project-templates> (Zugriff am: 29. November 2021.523Z).

- [29] *QMainWindow* — *Qt for Python*. [Online]. Verfügbar unter:
<https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QMainWindow.html> (Zugriff am: 28. Januar 2022.517Z).
- [30] *QMainWindow* — *Qt for Python*. [Online]. Verfügbar unter:
<https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QMainWindow.html> (Zugriff am: 24. Januar 2022.890Z).
- [31] *QStackedWidget* — *Qt for Python*. [Online]. Verfügbar unter:
<https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QStackedWidget.html?highlight=qstacked#PySide2.QtWidgets.PySide2.QtWidgets.QStackedWidget.setCurrentIndex> (Zugriff am: 24. Januar 2022.714Z).

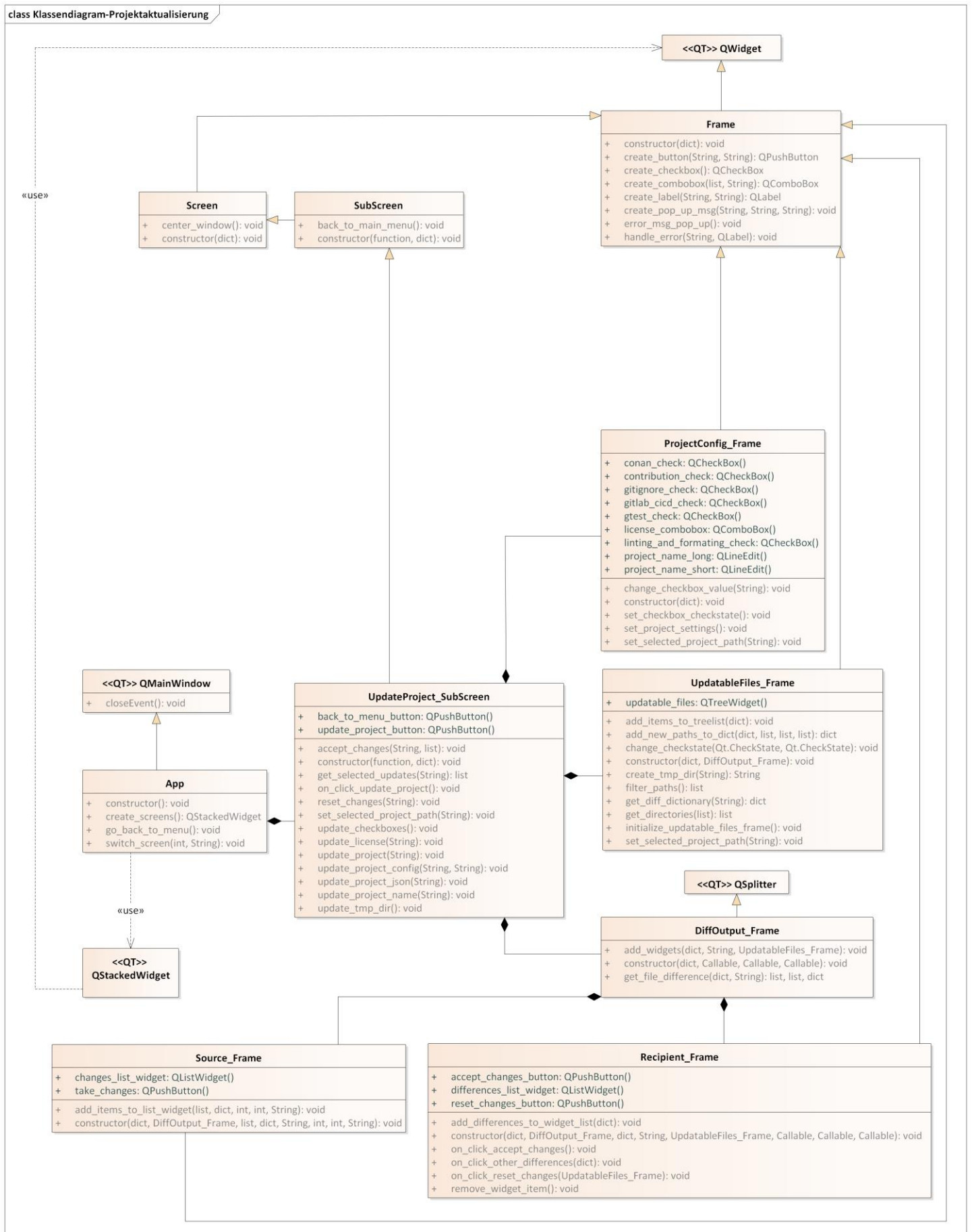
Anhang A: Klassendiagramm-Hauptmenü



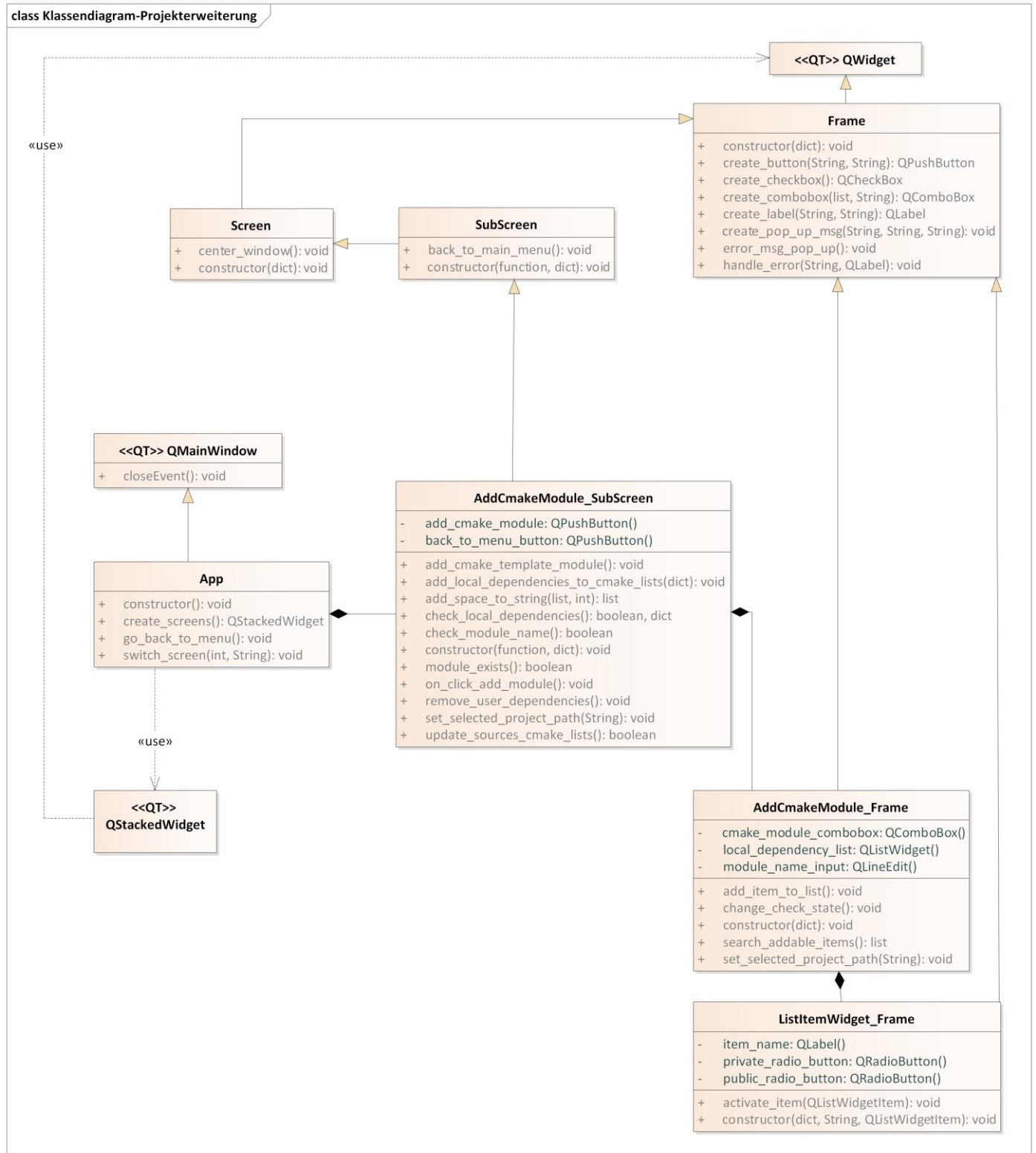
Anhang B: Klassendiagramm-Projekterstellung



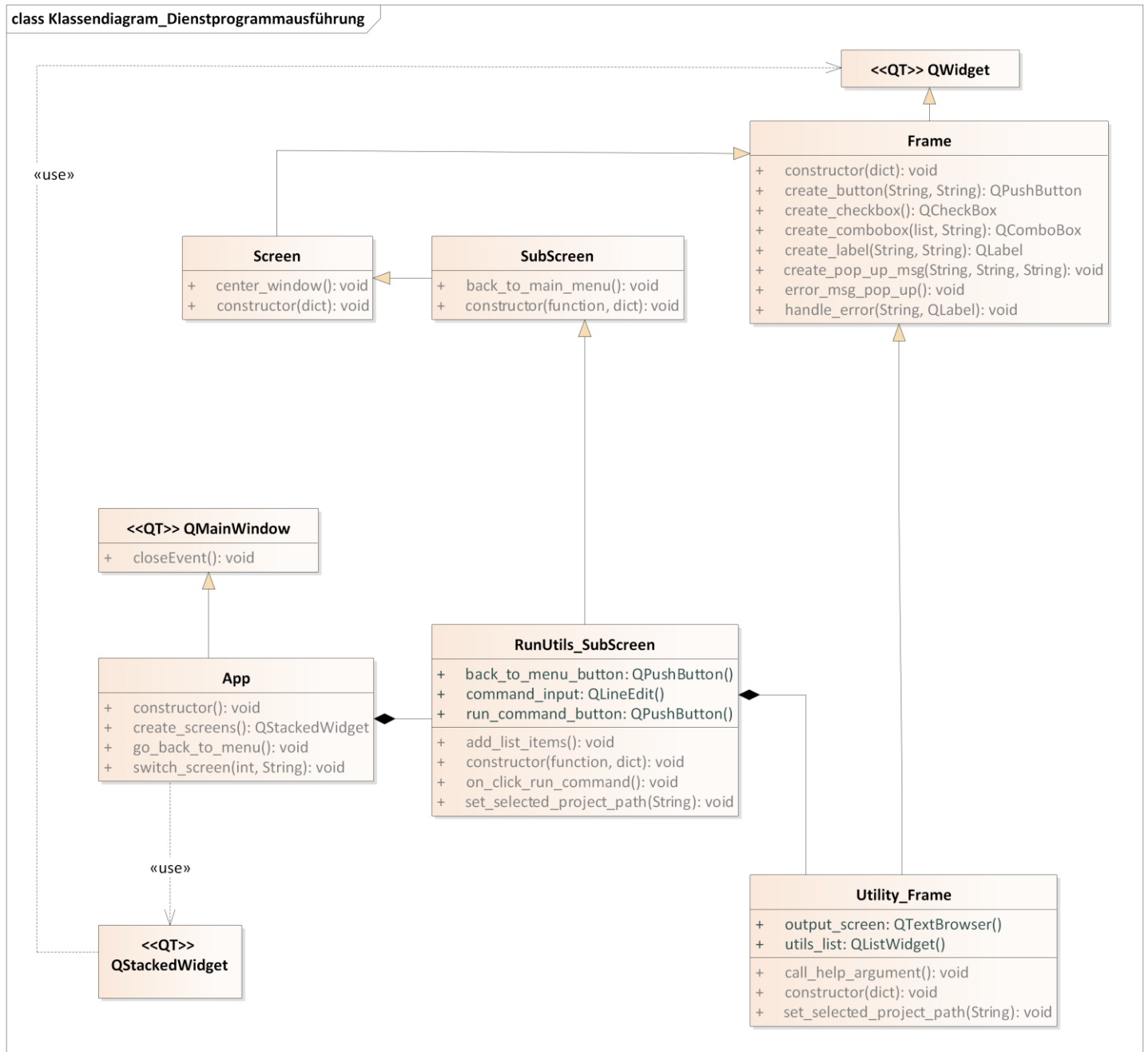
Anhang C: Klassendiagramm-Projektaktualisierung



Anhang D: Klassendiagramm-Projekterweiterung



Anhang E: Klassendiagramm-Dienstprogrammausführung



Anhang F: Klassendiagramm-Ladebildschirm

