**IU Internationale Hochschule**

Studiengang: Informatik M.Sc.

**Phase 1 und Phase 2: Projektdokumentation, Anforderungsdokument, Spezifikationsdokument, Architekturdokumen**

Portfolio zur Prüfung im Kurs Projekt: Software Engineering (DLMCSPSE01\_D)

eingereicht von: Kevin Walter

Matrikel-Nr.: 92212082

Tutor: Prof. Dr.-Ing. David Kuhlen

Datum: 03.07.2025

Inhaltsverzeichnis

[Phase 1: Konzeptionsphase – Projektdokument 1](#_Toc203296479)

[1. Projektidee und angestrebtes Ergebnis 1](#_Toc203296480)

[2. Potenzielle Risiken und Gegenmaßnahmen 1](#_Toc203296481)

[2.1 Skala: 1](#_Toc203296482)

[2.2 Tabelle potenzielle Risiken und Gegenmaßnahmen 1](#_Toc203296483)

[3. Zeitplanung 2](#_Toc203296484)

[Phase 1: Konzeptionsphase - Anforderungsdokument 3](#_Toc203296485)

[4. Stakeholder (Ziel- und Benutzergruppe) 3](#_Toc203296486)

[5. Funktionale Anforderungen 3](#_Toc203296487)

[5.1 Funktionsliste 3](#_Toc203296488)

[5.2 User Stories 3](#_Toc203296489)

[6. Nicht-funktionale Anforderungen 4](#_Toc203296490)

[7. Glossar 4](#_Toc203296491)

[Phase 1: Konzeptionsphase - Spezifikationsdokument 5](#_Toc203296492)

[8. Datenmodell 5](#_Toc203296493)

[8.1 Aufgabe 5](#_Toc203296494)

[8.2 UML-Klassendiagramm 5](#_Toc203296495)

[9. Geschäftsprozesse 6](#_Toc203296496)

[10. Geschäftsregeln 7](#_Toc203296497)

[11. Systemschnittstellen 8](#_Toc203296498)

[12. Benutzerschnittstellen 8](#_Toc203296499)

[12.1 Struktur der Oberfläche 8](#_Toc203296500)

[12.2 Wichtigste Dialoge & Abläufe 8](#_Toc203296501)

[12.3 Skizze der Anwendung 9](#_Toc203296502)

[Phase 2: Erarbeitungs- und Reflexionsphase – Architekturdokument 10](#_Toc203296503)

[13. Technologieübersicht 10](#_Toc203296504)

[13.1 Programmiersprache 10](#_Toc203296505)

[14. Architekturübersicht 11](#_Toc203296506)

[15. Struktur 11](#_Toc203296507)

[16. Verhalten 13](#_Toc203296508)

# Phase 1: Konzeptionsphase – Projektdokument

Link zum GitHub Repository: <https://github.com/KTWIU/SEProject>

# Projektidee und angestrebtes Ergebnis

Bevor der Zweck dieser Anwendung genannt wird, möchte ich ein Problem aus meinem Alltag nennen. Ich plane meinen Tag bisher noch auf einem klassischen Notizblock, was auf lange Sicht nicht nur unpraktisch, sondern auch ressourcenverschwendend ist. Zwar gibt es die „Erinnerungen“-App für das iPhone, jedoch fehlt mir eine einfache und schlanke Anwendung für den PC, die mich in meinem Alltag unterstützt. Die gängigen digitalen Tools sind häufig entweder zu kompliziert oder mit unnötigen Zusatzfunktionen überladen.

Die Zielsetzung dieses Portfolios ist es daher, eine schlanke Desktop-Anwendung für Windows-PCs zu entwickeln, mit der man seine Aufgaben für den nächsten Tag (oder auch die gesamte Woche) möglichst effizient und übersichtlich aufschreiben kann. Die Bedienung soll dabei so einfach wie möglich gehalten werden, also ohne unnötig viele Menüs oder Reiter, sondern mit einem klaren Fokus auf tägliche Aufgabenplanung, Deadlines und einer Kalenderansicht.

# Potenzielle Risiken und Gegenmaßnahmen

## 2.1 Skala:

|  |  |  |  |
| --- | --- | --- | --- |
| Wahrscheinlichkeit | niedrig | mittel | hoch |
| Schadensausmaß | niedrig | mittel | hoch |

## 2.2 Tabelle potenzielle Risiken und Gegenmaßnahmen

|  |  |  |  |
| --- | --- | --- | --- |
| Risiko | Eintrittswahrscheinlichkeit | Schadensausmaß | Gegenmaßnahme |
| Einarbeitung in Qt und CMake dauert zu lange | mittel | mittel | Früh mit Einarbeitung beginnen, Beispiele als Orientierung verwenden |
| Release-Package funktioniert nicht auf anderen Rechnern | hoch | hoch | windeployqt.exe verwenden,  damit alle DLLs und Dateien vorhanden sind |
| Projektumfang zu groß /  Zeitmangel | gering | mittel | Realistischer Zeitplan,  Fokus auf Kernfunktionen  (sh. Abschnitt 1) |
| Fehler durch Pointer /  Referenzen | mittel | hoch | Einfacher Code, Debugging-Tools und Testing |
| Codeverlust /  verschiedene Versionen | gering | hoch | Regelmäßige Pushs zu GitHub |

# Zeitplanung



# Phase 1: Konzeptionsphase - Anforderungsdokument

# Stakeholder (Ziel- und Benutzergruppe)

Die Anwendung richtet sich an alle Nutzer, die ihren Tag gerne strukturiert planen möchten, unabhängig davon ob sie Studenten, Berufstätige oder Privatpersonen sind. Das Programm bietet einen schnelle und unkomplizierten digitalen Ersatz für einen analogen Notizblock und unterstützt dabei, einen Überblick über anstehende Aufgaben und Deadlines zu behalten. Das Ziel ist es, die Vorteile moderner Aufgabenverwaltung am PC mit der Einfachheit eines Papier-Notizblocks zu verbinden, um papierlos und effizienter planen zu können.

# Funktionale Anforderungen

## 5.1 Funktionsliste

* Aufgaben anlegen, bearbeiten, löschen
* Deadlines setzen
* Aufgaben als erledigt markieren
* Kalender-/Tagesansicht
* Daten speichern/laden

## 5.2 User Stories

Als Benutzer möchte ich einfach per Texteingabe mit der Tastatur Aufgaben anlegen, bearbeiten und löschen können.

Als Benutzer möchte für meine Aufgabe auch ein Fälligkeitsdatum setzen können, damit ich mir nicht selbst notieren muss, bis wann ich die Aufgabe erledigen will. Es ist vorerst nicht vorgesehen, eine Benachrichtigung zu versenden, dass die Aufgabe überfällig ist. Über das heutige Datum und dem notierten Fälligkeitsdatum kann ich mir das selbst herleiten.

Als Benutzer möchte ich die Aufgabe als erledigt markieren können, damit ich direkt sehen kann, welche Aufgaben ich heute noch vor habe.

Als Benutzer möchte ich eine Kalender- und Tagesansicht sehen können, damit ich so beispielsweise auch eine ganze Woche planen kann.

Als Benutzer möchte ich die Daten speichern können, das heißt entweder per „save“ Button, oder mit automatischer Speicherung.

Als Benutzer muss ich mich nicht anmelden oder registrieren. Die Anwendung funktioniert wie ein klassisches Notizbuch: Ich kann direkt starten, ohne Account oder Internetverbindung. Die Aufgaben werden lokal auf meinem Windows-PC gespeichert. Eine Synchronisierung mit einer Cloud ist nicht vorgesehen.

Abbildung 1: Use-Case-Diagramm

Ein Bild, das Text, Screenshot, Schrift, Diagramm enthält.

KI-generierte Inhalte können fehlerhaft sein.

Quelle: Eigene Darstellung mit Hilfe von Visio

# Nicht-funktionale Anforderungen

* **Benutzerfreundlichkeit:** Die Anwendung soll eine einfache Bedienung und eine übersichtliche Oberfläche besitzen
* **Performance:** Die Anwendung ist schlank und überzeugt durch schnelle Ladezeiten, z. B. keine langen Ladezeiten für Reiter oder Seiten
* **Datensicherheit:** Alle Daten werden lokal auf dem PC gespeichert, es besteht keine Internetverbindung und keine Cloud-Synchronisierung. Dadurch sind die Daten bestmöglich vor externen Angriffen geschützt
* **Verschlüsselung:** Eine Verschlüsselung der Daten ist vorerst nicht vorgesehen
* **Plattform:** Die Anwendung wird als Windows Desktop-Anwendung bereitgestellt und nicht als Web-App
* **Eingabevalidierung:** Es wird eine Eingabevalidierung für Pflichtfelder und sinnvolle Werte umgesetzt

# Glossar

* **Aufgabe:** Ein Eintrag mit Titel, Beschreibung und Fälligkeitsdatum
* **Tagesansicht:** Auflistung aller an diesem Tag zu erledigenden/eingetragenen Aufgaben
* **Deadline:** Datum, bis zu dem die Aufgabe abgeschlossen sein soll
* **GUI:** Grafische Benutzeroberfläche
* **User** **Story:** Beschreibung aus Anwendersicht
* **UML:** Unified Modeling Language

# Phase 1: Konzeptionsphase - Spezifikationsdokument

# Datenmodell

## 8.1 Aufgabe

* Es ist das wichtigste Objekt
* Attribute einer Aufgabe sind: Titel, Beschreibung, Fälligkeitsdatum, Status (offen/erledigt) mit Anzeigen in grün/rot
* Benötigt mindestens einen Titel (Pflichtfeld), ein optionales Fälligkeitsdatum und einen Status (offen/erledigt), kann zusätzliche Beschreibung haben

## 8.2 UML-Klassendiagramm

Abbildung 2: UML-Klassendiagramm

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Quelle: Eigene Darstellung mit Hilfe von Visio

# Geschäftsprozesse

**Geschäftsprozess 1: Aufgabe anlegen**

1. Der Benutzer startet die Anwendung
2. Im Hauptfenster klickt er auf „Aufgabe hinzufügen“
3. Es öffnet sich ein Modal
   * Der Benutzer gibt Titel, Beschreibung, Fälligkeitsdatum ein
   * Mit „Speichern“ wird die Aufgabe übernommen
4. Die neue Aufgabe erschein in der Aufgabenliste

**Geschäftsprozess 2: Aufgabe als erledigt markieren**

1. Benutzer sieht die Aufgabenliste
2. Bei einer Aufgabe klickt er auf „Aufgabe als erledigt markieren“
3. Die Aufgabe wird als erledigt markiert und ggf. anders dargestellt (je nach Qt Funktionen)

**Geschäftsprozess 3: Überfällige Aufgaben anzeigen**

1. Beim Start prüft das Programm alle Aufgaben
2. Aufgaben mit Fälligkeitsdatum < wie aktuelles Datum werden als „überfällig“ markiert/angezeigt

**Geschäftsprozess 4: Aufgabe löschen**

1. In der Aufgabenliste wählt der Nutzer eine Aufgabe aus
2. Mit Klick auf „Aufgabe löschen“ wird die Aufgabe entfernt (komplett gelöscht)

Abbildung 3: UML-Aktivitätsdiagramm für Kernprozess "Aufgabe anlegen"

Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.

Quelle: Eigene Darstellung mit Hilfe von Visio

# Geschäftsregeln

* Eine Aufgabe muss immer mindestens einen Titel besitzen
* Das Fälligkeitsdatum einer Aufgabe darf nicht in der Vergangenheit liegen
* Eine Aufgabe kann erst gespeichert werden, wenn alle Pflichtfelder (Titel) ausgefüllt sind
* Jede Aufgabe ist eindeutig identifizierbar (z.B. durch eine ID)
* Bereits erledigte Aufgaben können nicht bearbeitet werden (nur noch löschen möglich)
* Das Fälligkeitsdatum von bereits erledigten Aufgaben führt nicht mehr zu einem Zustandswechsel (bspw. Verändern der Schrift von grün auf rot o.Ä.)
* Beim Löschen einer Aufgabe öffnet sich ein Modal, um unbeabsichtigtes löschen zu verhindern
* Alle angegebenen Daten werden beim Schließen der Anwendung gespeichert
* Das Fälligkeitsdatum ist optional, aber wenn es gesetzt wird, muss es nach dem Erstellungsdatum liegen

# Systemschnittstellen

In diesem Projekt sind keine externen technischen Schnittstellen (z.B. HTTP, FTP) vorgesehen. Die Anwendung läuft komplett lokal auf dem Rechner des Benutzers. Der Datenformat der Aufgaben wird zum Beispiel TXT, CSV oder JSON sein.

# Benutzerschnittstellen

Die Anwendung bietet eine schlanke grafische Benutzeroberfläche (GUI), die für einfache Bedienung und Übersichtlichkeit ausgelegt ist, ohne viele Reiter / Dialoge.

## 12.1 Struktur der Oberfläche

* Hauptfenster: Zeigt alle Aufgaben übersichtlich als Liste
* Neue Aufgabe anlegen: Button „+ Aufgabe anlegen“ öffnet Dialog
* Aufgabendetails: Im Dialog kann der Nutzer die Attribute eingeben
* Löschen/Erledigt: Neben jeder Aufgabe befinden sich Buttons für „Löschen“ oder „als erledigt markieren“
* Kalender: Der Kalender soll im Hauptfenster angezeigt werden

## 12.2 Wichtigste Dialoge & Abläufe

**Neue Aufgabe anlegen**

* Klick auf „+ Aufgabe hinzufügen“
* Eingabefeld für Titel (Pflichtfeld)
* Eingabefeld für Fälligkeitsdatum (optional)
* Eingabefeld für Beschreibung (optional)
* „Speichern“ Button, der nur aktiv ist, wenn das Pflichtfeld ausgefüllt ist
* Fehlermeldung, falls Pflichtfeld leer bleibt oder Datum ungültig ist

**Aufgabe löschen**

* Klick auf „Löschen“-Button öffnet ein Modal/Bestätigungsdialog und erst danach wir die Aufgabe gelöscht

**Eingabevalidierung**

* Titel muss ausgefüllt sein
* Fälligkeitsdatum darf nicht in der Vergangenheit liegen

## 12.3 Skizze der Anwendung

Abbildung 4: Skizze Aufgabenplaner

Ein Bild, das Text, Screenshot, Rechteck, Design enthält.

KI-generierte Inhalte können fehlerhaft sein.

Quelle: Eigene Darstellung mit Hilfe von Visio

# Phase 2: Erarbeitungs- und Reflexionsphase – Architekturdokument

# Technologieübersicht

## Programmiersprache

Die für dieses Projekt ausgewählte Programmiersprache ist C++, folgend werden die wichtigsten Gründe für die Auswahl erläutert.

* Motivation: Vertiefung der eigenen Kenntnisse, wichtig für Ingenieurs- und Software-Engineering Berufe
* Fördert allgemeines Verständnis für Speicherverwaltung, effiziente Algorithmen und systemnahe Entwicklung
* Große Ökosystem, viele Libraries und plattformübergreifend einsetzbar
* Auch wenn Entwicklung schwerer als bspw. mit Python ist, ist der Lerneffekt für Studium/Beruf höher
* GUI-Anbindung mit Qt

## 13.2 Frameworks

In diesem Projekt werden primär die Frameworks Qt (Qt6) und googletest verwendet. In diesem Abschnitt werden die Gründe dafür erklärt.

* Qt (Qt6)
  + Wird für grafische Oberfläche (GUI) verwendet
  + Ermöglicht plattformübergreifende Desktop-Entwicklung
  + Gute Unterstützung für UI-Design, Events und eigene Widgets
  + Integration in C++ Projekt und Build-System (CMake) ist Standard
* googletest
  + Framework für automatisierte Unit-Tests
  + Weit verbreitet im C++ Umfeld, auch im industriellen Einsatz
  + Moderne C++ Features für Testfälle und Test Suites
  + Bringt professionelle Entwicklungsmethodik (Test Driven Development, Absicherung von Code…)

## 13.3 Bibliotheken

Bisher werden im Projekt folgende Bibliotheken verwendet, diesen können sich jedoch je nach Projektablauf noch verändern.

* C++ Standardbibliothek (STL): Für Vektoren, Strings, Maps usw.
* QtCore, QtGui, QtWidgets

## 13.4 Entwicklungswerkzeuge

* VS Code mit C++ und Qt Extension
* Qt Creator 17.0.0
* CMake als Buildsystem
* GitHub zur Versionskontrolle
* windeployqt zum Erstellen von Release-Paket (Deployment für Windows) 🡪 wird in Phase 3 implementiert

# Architekturübersicht

In diesem Projekt habe ich mich auf eine Schichtenarchitektur entschieden (3-Layer). Diese ist übersichtlich, leicht erweiterbar und testbar. Für das Projekt ist diese Architektur meiner Meinung nach ausreichend.

* **UI-Schicht:** Besteht aus den Klassen MainWindow und den Dialogklassen
  + Zuständig für grafische Anzeige, Benutzerinteraktion und Aufruf der Logik
* **Logikschicht:** Hauptkomponente ist Klasse TaskManager
  + Aufgaben hinzufügen, löschen, bearbeiten, speichern…
  + Schnittstellt zwischen UI und Datenhaltung
* **Datenhaltung:** durch Klasse Task und Datei-Verwaltung innerhalb TaskManager realisiert
  + Aufgaben werden lokal als .csv gespeichert

# Struktur

## 15.1 Hauptkomponenten

* MainWindow
  + Verantwortlich für das Hauptfenster und die Steuerung der Oberfläche
  + Zeigt die Liste der offenen Aufgaben und bietet Buttons für Aktionen
  + Öffnet bei Bedarf die verschiedenen Dialogfenster
* DialogTaskHinzufuegen / DialogTaskBearbeiten
  + Separate Dialogfenster für das Hinzufügen bzw. Bearbeiten einer Aufgabe
  + Erfassen und Bearbeiten der Aufgabendaten durch den Benutzer
* erledigteAufgabenliste / ueberfaelligeAufgaben
  + Dialogfenster zur Anzeige aller erledigten bzw. überfälligen Aufgaben
* TaskManager
  + Verwalten der Sammlung aller Aufgaben (Hinzufügen, Löschen, Suchen, Bearbeiten)
  + Schnittstelle zwischen UI und Dateispeicherung
  + Verantwortlich für das Speichern und Laden der Aufgaben
* Task
  + Datenklasse für einzelne Aufgaben (Attribute: Titel, Beschreibung, Fälligkeitsdatum, Status, Index)
  + Methoden zum Ändern und Auslesen der Aufgabendaten

## 15.2 Abhängigkeiten

* MainWindow nutzt TaskManager für Logik und Aufgabeliste
* TaskManager verwaltet eine Sammlung von Task-Objekten
* MainWindow öffnet die Dialogfenster zur Interaktion mit dem Benutzer
* Dialogfenster übergeben ihre Eingaben an MainWindow/TaskManager

Abbildung 5: UML-Klassendiagramm

**Ein Bild, das Text, Screenshot, Software, Design enthält.

KI-generierte Inhalte können fehlerhaft sein.**

Quelle: Eigene Darstellung mit Hilfe von Visio

# Verhalten

1. Der Benutzer gibt die Aufgabendaten im Dialog ein
2. MainWindow ruft addAufgabe im TaskManager auf
3. TaskManager legt ein neues Task-Objekt an und speichert alle Aufgaben in der Datei
4. MainWindow aktualisiert die Aufgabenliste in der Benutzeroberfläche

Abbildung 6: UML-Sequenzdiagramm

Ein Bild, das Text, Screenshot, Reihe, parallel enthält.

KI-generierte Inhalte können fehlerhaft sein.

Quelle: Eigene Darstellung

# Phase 3: Finalisierungsphase – Testdokument

# Teststrategie

## 17.1 Unit-Tests

Zum Testen der zentralen Kernfunktionen wurden automatisierte Unit-Tests mit Googletest geschrieben, wie in Abschnitt 13.2 bereits erwähnt. Konkret wurden folgende Funktionen getestet:

* Eingabevalidierung für Titel und Datum (T01, T02)
* TaskManager Methoden für Hinzufügen und Entfernen einer Aufgabe (T03, T04)

Ziel der Unit-Tests war es, sicherzustellen, dass die wichtigsten Kernfunktionen der Anwendung funktionieren (z. B. addAufgabe).

## 17.2 Integrationstests

Zusätzlich wurden Integrationstests durchgeführt, bei denen ein typischer Workflow aus Anwendersicht getestet wurde. Dabei wurde automatisiert getestet:

* Aufgabe anlegen (T05)
* Angelegte Aufgabe bearbeiten (T05)

Außerdem wurde ergänzend getestet, ob nach dem Markieren einer Aufgabe als erledigt diese wie vorhergesehen aus dem ListWidget der offenen Aufgaben verschwindet (T05). Diese Funktion lässt sich nur manuell in der GUI überprüfen und wird daher mit einem Screenshot im Testprotokoll hinterlegt.

Das Ziel der Integrationstests ist es, sicherzustellen, dass das Zusammenspiel der Anwendungsabläufe vom Benutzer wie erwartet funktionieren.

## 17.3 Systemtests

Die Systemtests wurden komplett manuell durchgeführt. Dabei wurde die komplette Anwendung inklusive grafischer Benutzeroberfläche getestet. Die Testergebnisse werden zur Nachvollziehbarkeit durch Screenshots dokumentiert und dem Testprotokoll beigefügt. Folgende Funktionen wurden geprüft:

* Aufgabe anlegen und bearbeiten über GUI (S01, S02)
* Aufgabe als erledigt markieren (S03)
* Überfällige Aufgaben anzeigen (S04)
* Speichern und Laden der Aufgaben beim Schließen und Öffnen der Anwendung (S05)
* Aufgabe löschen über GUI (S06)

Ziel der Systemtests ist es, die Anwendung unter realen Einsatzbedingungen (nicht aus dem Testumfeld heraus) zu prüfen und sicherzustellen, dass sie aus Anwendersicht stabil und bedienbar ist.

# Testprotokoll

Tabelle 1: Unit-Tests und Integrationstest

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test-ID** | **Kurzbeschreibung** | **Vorbedingung** | **Aktion** | **Erwartetes Ergebnis** | **Tatsächliches Ergebnis** |
| T01 | Prüfung, ob Titel leer ist | Keine | Titel = “Mein Titel“;  Titel = ““ | True für “Mein Titel“;  False für ““ | Wie erwartet |
| T02 | Prüfung, ob Fälligkeitsdatum heute oder später ist | Keine | Datum = gestern;  Datum = heute | False für gestern;  True für heute | Wie erwartet |
| T03 | Aufgabe hinzufügen erhöht Aufgabenanzahl (Index) | Leerer TaskManager | addAufgabe(…) | Aufgabenanzahl + 1 | Wie erwartet |
| T04 | Aufgabe löschen verringert die Aufgabenzahl | Mindestens eine Aufgabe vorhanden | delAufgabe(letzter Index) | Aufgabenanzahl – 1 | Wie erwartet |
| T05 | Typischer Anwender-Workflow: Aufgabe anlegen und bearbeiten; Aufgabe als erledigt markieren | Leerer TaskManagere | addAufgabe(…), danach editAufgabe(…) mit anderen Werten; Aufgabe als erledigt markieren | Aufgabe hat neue Werte (nach edit) und Aufgabenanzahl bleibt 1; Aufgabe verschwindet in Liste der erledigten Aufgaben | Wie erwartet |

Quelle: Eigene Darstellung

**Zusatz Integrationstest T05**

* Schritt 1 (Vorbereitung): Aufgabe anlegen
* Schritt 2: ausgewählte Aufgabe als erledigt markieren
* Schritt 3: Prüfen, ob die Aufgabe in der Liste der erledigten Aufgaben ist

Abbildung 7: Integrationstest T05

Ein Bild, das Text, Screenshot, Software, Webseite enthält.

KI-generierte Inhalte können fehlerhaft sein.

Quelle: Eigene Darstellung

**Systemtests**

Tabelle 2: Tabelle Systemtests

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test-ID** | **Kurzbeschreibung** | **Vorbedingung** | **Aktion** | **Erwartetes Ergebnis** | **Tatsächliches**  **Ergebnis** |
| S01 | Aufgabe anlegen (GUI) | Anwendung geöffnet | Über GUI Aufgabe anlegen | Aufgabe erscheint in Liste | Wie erwartet |
| S02 | Aufgabe bearbeiten (GUI) | Aufgabe vorhanden | Aufgabe per GUI bearbeiten | Änderungen werden übernommen | Wie erwartet |
| S03 | Aufgabe als erledigt markieren (GUI) | Aufgabe vorhanden | Aufgabe per GUI als erledigt markieren | Aufgabe verschwindet aus offener Liste | Wie erwartet |
| S04 | Überfällige Aufgaben anzeigen | Überfällige Aufgabe vorhanden | Überfällige Aufgabe über GUI anzeigen | Überfällige Aufgabe wird korrekt markiert und in Liste angezeigt | Wie erwartet |
| S05 | Speichern und Laden testen | Aufgabe vorhanden | Anwendung schließen und öffnen | Aufgaben werden korrekt geladen | Wie erwartet |

Quelle: Eigene Darstellung

Abbildung 8: Systemtest S01 - Aufgabe anlegen

Ein Bild, das Text, Screenshot, Software, Computersymbol enthält.

KI-generierte Inhalte können fehlerhaft sein.

Quelle: Eigene Darstellung

Abbildung 9: Systemtests S02 - Aufgabe bearbeiten

Ein Bild, das Text, Screenshot, Software, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.

Quelle: Eigene Darstellung

Abbildung 10: Systemtests S03 - Aufgabe erledigt

Ein Bild, das Text, Screenshot, Software, Webseite enthält.

KI-generierte Inhalte können fehlerhaft sein.

Quelle: Eigene Darstellung

Abbildung 11: Systemtests S04 - Aufgabe überfällig

***Ein Bild, das Text, Screenshot, Software, Webseite enthält.

KI-generierte Inhalte können fehlerhaft sein.***

Quelle: Eigene Darstellung

Abbildung 12: Systemtests S05 - Speichern/Laden

Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.

Quelle: Eigene Darstellung

Abbildung 13: Systemtests S06 - Aufgabe löschen/entfernen

**Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.**

Quelle: Eigene Darstellung

# Benutzeranleitung

Damit die Anwendung auf jedem Windows-PC ohne weitere Qt-Installation läuft, müssen alle not-wendigen DLLs im selben Ordner wie die .exe liegen. Folgende DLLs werden zum Ausführen der Anwendung benötigt beziehungsweise sind empfohlen.

• Qt6Core.dll

• Qr6Gui.dll

• Qt6Widgets.dll

• D3Dcompiler\_47.dll

• libgcc\_s\_seh-1.dll

• libstdc++-6.dll

• libwinpthread-1.dll

• opengl32sw.dll

Diese Dateien sowie alle von windeployqt automatisch erzeugten Unterordner (z. B. platforms, imageformats, translations usw.) befinden sich bereits im bereitgestellten ZIP-Ordner.

Die Anwendung kann nach dem Entpacken direkt per Doppelklick auf die EXE gestartet werden, eine weitere Installation ist nicht notwendig.

# Abstract

Im vorliegenden Portfolioprojekt wird eine Aufgabenplaner-Anwendung für Windows entwickelt. Die Anwendung wurde mit C++ und dem Qt Framework erstellt und ermöglicht es Privatpersonen, Berufstätigen und Studenten ihren Tag einfach zu planen. Typische Anforderungen an den Aufgabenplaner sind das Anlegen, Bearbeiten, Löschen und Filtern von Aufgaben. Dabei stand eine besonders benutzerfreundliche Oberfläche im Vordergrund, sowie die grundlegenden Prinzipien der objektorientierten Programmierung praktisch anzuwenden. Das Projekt wurde bis auf das durch den Dozenten erhaltene Feedback komplett selbst durchgeführt. Nicht nur die Anwendung stand im Mittelpunkt, sondern der gesamte Entwicklungsprozess inklusive dem definieren der Anforderungen, Testen, Deployment usw.

Zu Beginn, in Phase 1, stand die Konzeption der Anwendung an. Im ersten Schritt habe ich mir potenziellen Risiken einfallen lassen und Gegenmaßnahmen definiert. Keine der Risiken sind aufgetreten, trotzdem war dieser Schritt wichtig, denn entsprechend den Risiken plant man das Projekt bereits anders. Anschließend habe ich eine Zeitplanung mit Gantt-Diagramm erstellt, der Zeitplan wurde größtenteils eingehalten. Nach der Zielgruppenanalyse habe ich die funktionalen und nicht funktionalen Anforderungen definiert, welche alle eingehalten wurden. Entsprechend fiel die Entscheidung für eine klassische Desktop-Anwendung auf Basis von C++ und dem Qt-Framework, um sowohl die grafische Oberfläche als auch die darunterliegende Logik sauber voneinander zu trennen.

Im weiteren Verlauf des Projekts stand dann die objektorientierte Programmierung im Mittelpunkt. Die Klassen *Task* und *TaskManager* wurden entwickelt, um die Aufgaben als Objekte abbilden zu können und diese dann innerhalb der Anwendung verwalten zu können. Für die verschiedenen Dialog Fenster (z.B. Entfernen und Hinzufügen von Aufgaben) musste jeweils ein eigenes Objekt mit Logik angelegt werden. Die Programmstruktur sollte eigentlich noch mehr von Vererbungen profitieren, zum Beispiel mit einer Basisklasse für alle Dialoge und gemeinsamen Methoden (z.B. setTitel()). Jedoch sind mir viele dieser Sachen erst aufgefallen, nachdem ich die Anwendung programmiert habe. Eine Änderung des gesamten Codes wäre hier einfach zu aufwendig, diese Erkenntnisse nehme ich aber für das nächste Projekt mit.

Auch der Umgang der Datei- und Pfadangaben sowie allgemein die Ordnerstruktur waren ein zentrales Thema, was ich deutlich unterschätzt habe. Während der Implementierung kam es immer wieder zu Problemen, die Möglicherweise von der Verwendung eines OneDrive Ordners kamen. Nach Umstellung auf einen lokalen Ordner sind weitere Probleme aufgetreten, obwohl sich die Anwendung aus QtCreator heraus bereits starten ließ. Das größte Problem war das einbinden von Header-Dateien, da ich anfangs als absolute Pfade eingebunden habe. Beim wechseln zwischen VS Code und QtCreator, sowie beim verschieben des Ordners aus OneDrive entstanden viele Fehlermeldungen, die teilweise schwer zu debuggen waren. In meinem nächsten Projekt werde ich mir bereits in der Konzeptionsphase Gedanken zur Ordnerstruktur machen und überlegen wie und wo ich bestmöglich Frameworks wie googletest oder Qt einbinde.

In Phase 3 wurde dann größtenteils überlegt, welche Tests Sinn ergeben. Ich habe gezielt Tests geschrieben, vor allem die Eingabevalidierung und das Hinzufügen oder Bearbeiten von Aufgaben automatisiert testen. Beim Verwenden von Googletest war es wichtig zu unterscheiden, welche Tests automatisiert werden können und welche manuell durchgeführt werden müssen. Die Automatisierten Tests basieren vor allem auf dem Testen der Logik, was mit Hilfe der Expected Assertions einfach möglich ist. Die GUI-bezogenen Abläufe lassen sich jedoch nicht so leicht automatisieren, weshalb diese mit Screenshots im Testprotokoll dokumentiert wurden.

Ein weiterer Knackpunkt war das Thema Deployment. Damit Qt reibungslos Anwendungen aus anderen Rechnern ausführen kann, benötigt es bestimmte DLLs Dateien und deren Abhängigkeiten. Anfangs führte das zu vielen Fehlermeldungen, da ich nicht die entsprechenden DLLs Dateien eingebunden habe. Durch das Tool *windeployqt* kann das Deployment jedoch deutlich erleichtert werden. Windeployqt kopiert automatisch alle benötigten DLLs Dateien in den Build-Ordner, welcher später für den Release auf anderen Rechnern verwendet werden kann.

Im Zeitplan gab es nur kleine Verzögerungen, die beispielsweise auf wiederholte Build-Probleme oder Probleme mit verschiedenen Compilern (MinGW & MSVC) zurückzuführen sind, jedoch keine gravierenden Verzögerungen. Bei zukünftigen Projekten würde ich im Allgemeinen mehr Pufferzeit einbauen, aber auch für das Deployment genauer betrachten.

Abschließend wurde das Projektziel trotz einiger Probleme erreicht, die Anwendung läuft stabil, die Kernfunktionen sind umgesetzt und durch Tests abgesichert und auch das Deployment ist gelöst. Für mich war das Projekt eine wichtige Erfahrung in Sachen praktischer Softwareentwicklung die ich hoffentlich auch so in das Berufsleben mitnehmen kann. Besonders wertvoll waren für mich die Erkenntnisse aus GitHub, den Build-Systemen, Pfadmanagement und der Testautomatisierung. Für weitere Projekte würde ich von Anfang an eine klare OOP-Struktur, besseres Testdesign und eine saubere Projektorganisation (Pfadmanagement usw.) umsetzen. Auch die Dokumentation und Reflexion im Rahmen des Portfolios hat mir deutlich weiter geholfen, beispielsweise um Schwachstellen zu erkennen und konkrete Verbesserungsansätze für kommende Projekte abzuleiten.