**IU Internationale Hochschule**

Studiengang: Informatik M.Sc.

**Phase 3: Abstract**

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

[1. Abstract 3](#_Toc204091807)

# 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 bauen ließ. Das größte Problem war das Einbinden von Header-Dateien, die 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, die 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 auf 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 das Thema 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.