**IU Internationale Hochschule**

Studiengang: Informatik M.Sc.

**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: 19.08.2025

Inhaltsverzeichnis

[1. Abstract 3](#_Toc206496955)

# 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. Die Oberfläche wurde benutzerfreundlich gestaltet und dient gleichzeitig als praktisches Beispiel für die Anwendung objektorientierter Programmierung. Im Fokus stand nicht nur die Anwendung selbst, sondern auch der gesamte Entwicklungsprozess inklusive der Anforderungsdefinition, Testen und Deployment.

Die entwickelte Architektur basiert auf dem MVC-Muster. Das Model (TaskManager) verwaltet alle Aufgaben, kapselt die Geschäftslogik und speichert die Daten dauerhaft in einer CSV-Datei (Persistenz). Die View (Dialog- und Hauptfenster) stellt die grafische Benutzeroberfläche bereit. Für die Erzeugung neuer Aufgaben kommt eine Factory-Klasse (TaskFactory) zum Einsatz, um eine lose Kopplung zu gewährleisten.

Zu Beginn der Projektarbeit (Phase 1) stand die Konzeption an. Potenzielle Risiken wurden mit Frühwarnindikatoren identifiziert und Gegenmaßnahmen definiert. Außerdem wurde eine Zeitplanung mittels Gantt-Diagramm erstellt, welche vor allem wegen fehlender Puffer deutlich zu knapp geplant wurde. Danach wurde eine Zielgruppenanalyse durchgeführt und funktionale und nicht-funktionale Anforderungen festgelegt. Auf dieser Grundlage fiel die Entscheidung für eine klassische Desktop-Applikation mit C++ und Qt, um Oberfläche und Logik sauber zu trennen.

Im weiteren Verlauf lag der Schwerpunkt auf der Umsetzung des Designs. Klassen wie Task und TaskManager wurden entwickelt, um Aufgaben als Objekte abzubilden und innerhalb der Anwendung verwalten zu können. Für die verschiedenen Dialogfenster (z. B. Hinzufügen und Entfernen) wurden separate Klassen mit spezifischer Logik erstellt. Die Programmstruktur könnte künftig noch stärker von Vererbungen profitieren (etwa durch eine Basisklasse für Dialoge), diese Erkenntnis wird für kommende Projekte berücksichtigt.

Der Umgang mit Dateien und Pfadangaben sowie die Ordnerstruktur stellte sich als zentrales Thema heraus: Die Nutzung eines OneDrive‑Ordners führte zu Build‑Problemen, die erst nach dem Wechsel auf einen lokalen Ordner und der Umstellung von absoluten auf relative Include-Pfadangaben behoben werden konnten. Für zukünftige Projekte empfiehlt es sich, frühzeitig eine klare Ordnerstruktur festzulegen und die Einbindung externer Frameworks wie GoogleTest oder Qt systematisch auf jeder Ebene zu planen.

In Phase 3 wurde der Schwerpunkt auf die Qualitätssicherung gelegt. Es wurden automatisierte Unit-Tests entwickelt, welche die Kernfunktionen des Models abdecken. Die Erzeugung von Aufgaben über die Factory, das Hinzufügen, Bearbeiten und Löschen von Aufgaben, das Setzen des Erledigt-Status sowie das Speichern und Laden der Daten wurden als Unit-Tests getestet.

Ein Integrationstest prüft den typischen Workflow (Anlegen, Bearbeiten, Erledigen), während Systemtests die GUI-Abläufe dokumentieren, die sich nicht automatisieren lassen. Ein weiterer Knackpunkt war das Deployment. Damit Qt‑Anwendungen auf anderen Rechnern lauffähig sind, müssen die benötigten DLLs eingebunden werden. Anfangs führte das zu Fehlermeldungen, mit dem Tool windeployqt ließ sich das Deployment jedoch automatisieren, indem alle erforderlichen Bibliotheken in den Release-Ordner kopiert wurden.

Die Verzögerungen im Zeitplan traten vor allem durch wiederholte Build-Probleme oder den Wechsel zwischen den Compilern MinGW und MSVC auf, blieben jedoch ohne gravierende Auswirkungen. Das Projektziel eine stabile, getestete Aufgabenplaner-Anwendung mit klarer MVC-Struktur, lokaler Persistenz und benutzerfreundlicher Oberfläche zu entwickeln wurde erreicht. Das Projekt bot wertvolle Erfahrungen im Umgang mit GitHub, Architekturentscheidungen, Syntax, Testautomatisierung und Deployment. Für kommende Vorhaben empfiehlt sich, von Beginn an eine klare OOP-Struktur, ein umfassendes Testdesign und eine sorgfältige Projektorganisation zu etablieren; die Portfoliodokumentation zeigte, wie hilfreich Reflexion und Dokumentation sind, um Schwachstellen zu erkennen und konkrete Verbesserungsansätze zu erkennen.