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

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

1. Projektidee und angestrebtes Ergebnis

Zu Beginn wird ein alltägliches Problem beschrieben. Die Tagesplanung erfolgt häufig auf einem klassischen Notizblock, dieses Vorgehen ist auf Dauer unpraktisch und ressourcenintensiv. Zwar existiert die Erinnerungen-App für das iPhone, jedoch fehlt eine einfache und schlanke Anwendung für den PC, die die Planung im Alltag unterstützt. Viele digitale Tools sind entweder zu komplex oder bieten unnötig viele Zusatzfunktionen.

Zielsetzung dieses Portfolios ist die Entwicklung einer schlanken Desktop-Anwendung für Windows-PCs, mit der Aufgaben für den kommenden Tag oder die gesamte Woche effizient und übersichtlich geplant werden können. Die Benutzeroberfläche ist bewusst einfach gehalten, ohne überflüssige Menüs oder Reiter, mit klarem Fokus auf die tägliche Aufgabenplanung, Deadlines und eine Kalenderansicht.

2. Potenzielle Risiken und Gegenmaßnahmen

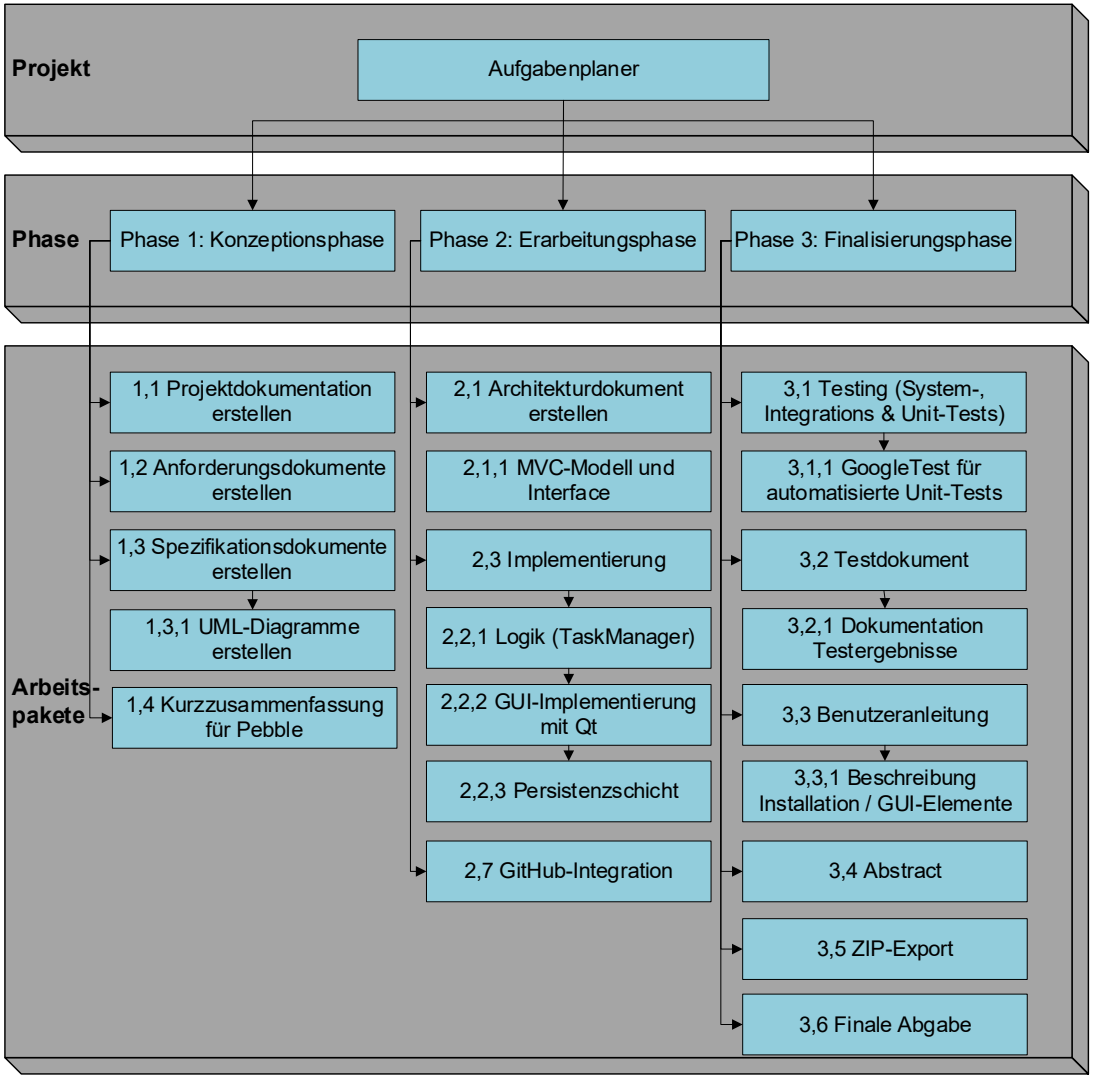
2.1 Skala:

Wahrscheinlichkeit	niedrig	mittel	hoch
Schadensausmaß	niedrig	mittel	hoch

2.2 Tabelle potenzielle Risiken und Gegenmaßnahmen

Risiko	Eintrittswahrscheinlichkeit	Frühwarnindikator	Schadensausmaß	Gegenmaßnahme
Einarbeitung dauert zu lange	mittel	Erste GUI-Tests >2 Tage verspätet	mittel	Früh mit Einarbeitung beginnen, Beispiele als Orientierung verwenden
Release-Package funktioniert nicht auf anderen Rechnern	hoch	Anwendung startet nicht auf Test-PC	hoch	windeployqt verwenden, damit alle DLLs und Dateien vorhanden sind
Projektumfang zu groß / Zeitmangel	gering	Aufgabenrückstand in Woche 2	mittel	Realistischer Zeitplan, Fokus auf Kernfunktionen (sh. Abschnitt 1)
Fehler durch Pointer / Referenzen	mittel	Wiederholte Abstürze oder Warnungen	hoch	Einfacher Code, Debugging-Tools und Testing
Codeverlust / verschiedene Versionen	gering	Mehrere lokale Kopien auf Laufwerk	hoch	Regelmäßige Pushs zu GitHub (GitHub, Version 2.46.0.windows.1)

3. Projektstrukturplan (PSP)

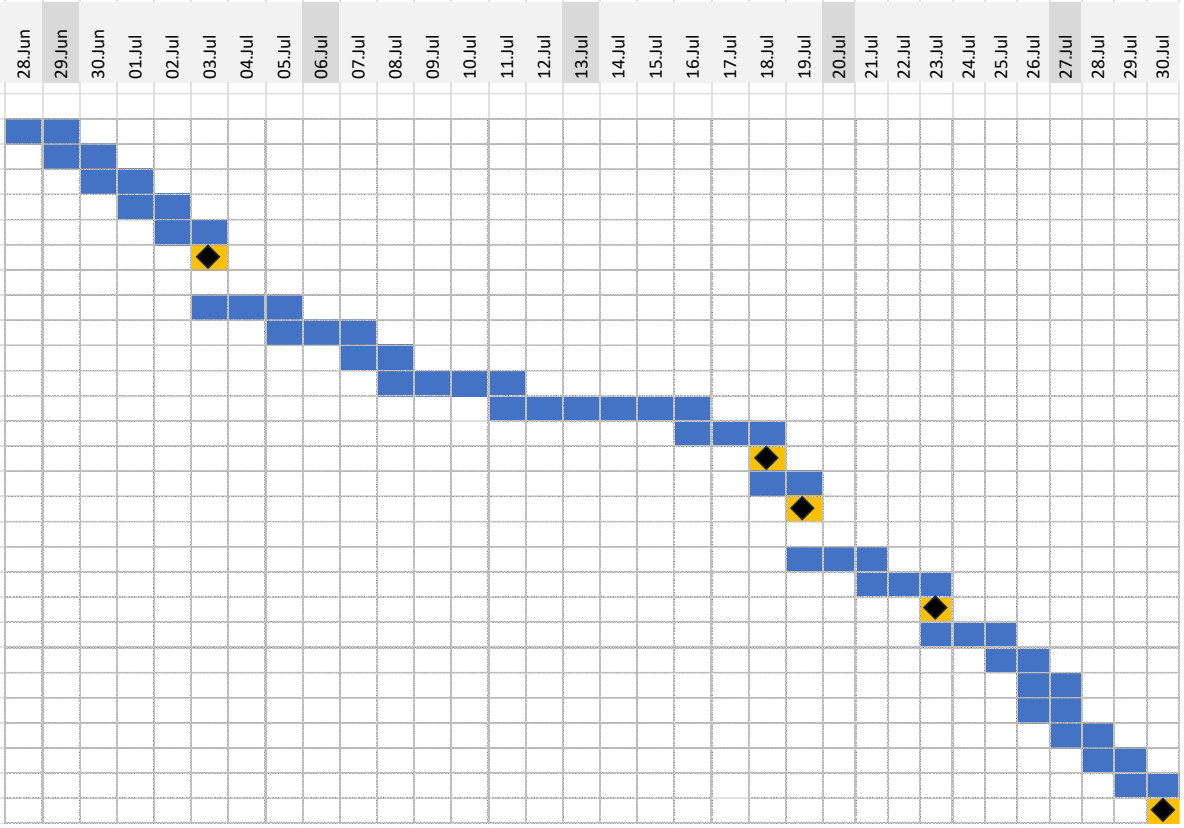


Quelle: Eigene Darstellung mit Microsoft Visio (Microsoft 365, Version 2506)

4. Gantt-Diagramm

Projektplan				
	Projektname	Aufgabenplaner		
	Projektleiter	Kevin Walter		
	Projektstart	28.06.2025		

AP-Nr. & Meilenstein	Bezeichnung	Start	Tage	Ende
	Phase 1: Konzeptionsphase			
1,1	Projektdokumentation erstellen	28.06.2025	1	29.06.2025
1,2	Anforderungsdokumente erstellen	29.06.2025	1	30.06.2025
1,3	Spezifikationsdokumente erstellen	30.06.2025	1	01.07.2025
1,3,1	UML-Diagramme erstellen	01.07.2025	1	02.07.2025
1,4	Kurzzusammenfassung für Pebble	02.07.2025	1	03.07.2025
M1	Konzeptionsphase abgeschlossen	03.07.2025		03.07.2025
	Phase 2: Erarbeitungsphase/Reflexionsphase			
2,1	Architekturdokument erstellen	03.07.2025	2	05.07.2025
2,1,1	MVC-Modell und Entkopplung durch Interface	05.07.2025	2	07.07.2025
2,3	Implementierung	07.07.2025	1	08.07.2025
2,2,1	Logik (TaskManager)	08.07.2025	3	11.07.2025
2,2,2	GUI-Implementierung mit Qt	11.07.2025	5	16.07.2025
2,2,3	Persistenzschicht	16.07.2025	2	18.07.2025
M2	Implementierung abgeschlossen	18.07.2025		18.07.2025
2,7	GitHub-Integration	18.07.2025	1	19.07.2025
M3	Erarbeitungsphase abgeschlossen	19.07.2025		19.07.2025
	Phase 3: Finalisierungsphase			
3,1	Testing (System- und Integrations, Unit-Tests)	19.07.2025	3	21.07.2025
3,1,1	GoogleTest für automatisierte Unit-Tests	21.07.2025	3	23.07.2025
M4	Alle Tests erfolgreich abgeschlossen	23.07.2025		23.07.2025
3,2	Testdokument	23.07.2025	2	25.07.2025
3,2,1	Dokumentation der Testergebnisse	25.07.2025	1	26.07.2025
3,3	Benutzeranleitung	26.07.2025	1	27.07.2025
3,3,1	Beschreibung von Installation/GUI-Elemente	26.07.2025	1	27.07.2025
3,4	Abstract	27.07.2025	1	28.07.2025
3,5	ZIP-Export	28.07.2025	1	29.07.2025
3,6	Finale Abgabe	29.07.2025	1	30.07.2025
M6	Projekt abgeschlossen / Finale abgabe	30.07.2025		30.07.2025



Quelle: Eigene Darstellung mit Microsoft Excel (Microsoft 365, Version 2506)

Phase 1: Konzeptionsphase - Anforderungsdokument

5. 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. Die Anwendung stellt eine digitale Alternative zum analogen Notizblock dar und dient der übersichtlichen Verwaltung anstehender Aufgaben sowie Fälligkeitstermine. 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.

6. Funktionale Anforderungen

Im folgenden Kapitel werden die funktionalen Anforderungen der Anwendung aufgezeigt, außerdem wird in Abbildung 1 ein Use-Case Diagramm dargestellt.

6.1 Funktionsliste

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

6.2 User Stories

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

Als Benutzer möchte ich 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.

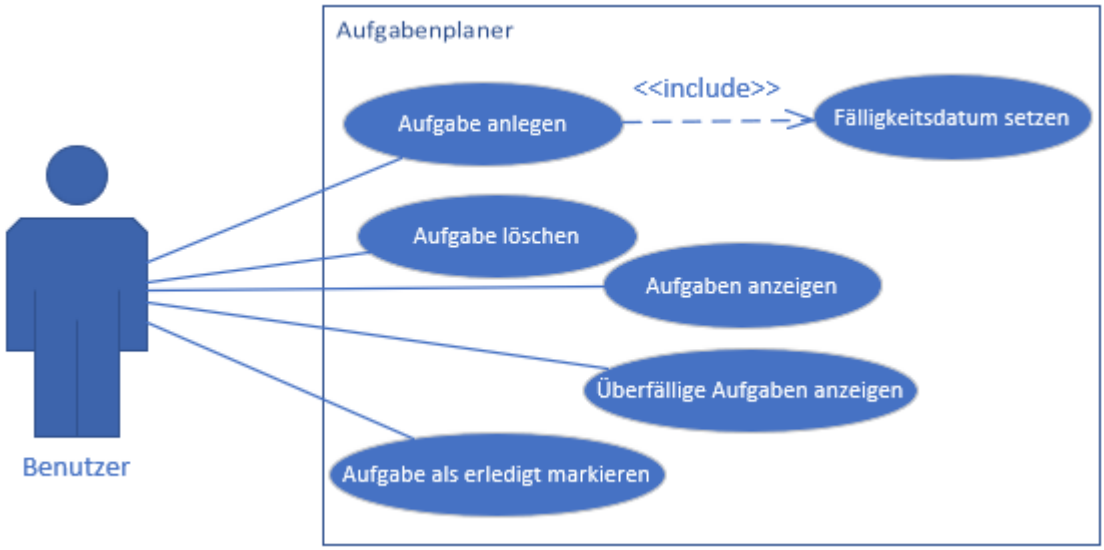
Als Benutzer möchte ich eine Kalender- und Tagesansicht sehen können, damit ich so beispielsweise auch eine ganze Woche planen kann.

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



Quelle: Eigene Darstellung mit Microsoft Visio (Microsoft 365, Version 2506)

7. Nicht-funktionale Anforderungen

Nicht-funktionale Anforderung	Beschreibung und Erfüllbarkeit
Benutzerfreundlichkeit	Die Anwendung besitzt nur ein Hauptfenster und maximal 2 Dialoge. Alle Funktionen sind mit maximal 2 Klicks erreichbar.
Performance	Startzeit < 2 Sekunden auf einem Windows-PC mit SSD. Alle Listen reagieren ohne spürbare Verzögerung.
Datensicherheit	Alle Daten werden lokal auf dem PC abgespeichert. Keine Verbindung zu einem Netzwerk oder Cloud-Service.
Plattform	Die Anwendung läuft unter Windows 10/11 und wurde mit Qt getestet. Kein plattformspezifischer Code.
Eingabevalidierung	Pflichtfeldprüfung (Titel darf nicht leer sein) und Fälligkeitsdatum darf nicht in der Vergangenheit liegen.
Verschlüsselung	Nicht vorgesehen, da die Daten lokal und ohne Internetzugriff gespeichert werden.

8. Konzept zur Qualitätssicherung

Zur Sicherstellung der Softwarequalität werden drei Teststufen geplant:

- Unit-Tests: Um die Kernfunktionen der Geschäftslogiksicherzustellen (z.B. TaskManager-Methoden) werden automatisierte Unit-Tests geschrieben. Diese Tests werden mit Hilfe von googletest (googletest, Version 1.17.0) automatisiert und decken zum Beispiel das Hinzufügen, Löschen und Bearbeiten von Aufgaben ab.
- Integrationstests: Um das Zusammenspiel zwischen den einzelnen Komponenten (z.B. zwischen Controller und View) zu testen, werden Integrationstests geschrieben. Auch die Integrationstests sollen so weit wie möglich automatisiert werden, ist das nicht Möglich werden die Tests durch Screenshots im Kapitel Qualitätssicherung in Phase 3 dokumentiert.
- Systemtests: Die gesamte Anwendung wird unter realen Einsatzbedingungen manuell getestet. Hier werden typische Benutzeraktionen über die grafische Benutzeroberfläche geprüft (z.B. Aufgabenverwaltung, Speichern/Laden). Eine vollständige Automatisierung dieser Tests ist nicht vorgesehen.

Das Ziel der Tests ist es, die Zuverlässigkeit der Anwendung auf jeder Ebene sicherzustellen und Fehler frühzeitig zu erkennen.

9. 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

10. Datenmodell

10.1 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

10.2 MVC-Muster und Stellungnahme zum Einsatz

Das Model-View-Controller (MVC) Muster wird zur Trennung von Präsentation, Steuerung und Datenmodell verwendet. Es wird eingesetzt, wenn unterschiedliche Schichten auf die Daten notwendig sind oder zukünftige Anforderungen an die Darstellung und Interaktion noch nicht feststehen. Das Modell übernimmt die Datenhaltung, die View stellt die Darstellung (GUI) sicher und der Controller vermittelt zwischen den beiden Instanzen, indem er die Benutzereingabe verarbeitet und Veränderungen im Modell anstößt. Diese Trennung erhöht die Wartbarkeit und Erweiterbarkeit der Anwendung (Sommerville, 2012, S. 191–193).

Für die Umsetzung konkret dieser Anwendung wird das MVC-Muster wie in der gängigen Literatur bekannt als Rahmen gewählt. Die drei Schichten View (MainWindow), Model (Task, TaskManager) und Controller (TaskController) werden strikt voneinander getrennt. Zur besseren Entkopplung ist außerdem ein Interface (TaskInterface) vorgesehen. Mit dieser Architektur ist es dann auch leichter möglich, die Anwendung über dieses Projekt hinaus weiterzuentwickeln (z.B. Netzwerkschnittstelle).

10.3 Factory Pattern (TaskFactory)

Das Factory-Pattern ist ein Entwurfsmuster, das die Erzeugung von Objekten kapselt und zentralisiert. Anstatt überall im Code direkt neue Objekte zu erstellen, übernimmt eine spezielle Factory-Klasse (TaskFactory) diese Aufgabe (Balzert, 2009, S. 131).

Die Klasse TaskFactory erzeugt alle Task-Objekte. Die Factory wird sowohl beim Hinzufügen einer neuen Aufgabe durch den Nutzer als auch beim Laden von Aufgaben aus der Datei verwendet. So bleibt die Objekt-Erzeugung an einer Stelle gebündelt und kann bei Änderungen leicht angepasst werden.

10.4 Persistenzschicht

Damit Aufgaben dauerhaft, also auch über die Sitzung hinaus gespeichert werden können, verfügt die Anwendung über eine Persistenzschicht. Die Aufgaben werden lokal in einer Datendatei gespeichert (XML, TXT oder CSV) und beim Start der Anwendung wieder eingelesen (Rau & Schuster, 2021, S. 58). Diese Vorgehen erfüllt die Anforderungen an eine Persistenzschicht, wie sie auch mit

einer Datenbank realisierbar wäre. Die Architektur ermöglicht es, die Persistenzschicht später problemlos durch eine alternative Implementierung wie einer Datenbank zu ersetzen.

10.5 Beans

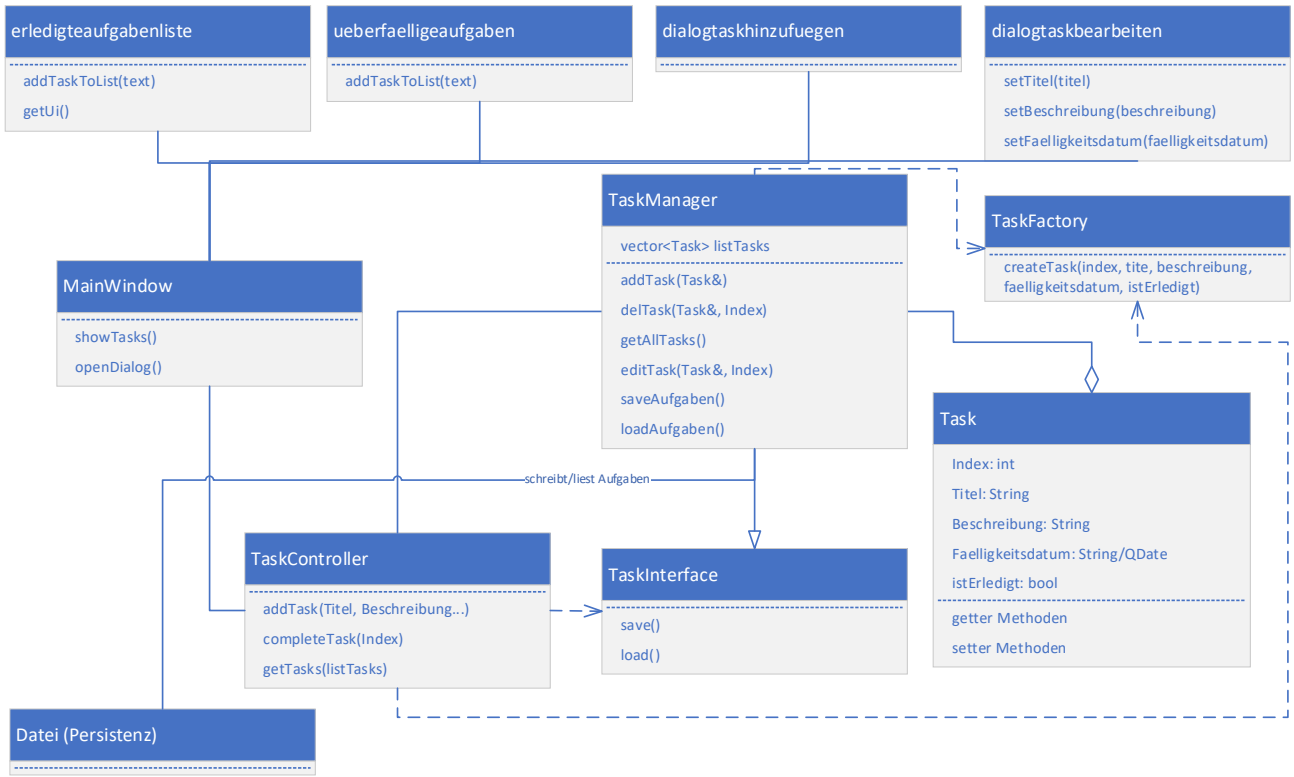
Das JavaBeans-Muster sieht vor, dass zur Objekterzeugung ein parameterloser Konstruktor verwendet wird und die Eigenschaften anschließend über Set-Methoden gesetzt werden (Boch, 2018, S. 12–13). Darüber hinaus ist es üblich, alle Attribute privat zu deklarieren und ausschließlich über öffentliche Getter- und Setter-Methoden darauf zuzugreifen, um die Datenkapselung zu gewährleisten (Silberbauer, 2020, S. 51–52).

Dieses Prinzip wurde in der Klasse Task übernommen, alle Attribute der Klasse sind als private deklariert und der Zugriff erfolgt ausschließlich über öffentliche Getter- und Setter-Methoden. Ein parameterloser Standard-Konstruktor ist vorhanden, sodass Aufgabenobjekte wie beim Bean-Prinzip erstellt und anschließend über Setter individuell befüllt werden können. Damit entspricht die Implementierung von Task dem JavaBeans-Konzept und gewährleistet eine saubere Datenkapselung.

10.6 UML-Klassendiagramm

Die aus dem vorherigen Abschnitt zum MVC-Muster gewonnen Erkenntnisse werden im UML-Klassendiagramm in Abbildung 2 angewendet.

Abbildung 2: UML-Klassendiagramm



Quelle: Eigene Darstellung mit Microsoft Visio (Microsoft 365, Version 2506)

10.7 Nachträgliche Ergänzungen und Änderungen

Wie bereits kurz in Abschnitt 9.2 erwähnt, ist es mit dem MVC-Muster leicht möglich, nachträgliche Ergänzungen und Änderungen einzubauen. Durch das MVC-Muster und die Einführung des Interfaces zwischen Controller und Model ist es möglich, neue Funktionen (z.B. alternative Speicherformate, Erweiterungen in der GUI oder zusätzliche Logik) einzuführen ohne tiefgreifende Änderungen an den bestehenden Komponenten vorzunehmen. Das Interface ermöglicht es außerdem, die Datenhaltung mit minimalen Anpassungsaufwand zu ersetzen. So kann beispielsweise im Nachhinein auf eine Datenbank oder Netzwerkspeicherung umgestiegen werden.

11. 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

Geschäftsprozess 3: Überfällige Aufgaben anzeigen

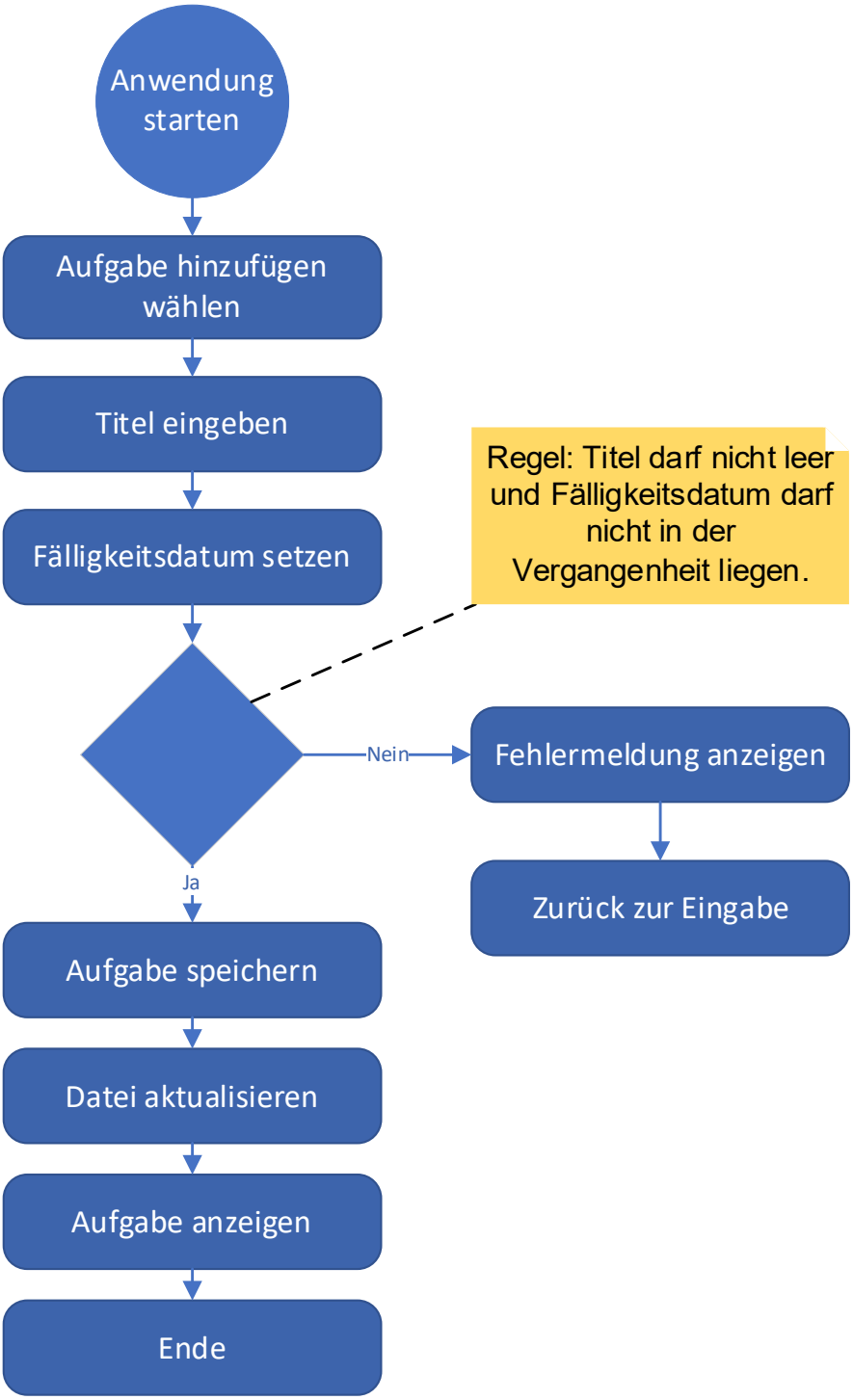
1. Beim Start prüft das Programm alle Aufgaben
2. Aufgaben mit Fälligkeitsdatum < wie aktuelles Datum werden als „überfällig“ markiert/angezeigt

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)

Eine Aktivität beschreibt in UML-Aktivitätsdiagrammen einen Ablauf von Aktionen, die zusammen eine bestimmte Funktionalität oder ein Verhalten implementieren. Dabei kann eine Aktivität sowohl eine einfache Aktion als auch eine komplexe, strukturierte Verhaltensweise darstellen (Balzert, 2009, S. 236). Abbildung 3 zeigt ein UML-Aktivitätsdiagramm für den Kernprozess des Anlegens einer Aufgabe.

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



Quelle: Eigene Darstellung mit Microsoft Visio (Microsoft 365, Version 2506)

12. 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

13. 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.

14. Benutzerschnittstellen

Die Anwendung ist als klassische Desktop-Fensteranwendung unter Windows mit Qt6 realisiert. Sie besteht aus einem Hauptfenster sowie mehreren Dialogfenstern, die jeweils klar abgegrenzte Aufgabenbereiche abbilden.

14.1 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

14.2 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 wird die Aufgabe gelöscht

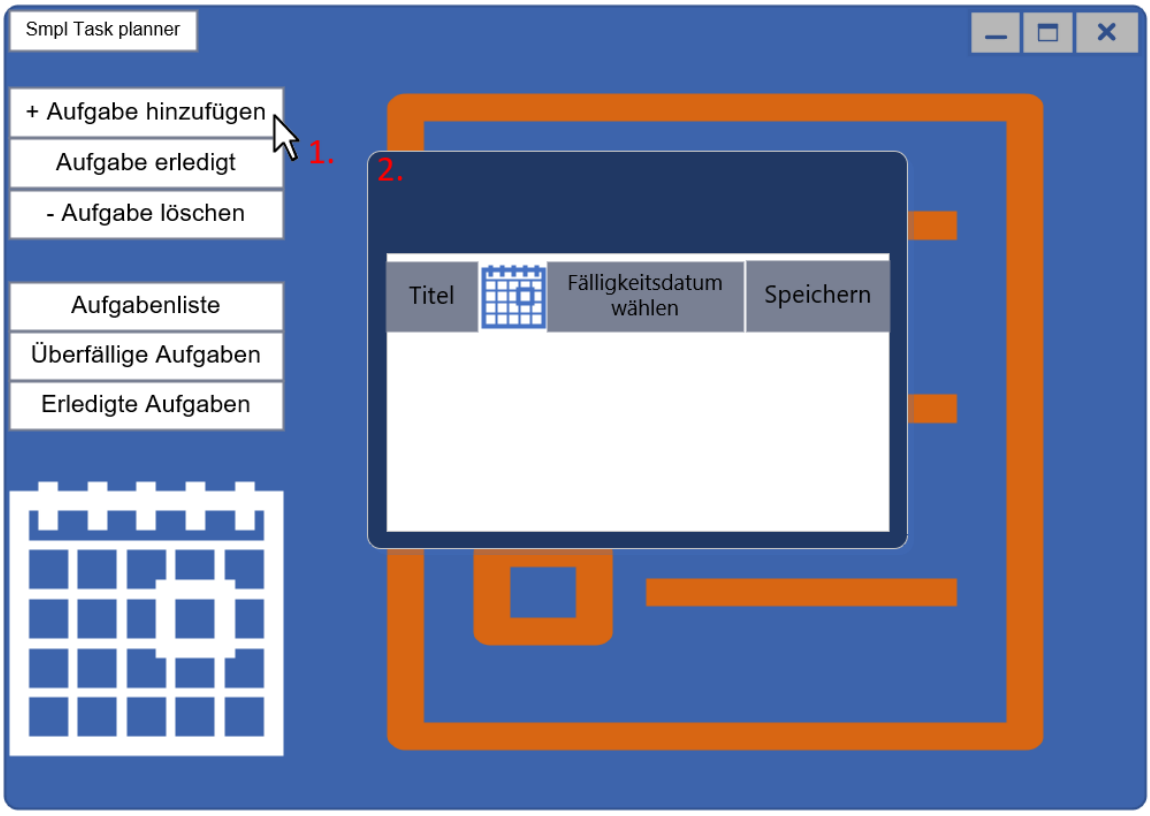
Eingabevalidierung

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

14.3 Skizze der Anwendung

Abbildung 4 zeigt eine grobe Skizze der grafischen Benutzeroberfläche.

Abbildung 4: Skizze Aufgabenplaner



Quelle: Eigene Darstellung mit Microsoft Visio (Microsoft 365, Version 2506)

15. Systemkontext und Datenflüsse

Der Systemkontext beschreibt die relevante Umgebung, die es um ein System herum gibt und die daher in der Entwicklung von Systemen zu beachten ist. Es gibt jedoch auch eine irrelevante Umgebung, die keinen Einfluss auf die Entwicklung hat und mit einer Grauzone an die relevante Umgebung anschließt. Die Interpretation einer Anforderung wird daher durch den Kontext beeinflusst (Balzert, 2009, S. 462), weshalb er ein fester Teil der ersten Phase des Softwareentwurfsprozesses ist (Sommerville, 2012, S. 217).

15.1 Anwendungsspezifischer Kontext

Im Fall dieser Anwendung ist das System vollständig lokal auf dem Windows-Rechner installiert. Es handelt sich um eine Desktop-Fensteranwendung, die weder Cloud-Dienste noch Netzwerkschnittstellen verwendet.

Die wichtigsten Quellen (Dateneingabe) sind:

- Der Benutzer, der über die GUI Aufgaben erstellt, bearbeitet oder löscht
- Eine lokale Datei (TXT, JSON oder CSV), die beim Start der Anwendung automatisch eingelesen wird und dann die Aufgaben in das Programm lädt

Die wichtigsten Senken (Daten-Ausgaben) sind:

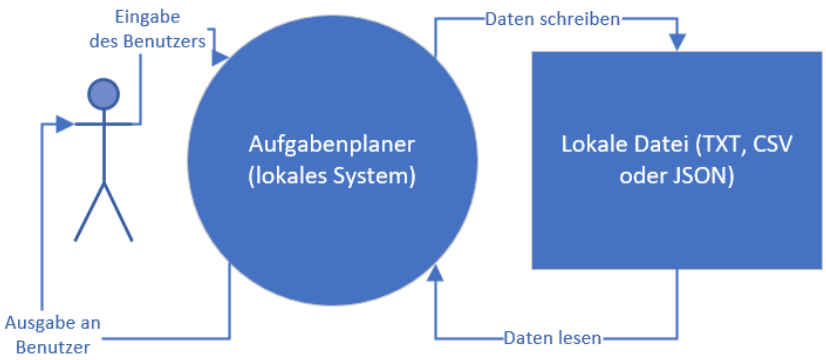
- Die GUI, welche Aufgaben und Statusänderungen visuell darstellt
- Eine lokale Datei, in die alle Änderungen der aktuellen Sitzung gespeichert werden, sobald der Benutzer eine Aufgaben hinzufügt oder bearbeitet

Externe Systeme wie APIs, Server oder Datenbanken sind in diesem Projekt nicht vorhanden.

15.2 Abgrenzung und Kontextdarstellung

Das System interagiert ausschließlich mit dem Betriebssystem (Datei I/O) und dem Benutzer. Es existieren keine weiteren externen Schnittstellen oder Abhängigkeiten in diesem Projekt. Abbildung 5 zeigt die Abgrenzung des Systemkontextes mit Hilfe eines einfachen Kontextdiagrammes.

Abbildung 5: Kontextdiagramm



Quelle: Eigene Darstellung mit Microsoft Visio (Microsoft 365, Version 2506)

Phase 2: Erarbeitungs- und Reflexionsphase – Architekturdokument

16. Technologieübersicht

16.1 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

16.2 Frameworks

Die grafische Oberfläche wurde mit dem Qt6-Framework (Qt 6, Version 17.0.0) als klassische Fensteranwendung für Windows realisiert. Es kommen QMainWindow sowie mehrere QDialog-basierte Komponenten zum Einsatz. Auch das googletest Framework wird für die späteren Tests implementiert.

- Qt6
    - Wird für grafische Oberfläche (GUI) verwendet
    - Ermöglicht plattformübergreifende Entwicklung für Fensteranwendungen
    - Gute Unterstützung für UI-Design, Events und eigene Widgets
    - Integration in C++ Projekt und Build-System 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…)

16.3 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

16.4 Entwicklungswerkzeuge

- Visual Studio Code (Windows, 2021, Version 1.102.3)
- Qt Creator (Qt 6, Version 17.0.0)
- CMake als Buildsystem (CMAKE, Version 4.0.0-rc3)
- GitHub zur Versionskontrolle (GitHub, Version 2.46.0.windows.1)
- windeployqt zum Erstellen von Release-Paketen (Qt For Windows - Deployment | Version 17.0.0)

17. Architekturübersicht

In diesem Projekt wird wie bereits vorher genannt das MVC-Muster (Model-View-Controller) umgesetzt. Es ermöglicht eine saubere Trennung von Benutzerschnittstelle, Anwendungslogik und Datenmodell, wodurch die Software leichter erweiterbar und testbar ist (Ludin, 2014, S. 97). Um eine lose Kopplung zu erreichen, wurde außerdem ein Interface implementiert.

Der Controller (TaskController) hält einen Zeiger auf das Interface (TaskInterface) und ruft Methoden wie addTask(), deleteTask(), loadAufgaben() und saveAufgaben() ausschließlich über das diese Interface auf. Das Model (TaskManager) implementiert das Interface und stellt die Logik für die Aufgabenverwaltung bereit. Die View (mainwindow) kommuniziert ausschließlich mit dem Controller und kennt damit weder die konkrete Implementierung des Models noch die Datenstrukturen.

- **View (Benutzerschnittstelle):**
    - Besteht aus den MainWindow und den Dialogklassen
    - Verantwortlich für Darstellung der Daten und Erfassung von Benutzereingaben
    - Kommuniziert nur mit dem Controller
- **Controller (TaskController):**
    - Vermittelt zwischen View und Model
    - Hält einen Zeiger auf das Interface
    - Ruft Methoden des Models ausschließlich über das Interface auf
- **Interface (TaskInterface):**
    - Definiert abstrakt die vom Model bereitgestellten Methoden
    - Ermöglicht lose Kopplung zwischen Controller und Model
    - Controller nutzt das Interface, Model implementiert es
- **Modell (Datenhaltung & Logik):**
    - Umfasst Task, TaskManager sowie die Dateioperationen (CSV)
    - Implementiert das Interface und enthält die eigentliche Logik
    - Verantwortlich für das Speichern, Laden und Verwalten der Aufgabenobjekte

18. Struktur

Im folgenden Kapitel wird die Struktur der Anwendung aufgezeigt. Abbildung 6 zeigt anhand eines UML-Klassendiagrammes, welche Klassen, Attribute und Methoden in der Anwendung verwendet werden. Um eine bessere Übersichtlichkeit zu gewährleisten, werden nur für das Verständnis relevante Methoden und Attribute aufgezeigt.

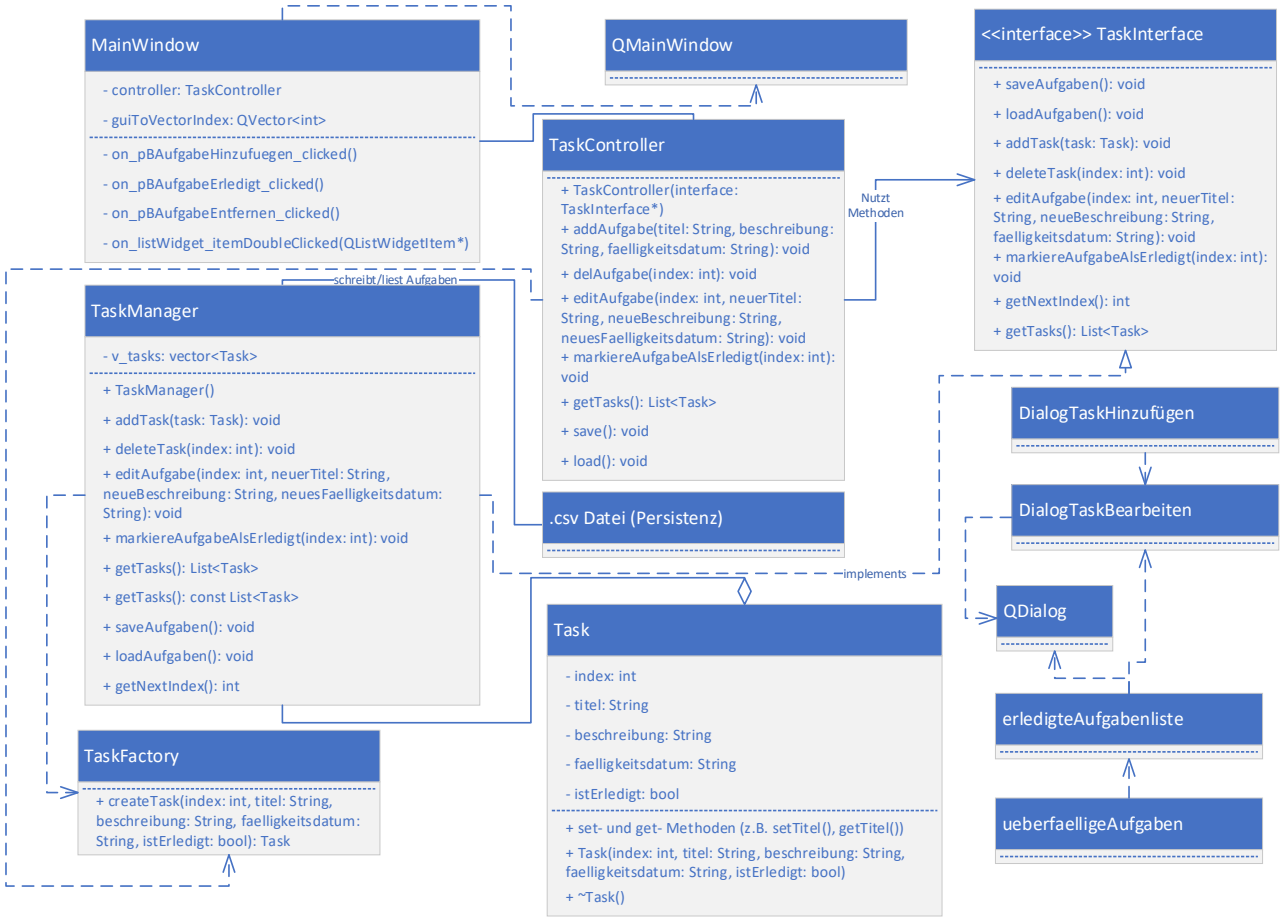
18.1 Hauptkomponenten

- MainWindow (View)
    - Verantwortlich für das Hauptfenster und die Darstellung der Aufgabenliste
    - Nimmt Benutzereingaben entgegen (z.B. Buttons, Doppelklicks)
    - Kommuniziert ausschließlich mit dem Controller
- DialogTaskHinzufuegen / DialogTaskBearbeiten (View)
    - Separate Dialogfenster für das Hinzufügen bzw. Bearbeiten einer Aufgabe
    - Erfassen und Bearbeiten der Aufgabendaten durch den Benutzer
    - Ergebnisse werden an MainWindow (und damit Controller) weitergereicht
- erledigteAufgabenliste / ueberfaelligeAufgaben (View)
    - Dialogfenster zur Anzeige erledigter bzw. überfälliger Aufgaben
- TaskController (Controller)
    - Vermittelt zwischen View und Model über das Interface
    - Nimmt Aufrufe der View entgegen, ruft Methoden am Interface (Model) auf
    - Kapselt Steuerungslogik (z.B. Validierung, Übergabe an Model, Aufruf von Speichern/Laden
    - TaskController nutzt die TaskFactory, um neue Aufgaben anzulegen.
- TaskInterface (Interface)
    - Definiert abstrakt alle Methoden, die das Model bereitstellen muss
    - Für Entkopplung von Controller und Model
- TaskManager (Model)
    - Implementiert das Interface, verwaltet die Sammlung aller Aufgaben
    - Verantwortlich für das Speichern, Laden, Hinzufügen, Löschen, Bearbeiten, Suchen der Aufgaben
    - Arbeitet ausschließlich über das Interface mit dem Controller
    - Die Erzeugung neuer Task-Objekte erfolgt zentral über die Klasse TaskFactory (nach Factory-Pattern)
- Task
    - Datenklasse für einzelne Aufgaben (Attribute: Titel, Beschreibung, Fälligkeitsdatum, Status, Index)
    - Methoden zum Ändern und Auslesen der Aufgabendaten

18.2 Abhängigkeiten

- TaskController und TaskManager verwenden die TaskFactory zur Erzeugung von Task-Objekten (Factory-Pattern)
- MainWindow und die Dialoge kommunizieren ausschließlich mit dem TaskController
- TaskController hält einen Zeiger auf das Interface (TaskInterface) und ruft Methoden wie addTask(), deleteTask(), loadAufgaben(), saveAufgaben() usw. ausschließlich über dieses Interface auf
- TaskManager implementiert das Interface und stellt die Logik zur Aufgabenverwaltung bereit
- TaskManager verwaltet eine Sammlung von Task-Objekten
- Dialogfenster übergeben ihre Eingaben über MainWindow an den Controller
- Die View kennt niemals die konkrete Implementierung des Models, sondern nutzt immer den Controller

Abbildung 6: UML-Klassendiagramm



Quelle: Eigene Darstellung mit Microsoft Visio (Microsoft 365, Version 2506)

18.3 Erweiterbarkeit

Die lose Kopplung durch das Interface ermöglicht es, das Model auszutauschen (z. B. durch eine neue Klasse für Datenbank- oder Netzwerkspeicherung), ohne dass Änderungen an dem Controller oder der View notwendig sind. Neue Features wie alternative Speicherformate, Synchronisierung

oder weitere Darstellungsformen können einfach durch neue Implementierungen des Interfaces ergänzt werden. Das MVC-Muster vereinfacht das Testen einzelner Komponenten, da View, Controller und Model isoliert getestet werden können.

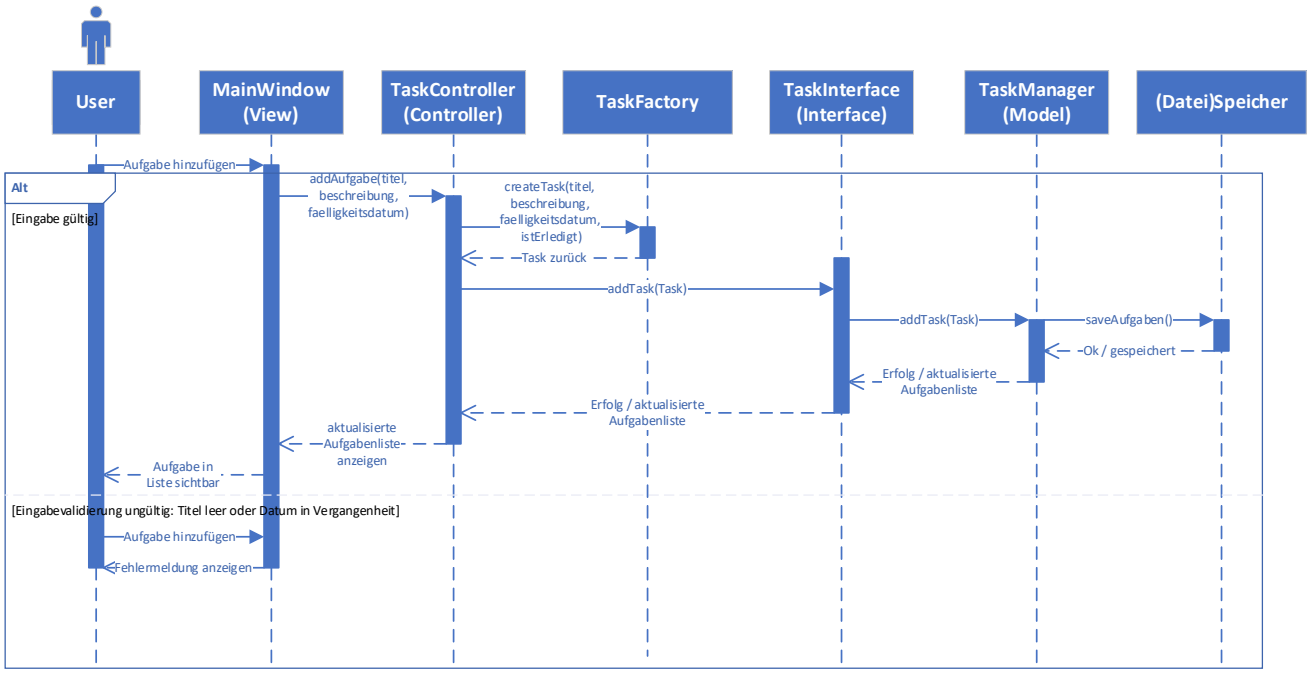
19. Verhalten

In der UML werden Sequenzdiagramme eingesetzt, um die Kommunikation und den Ablauf zwischen verschiedenen Objekten bzw. Komponenten einer Anwendung darzustellen. Sequenzdiagramme können mit Anwendungsfällen verknüpft werden und bieten eine Sicht auf den Ablauf der Methodenaufrufe und Antworten zwischen den beteiligten Instanzen (Sommerville, 2012, S. 162). Die Objekte und Akteure werden als Rechtecke mit Lebenslinien visualisiert, die Interaktionen als gerichtete Kanten (Pfeile) für Methodenaufrufe und Rückgaben (Balzert, 2009, S. 333–334). Zudem wird im Diagramm durch einen ALT-Block auch der Fehlerfall (z. B. ungültige Eingaben) dargestellt, wie es in Sommerville (2012, S. 162) empfohlen wird. Die Rückmeldung an den Benutzer erfolgt dann direkt über die grafische Oberfläche, ohne weitere Systeminteraktionen.

Das folgende Sequenzdiagramm in Abbildung 7 zeigt den Ablauf des Use Cases „Aufgabe anlegen“ (sh. Kapitel 11 – Geschäftsprozess 1). Im Diagramm ist zu erkennen, dass die Benutzeraktion zunächst an die View weitergegeben wird, die wiederum den Controller aufruft. Über das Interface wird das Model (TaskManager) angesprochen, was die Daten an die Persistenzschicht weiterleitet. Nach Abschluss der Operation werden Rückgabepfade von TaskManager über das Interface zum Controller und von dort zur View sowie als Rückmeldung an den Benutzer dargestellt.

Die Darstellung als Sequenzfluss wurde gewählt, um den logischen Ablauf und die zeitliche Reihenfolge der wichtigsten Interaktionen im Use Case „Aufgabe anlegen“ klar und übersichtlich zu veranschaulichen. Damit wird nachvollziehbar, wie die Benutzeraktion Schritt für Schritt durch alle beteiligten Komponenten des Systems weitergegeben und verarbeitet wird.

Abbildung 7: UML-Sequenzdiagramm: Aufgabe anlegen/hinzufügen



Quelle: Eigene Darstellung mit Microsoft Visio (Microsoft 365, Version 2506)

Quellenverzeichnis

Balzert, H. (2009). Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering. In Spektrum Akademischer Verlag eBooks. https://doi.org/10.1007/978-3-8274-2247-7

CMAKE - Upgrade your software build system. (o. D.). Version 4.0.0-rc3 https://cmake.org/ Lizenz: BSD 3-Clause License (https://cmake.org/licensing/)

Download Visual Studio Code - Mac, Linux, Windows. (2021, 3. November). Version 1.102.3 https://code.visualstudio.com/download Lizenz: MIT License (https://code.visualstudio.com/license)

GitHub Build and ship software on a single, collaborative platform. (2025). Version 2.46.0.windows.1 https://github.com/ Lizenz: GitHub Desktop/Git: GPLv2 (Git), MIT License (GitHub Desktop)

google. (o. D.). Version 1.17.0 GitHub - google/googletest: GoogleTest - Google Testing and Mocking Framework. GitHub. https://github.com/google/googletest Lizenz: BSD-3-Clause (https://github.com/google/googletest/blob/main/LICENSE)

Ludin, A. (2014). Learn BlackBerry 10 App Development: a Cascades-Driven approach. https://directory.doabooks.org/handle/20.500.12854/51477

Microsoft Excel: Kostenlose Online-Tabellenkalkulationssoftware | Microsoft 365. (o. D.). Version 2506 https://www.microsoft.com/de-de/microsoft-365/excel?market=de Lizenz: Proprietär, kommerziell

Microsoft Visio: Diagramming & Flowcharts | Microsoft 365. (o. D.). Version 2506 https://www.microsoft.com/en-us/microsoft-365/visio/flowchart-software Lizenz: Proprietär, kommerziell

Qt 6 - The latest version of Qt. (o. D.). Version 17.0.0 https://www.qt.io/product/qt6 Lizenz: GPL/LGPL (freie Version) oder kommerziell erhältlich (siehe https://www.qt.io/licensing/)

Rau, K. & Schuster, T. (2021). Agile objektorientierte Software-Entwicklung: Schritt für Schritt vom Geschäftsprozess zum Java-Programm. Springer Vieweg.

Sommerville, I. (2012). Software Engineering. ISBN: 978-3-86894-099-2

Windeployqt Qt for Windows - Deployment | Qt 6.9. (o. D.). Version 17.0.0 https://doc.qt.io/qt-6/windows-deployment.html Lizenz: Lizenz: GPL/LGPL oder kommerziell

Boch, J. (2018). *Effective java: Best Practices für die Java-Plattform*.

Silberbauer, C. (2020). *Einstieg in Java und OOP: Grundelemente, Objektorientierung, Design-Patterns und Aspektorientierung*. Springer Vieweg.