Portfolio – Phase 2

IU Internationale Hochschule Fernstudium

Studiengang: Master of Science Informatik

Kurs: Projekt – Software Engineering (DLMCSPSE01_D)

Projektdokumentation

Windows Anwendung:
Priorisierungsmatrix nach dem Sung-Diagramm

Gregor Hubmann

Matrikelnummer: 9196398

Hornstraße 185

5423 St. Koloman

Abgabedatum (Phase 2): 24.11.2024

Änderungen nach Phase 1:

Kapitel 4-8 hinzugefügt.

Inhaltsverzeichnis

Abbildungsverzeichnis

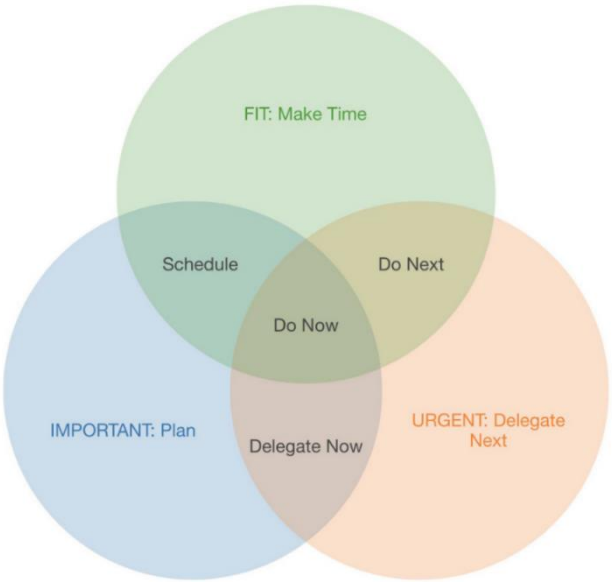
Tabellenverzeichnis

1. Projektübersicht

Dieses Projekt zielt darauf ab, eine Anwendung zu entwickeln, die Nutzer dabei unterstützt, ihre Aufgaben effektiv zu priorisieren und die Produktivität zu steigern. Die Anwendung basiert auf dem Sung-Diagramm, das eine detailliertere Betrachtung von Dringlichkeit (Urgent), Wichtigkeit (Important) und Fähigkeiten zur Erledigung (Fitness) von Aufgaben ermöglicht.

Die Anwendung soll es den Nutzern ermöglichen, ihre Aufgaben in verschiedene Prioritätskategorien einzuteilen, um so eine strukturierte und übersichtliche Planung zu ermöglichen. Die Benutzeroberfläche der Anwendung wird eine intuitive Drag-and-Drop-Funktion bieten, um Aufgaben in die sieben Felder des Sung-Diagramms einzuordnen. Zudem wird die Anwendung zusätzliche Funktionen bieten, wie etwa die Möglichkeit, Standardprioritäten festzulegen, Benachrichtigungen für fällige Aufgaben zu erhalten sowie erledigte Aufgaben zu archivieren. Dies ermöglicht den Nutzern eine umfassende Unterstützung bei der Entscheidungsfindung und trägt zur besseren Verwaltung und Priorisierung der Aufgaben bei.

Abb. 1 – Sung-Diagramm



Quelle: übernommen aus
Bratterud et al., 2020, S. 499

Das Ziel ist es, eine benutzerfreundliche und funktionsreiche Anwendung zu schaffen, die Nutzern hilft, ihre Aufgaben klar zu strukturieren und den Fokus auf die wirklich wichtigen und selbstdurchzuführenden Aktivitäten zu legen.

Das Endprodukt soll als native Desktop-Anwendung unter Windows 10/11 lauffähig sein und eine klare und einfach zu bedienende grafische Benutzeroberfläche bieten.

Bei der Konzeption der Anwendung werden verschiedene Best Practices des Software-Engineerings berücksichtigt, um eine hohe Datenqualität, Benutzerfreundlichkeit und Wartbarkeit zu gewährleisten. Dazu zählen die Trennung von Geschäftslogik und Datenstruktur, die Verwendung eindeutiger Attribute, die klare Trennung von Pflicht- und optionalen Feldern sowie die Definition von Geschäftsregeln zur Sicherstellung der Datenkonsistenz.

Um Konsistenz zwischen Code, Kommentaren und Dokumentation sicherzustellen, werden bereits jetzt teilweise englische Begriffe, in den verschiedenen Dokumenten, verwendet. Dies steht im Einklang mit den Best Practices der Python-Programmierung gemäß PEP8[1], wonach Kommentare und Code in englischer Sprache verfasst werden, um internationale Verständlichkeit und Austauschbarkeit zu gewährleisten.

[1] Python Enhancement Proposal 8 (https://peps.python.org/pep-0008/)

2. Risikomanagement

Ein Risiko wird definiert als ein Ereignis, das mit einer bestimmten Wahrscheinlichkeit auftreten und dem Projekterfolg erheblichen Schaden zufügen kann (Schatten et al., 2010, S. 105). Typisch sind u. a. der Ausfall von Teammitgliedern, Einsatz unbekannter Technologien oder unrealistische Zeitpläne. Im Folgenden werden die wichtigsten Risiken für das Projekt sowie entsprechende Gegenmaßnahmen dargestellt, um mögliche negative Auswirkungen zu reduzieren.

Normalerweise würden die Verantwortlichkeiten auf mehrere zielführende Personen aufgeteilt werden. Da es sich hier um ein Ein-Mann-Projekt handelt, ist der Projektleiter (auch als Entwickler, Tester, …) für alle Risiken verantwortlich.

Tab. 1 - Risikomatrix

Risiko	**Eintrittswahrscheinlichkeit**	**Schwere der Auswirkung**	**Auswirkungen**	**Gegenmaßnahmen**	**Verantwortlicher**	**Status:** **M**onitoring **E**ingetreten **V**erhindert
Verzögerungen im Zeitplan	Mittel	Mittel	Verzögerung der gesamten Entwicklung	Realistische Zeitplanung, regelmäßige Überprüfung des Fortschritts	Projektleiter	M
Technische Herausforderungen	Hoch	Mittel	Verzögerungen, unvollständige Funktionen	Einplanung von Pufferzeiten, Nutzung externer Expertise bei Bedarf	Projektleiter	M
Unklare Anforderungen	Mittel	Hoch	Falsche Implementierung, Nacharbeit	Regelmäßige Abstimmung mit Stakeholdern, frühzeitige Klärung der Anforderungen	Projektleiter	M
Mangelnde Benutzerakzeptanz	Mittel	Hoch	Geringe Nutzung der App	Einbindung von potenziellen Nutzern in die Testphase, Usability-Tests	Projektleiter	M
Fehlerhafte Implementierung	Niedrig	Hoch	Bugs, die die Nutzung einschränken	Umfassende Testphase (Unit-, Integrationstests), Code-Reviews	Projektleiter	M

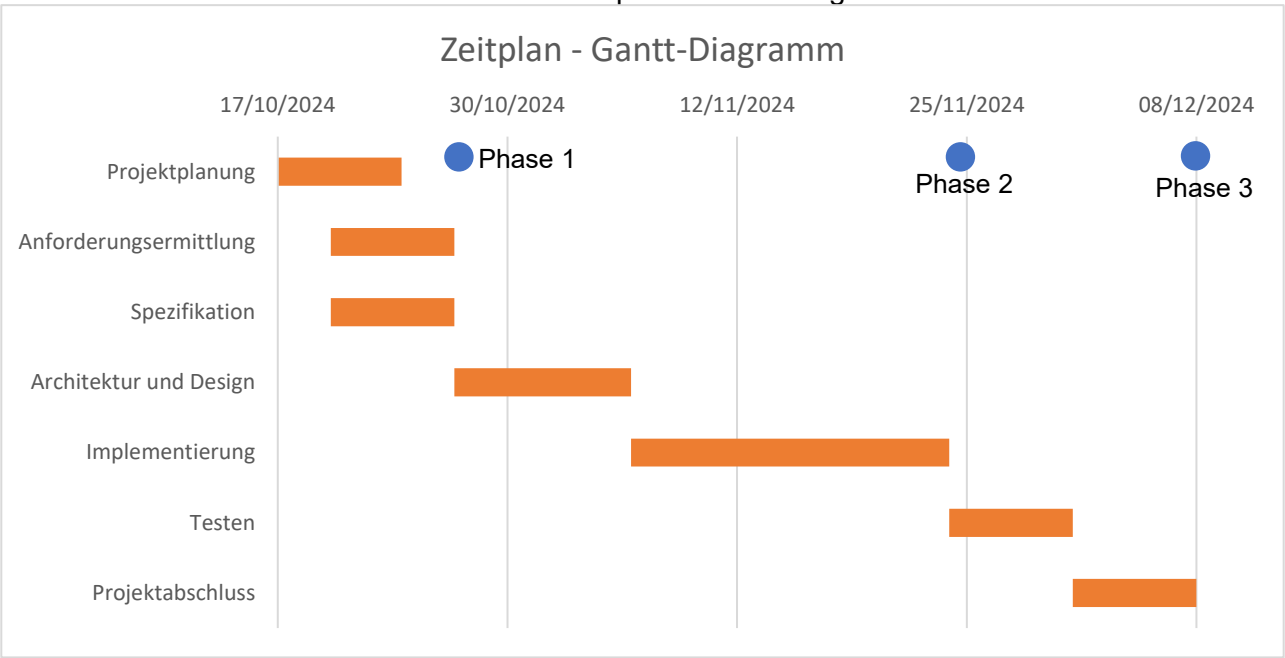
Quelle: eigene Darstellung

3. Zeitplanung

Für die Entwicklung der Anwendung wurde das Wasserfallmodell als Vorgehensmodell gewählt. Aufgrund der überschaubaren Projektgröße, der kurzen Projektzeit und der Anzahl der Mitarbeitenden (eine Person in der Funktion als Projektleiter und Projektmitarbeiter sowie eine Person als echter weiterer Stakeholder in der Rolle des Betreuers) eignet sich das Wasserfallmodell besonders gut. Es bietet eine klare Struktur und ermöglicht eine sequenzielle Vorgehensweise, die für das Projekt gut umsetzbar ist.

Die Projektplanung wurde durch ein Gantt-Diagramm visualisiert, welches die verschiedenen Phasen des Projekts darstellt. Diese Phasen umfassen die Projektplanung, Anforderungsermittlung, Spezifikation, Architektur und Design, Implementierung, Testen sowie den Projektabschluss. Jede Phase ist zeitlich definiert, um sicherzustellen, dass die Arbeit innerhalb des geplanten Zeitrahmens bis Mitte Dezember abgeschlossen wird.

Abb. 2 – Zeitplan – Gantt-Diagramm



Quelle: eigene Darstellung

Meilenstein:

- Abgabe Phase 1: 27.10.2024
- Abgabe Phase 2: 24.11.2024
- Abgabe Phase 3: 08.12.2024

4. Versionskontrolle

Das Projekt wird in einem GitHub-Repository verwaltet, das den gesamten Quellcode sowie alle relevanten Artefakte enthält.

Das Repository kann unter folgendem Link erreicht werden: https://github.com/craexxn/PrioritizationSung

Die Nutzung von GitHub ermöglicht eine effiziente Versionskontrolle und eine klare Nachverfolgbarkeit aller Änderungen im Projektverlauf.

5. Qualitätssicherung und Teststrategie

Im Rahmen des Projekts, das auf dem Wasserfallmodell basiert, wird Wert auf eine strukturierte und gründliche Qualitätssicherung gelegt. Da das Wasserfallmodell eine klare Trennung und Abfolge von Entwicklungsphasen vorsieht, werden die Tests hauptsächlich in der Finalisierungsphase systematisch durchgeführt.

Die Teststrategie umfasst dabei verschiedene Teststufen:

- Unit-Tests: Während der Implementierung werden die Hauptkomponenten durch initiale Unit-Tests geprüft. Diese Tests decken die grundlegenden Funktionalitäten der Klassen und Methoden ab und stellen sicher, dass die einzelnen Bausteine der Anwendung stabil und korrekt arbeiten.
- Integrationstests: Nach Abschluss der einzelnen Komponenten folgen Integrationstests, um die Zusammenarbeit der verschiedenen Module und Klassen sicherzustellen. Diese Tests prüfen die Interaktionen zwischen den Modulen und helfen, Fehler an Schnittstellen oder Inkompatibilitäten zwischen Komponenten frühzeitig zu identifizieren.
- Systemtests: In der finalen Testphase wird das gesamte System getestet, um die Anwendung in ihrer Gesamtheit zu validieren und die Einhaltung der funktionalen und nicht-funktionalen Anforderungen zu überprüfen. Hierbei wird die Anwendung unter realitätsnahen Bedingungen getestet.

Zusätzlich wird im Rahmen der Qualitätssicherung ein umfassendes Testdokument erstellt, das die gesamte Teststrategie dokumentiert und alle durchgeführten Testfälle protokolliert. Das Testdokument umfasst:

- Teststrategie: Eine Beschreibung des Vorgehens bei der Qualitätssicherung und wie die einzelnen Tests und Teststufen geplant und durchgeführt wurden.
- Testprotokoll: Ein strukturiertes Protokoll der durchgeführten Tests, einschließlich eindeutiger Testfall-IDs, Kurzbeschreibung jedes Testfalls, Vorbedingungen, Eingabedaten oder auszuführenden Aktionen, erwarteten Ergebnissen und tatsächlichen Ergebnissen.

6. Projektdokumentation

Im Rahmen des Projekts wurde eine umfassende Projektdokumentation erstellt, die alle wesentlichen Schritte und Entscheidungen entlang des Projektverlaufs nachvollziehbar beschreibt und die Basis für die Implementierung und Qualitätssicherung der Anwendung bildet. Die Dokumentation umfasst die folgenden Kernbestandteile:

- Anforderungsdokument
  Das Anforderungsdokument beschreibt die funktionalen und nicht-funktionalen Anforderungen an die Anwendung, die in der Konzeptionsphase gesammelt und analysiert wurden. Es stellt sicher, dass die Erwartungen an die Funktionalität und Benutzerfreundlichkeit klar definiert sind und dient als grundlegender Leitfaden für die Umsetzung und Validierung der Lösung. Die Anforderungen wurden priorisiert und in einzelne, überprüfbare Komponenten aufgeteilt, die die Basis für die Entwicklungsschritte bilden. Das Anforderungsdokument wurde nach Phase 1, aus Gründen der Nachvollziehbarkeit, nicht mehr verändert.

- Spezifikationsdokument
  Auf Grundlage der definierten Anforderungen wurden im Spezifikationsdokument detaillierte Spezifikationen zur Funktionalität und zum Verhalten der Anwendung erstellt. Es beschreibt, wie die Anforderungen in konkrete Funktionen und Interaktionen umgesetzt werden sollen. Hier sind die detaillierten Beschreibungen von Benutzerinteraktionen, Geschäftslogik und Prozessen enthalten. Das Spezifikationsdokument stellt sicher, dass die Anwendung konsistent und gemäß den definierten Anforderungen entwickelt wird und dass alle Funktionen umfassend spezifiziert sind. Das Spezifikationsdokument wurde nach Phase 1, aus Gründen der Nachvollziehbarkeit, nicht mehr verändert.

- Architekturdokument
  Das Architekturdokument beschreibt die technische Struktur und die Architektur der Anwendung. Es enthält die Auswahl und Begründung der verwendeten Technologien und Werkzeuge, eine Übersicht über die Softwarearchitektur sowie Klassendiagramme und Sequenzdiagramme, die die Struktur und das Verhalten der wichtigsten Komponenten visualisieren. Das Architekturdokument gewährleistet, dass die Implementierung auf einer soliden, skalierbaren und wartbaren Architektur basiert und alle Komponenten der Anwendung nahtlos zusammenarbeiten. Das Architekturdokument wurde in Phase 2 neu erstellt.

- Testdokument (*Anmerkung: in Phase 2 noch nicht erstellt*)
  Das Testdokument bildet die Grundlage für die Qualitätssicherung des Projekts und dokumentiert die Teststrategie sowie die Durchführung der einzelnen Teststufen – Unit-Tests, Integrationstests und Systemtests. Für jeden Testfall enthält das Testdokument eine Protokollierung, die alle notwendigen Informationen wie die eindeutige Testfall-ID, Vorbedingungen, Eingabedaten, erwartete und tatsächliche Ergebnisse umfasst. Dieses strukturierte

Vorgehen stellt sicher, dass die Qualität und Funktionsfähigkeit der Anwendung umfassend überprüft und die festgelegten Anforderungen vollständig erfüllt werden. Das Testdokument soll in Phase 3 neu erstellt werden.

- Projektdokumentation
  Zusätzlich zu den zentralen Dokumenten bietet die Projektdokumentation eine abschließende, zusammenführende Darstellung aller Arbeitsschritte und Ergebnisse. Sie verbindet die Anforderungen, Spezifikationen, Architekturentscheidungen und Testprozesse zu einem kohärenten Gesamtbild des Projekts und gewährleistet eine transparente Nachvollziehbarkeit des gesamten Entwicklungsverlaufs. Die Projektdokumentation fasst die Kernpunkte der vorherigen Dokumente prägnant zusammen und dient als zentrale Anlaufstelle für eine umfassende Übersicht über die Anwendung. Die Projektdokumentation wird in allen Phasen aktualisiert.

7. Abweichungen vom Anforderungs- und Spezifikationsdokument

Die Umsetzung des Projekts erfolgte weitgehend im Einklang mit dem ursprünglichen Anforderungs- und Spezifikationsdokument. Es kam lediglich zu wenigen, eher geringfügigen Änderungen, die auf praktische Überlegungen sowie die Optimierung der Anwendung zurückzuführen sind. Diese Anpassungen tragen dazu bei, die Effizienz, Benutzerfreundlichkeit und Wartbarkeit der Software zu verbessern, ohne die Kernanforderungen zu beeinträchtigen.

Push-Benachrichtigungen:
Anstatt der im Anforderungs- und Spezifikationsdokument vorgesehenen Push-Benachrichtigungen über HTTP/HTTPS wurde ein integrierter Benachrichtigungsdienst implementiert. Der Grund für diese Entscheidung lag in der Einfachheit und der lokalen Nutzung der Anwendung. Die Nutzung eines integrierten Benachrichtigungsdienstes reduziert die Abhängigkeit und bietet gleichzeitig eine schnellere Reaktionszeit. Der Benachrichtigungsdienst prüft direkt in der Anwendung alle Fälligkeitsdaten und generiert lokale Hinweise für den Nutzer. Diese lokale Implementierung ist nicht nur ressourcenschonender, sondern auch benutzerfreundlicher, da keine zusätzlichen Konfigurationen oder Schnittstellen erforderlich sind.

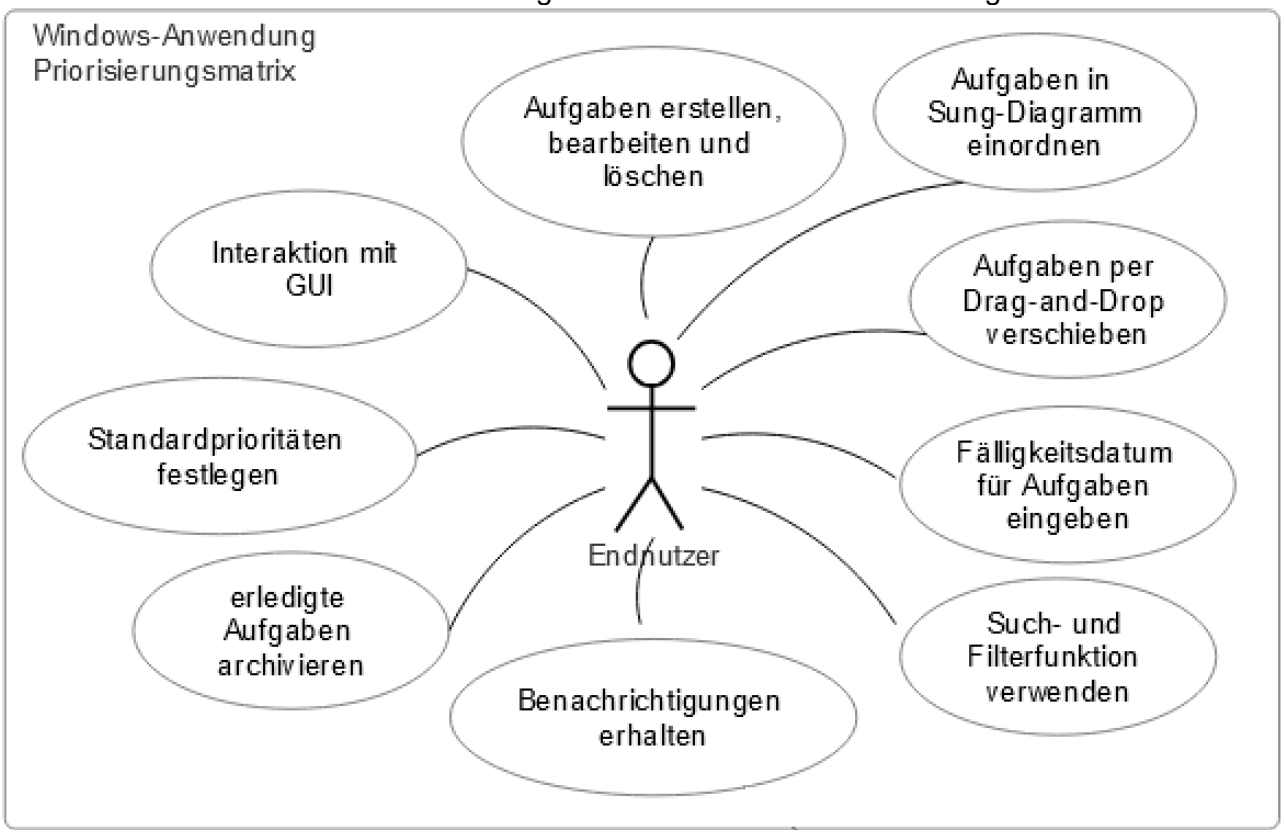
Klassen-Management:
Die Klasse "Category", wie sie im Anforderungs- und Spezifikationsdokument beschrieben wurde, wurde nicht als eigenständige Komponente implementiert. Stattdessen wurden die Kategorien direkt in die Klasse "Task" integriert. Diese Entscheidung wurde getroffen, um die Datenstruktur schlanker und weniger komplex zu gestalten. Jede Aufgabe trägt nun ihre Prioritätsinformationen (Importance, Urgency, Fitness) direkt als Attribute. Damit entfiel die Notwendigkeit einer separaten Kategorie-Klasse, da diese Priorisierungsinformationen ausreichend sind, um die Aufgaben korrekt im Sung-

Diagramm einzuordnen. Dies erleichtert sowohl die Speicherung als auch die Bearbeitung von Aufgaben und macht die Anwendung effizienter und weniger fehleranfällig.

Schichtenarchitektur:
Der GUI-Controller, als zentrale Vermittlungsschicht zwischen Benutzeroberfläche und Geschäftslogik, weicht in seiner Implementierung aufgrund von Unerfahrenheit teilweise von der vorgesehenen Schichtenarchitektur ab. Zwischenzeitlich wurde Geschäftslogik direkt im GUI-Controller implementiert, was eine klare Trennung der Verantwortlichkeiten beeinträchtigt. In der finalen Phase des Projekts soll dieser Umstand aufgegriffen und in der Projektdokumentation detailliert aufgearbeitet werden. Dabei werden konkrete Optimierungsvorschläge für zukünftige Iterationen festgehalten, um eine konsequente Einhaltung der Schichtenarchitektur sicherzustellen.

Aktualisiertes Use-Case-Diagramm:
Das folgende aktualisierte Use-Case-Diagramm (vgl. Anforderungsdokument, S. 10) visualisiert die funktionalen Anforderungen. Es zeigt die Interaktionen der Endnutzer mit den verschiedenen Funktionen der Anwendung und stellt die zentralen Anwendungsfälle dar, die für die Umsetzung der Anforderungen relevant sind.

Abb. 3 – aktualisiertes Use-Case-Diagramm zu funktionalen Anforderungen und Endnutzer



Quelle: eigene Darstellung

8. Status der Implementierung zum Ender der Phase 2

Die Implementierung wurde einen Tag vor dem Ender der Phase 2 (Ende Phase 2: 24.11.2024) eingefroren, um eine fristgerechte Fertigstellung der restlichen Projektarbeit und Dokumentation zu gewährleisten. Alle grundlegenden Anforderungen wurden erfolgreich umgesetzt, wobei Verbesserungen und Verfeinerungen – wie bei jedem Projekt – immer möglich sind. Diese werden, soweit zeitlich realisierbar, in der Finalisierungsphase aufgegriffen bzw. nur mehr dokumentiert. Die verfügbare Zeit stellte eine begrenzende Ressource dar, weshalb der Fokus auf eine stabile Umsetzung und eine saubere Dokumentation gelegt wird, um die Projektziele zu erreichen.

Literaturverzeichnis

Bratterud, H., Burgess, M., Fasy, B. T., Millman, D. L., Oster, T., & Sung, E. (Christine). (2020). The Sung Diagram: Revitalizing the Eisenhower Matrix. In A.-V. Pietarinen, P. Chapman, L. Bosveld-de Smet, V. Giardino, J. Corter, & S. Linker (Hrsg.), *Diagrammatic Representation and Inference* (S. 498–502). Springer International Publishing. https://doi.org/10.1007/978-3-030-54249-8_43

Schatten, A., Biffl, S., Demolsky, M., Gostischa-Franta, E., Östreicher, T., & Winkler, D. (2010). *Best Practice Software-Engineering: Eine Praxiserprobte Zusammenstellung Von Komponenten-orientierten Konzepten, Methoden und Werkzeugen*. Spektrum Akademischer Verlag GmbH. http://ebookcentral.proquest.com/lib/badhonnef/detail.action?docID=511284

Portfolio – Phase 2

IU Internationale Hochschule Fernstudium

Studiengang: Master of Science Informatik

Kurs: Projekt – Software Engineering (DLMCSPSE01_D)

Anforderungsdokument

Windows Anwendung:
Priorisierungsmatrix nach dem Sung-Diagramm

Gregor Hubmann

Matrikelnummer: 9196398

Hornstraße 185

5423 St. Koloman

Abgabedatum (Phase 2): 24.11.2024

Änderungen nach Phase 1:
Keine Änderungen vorgenommen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Das gegenständliche Anforderungsdokument beschreibt die funktionalen und nicht-funktionalen Anforderungen für die Entwicklung der Anwendung. Das Ziel ist es, eine klare und strukturierte Übersicht der Anforderungen zu geben, die sicherstellen, dass alle Stakeholder ein einheitliches Verständnis der zu entwickelnden Anwendung haben.

Das Dokument beginnt mit der Identifikation der Stakeholder, um die relevanten Interessensgruppen festzulegen. Anschließend werden die funktionalen Anforderungen der Anwendung durch User Stories beschrieben. Die nicht-funktionalen Anforderungen umfassen unter anderem die Benutzerfreundlichkeit und die Performanz der Anwendung. Ein Glossar am Ende des Dokuments soll sicherstellen, dass alle projektbezogenen Begriffe eindeutig definiert sind.

1. Stakeholder

Die Identifikation der Stakeholder ist ein entscheidender Schritt im Anforderungsmanagement, da die Bedürfnisse und Erwartungen der Stakeholder maßgeblich den Erfolg des Projekts beeinflussen (Schatten et al., 2010, S. 17). Stakeholder sind alle Personen oder Gruppen, die ein Interesse an dem Projekt haben oder von dessen Ergebnissen betroffen sind. Bei dem gegenständlichen Projekt handelt es sich um ein Einzelprojekt, bei dem die Rollen von Projektleiter, Entwickler und Tester in einer Person vereint sind. Zudem gibt es einen weiteren Stakeholder, den Betreuer, der das Projekt bewertet und gegebenenfalls Anforderungen vorgibt.

In einem Unternehmensumfeld wären zusätzlich Stakeholder wie das Management, eine IT-Abteilung oder die Finanz-Abteilung relevant. Für Projekte allgemein ist es wichtig, die relevanten Stakeholder zu identifizieren, um deren Anforderungen und Erwartungen frühzeitig zu berücksichtigen und somit die Entwicklung zielgerichtet und nutzerorientiert zu gestalten.

1.1 Stakeholder

1. *Projektleiter*: Verantwortlich für die Planung und Koordination des Projekts, übernimmt die *Gesamtverantwortung* für den Projekterfolg.
2. *Entwickler*: Setzt die Anforderungen technisch um und entwickelt die Anwendung.
3. *Tester*: Testet die Anwendung auf Funktionalität, Benutzerfreundlichkeit und Fehlerfreiheit.
4. Betreuer: Bewertet das Projekt und stellt sicher, dass die Anforderungen des Projekts erfüllt werden.
5. *Endnutzer*: Personen, die die Anwendung nutzen werden, um ihre Aufgaben zu priorisieren und ihre Produktivität zu steigern.

Die anvisierten Endnutzer der Anwendung sind Personen, die ihre Aufgaben effizient priorisieren und organisieren möchten.

Die Zielgruppe umfasst:

Berufstätige mit hohem Aufgabenaufkommen: Menschen, die täglich viele Aufgaben zu bewältigen haben und eine klare Priorisierung benötigen, um ihre Zeit effizient zu nutzen. Dazu gehören Manager, Teamleiter und Selbstständige.

Studenten: Studierende, die ihre akademischen Aufgaben und Termine besser organisieren möchten, um den Überblick über ihre Studienleistungen und Fristen zu behalten.

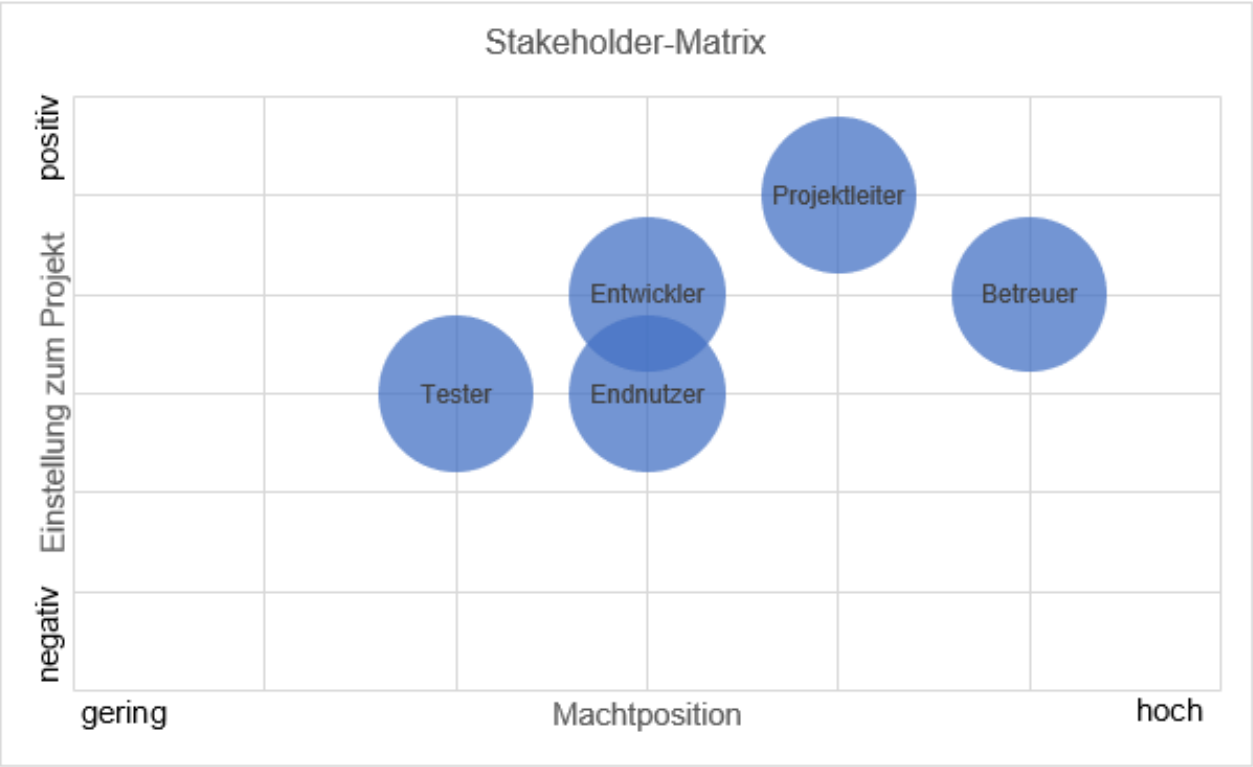
Privatpersonen mit komplexem Aufgabenmanagement: Menschen, die auch im privaten Bereich viele Verpflichtungen haben, wie z. B. Eltern, die sowohl berufliche als auch familiäre Aufgaben koordinieren müssen.

Produktivitätsorientierte Personen: Nutzer, die generell ein Interesse an Selbstorganisation und Produktivität haben und Tools zur Aufgaben-Priorisierung schätzen, um ihre Effizienz zu steigern.

1.2 Stakeholder-Matrix

Die Stakeholder-Matrix dient dazu, die verschiedenen Stakeholder des Projekts anhand ihrer Einstellung zum Projekt und ihrer Machtposition zu visualisieren. Sie zeigt, wie stark die einzelnen Stakeholder Einfluss auf das Projekt haben und ob sie diesem eher positiv oder negativ gegenüberstehen.

Abb. 1 – Stakeholder-Matrix



Quelle: eigene Darstellung

2. Funktionale Anforderungen

Für die Anforderungsanalyse des Projekts werden Best Practices angewandt, um sicherzustellen, dass die Anforderungen vollständig und korrekt erfasst werden (Ali Khan et al., 2015; Schatten et al., 2010; Sommerville, 2012).

Diese Best Practices umfassen:

- *Stakeholder-Einbindung*: Alle relevanten Stakeholder werden frühzeitig identifiziert und in den Anforderungsprozess einbezogen, um sicherzustellen, dass deren Bedürfnisse und Erwartungen berücksichtigt werden.
- *Iterative Verfeinerung*: Die Anforderungen werden iterativ überarbeitet, um sicherzustellen, dass sie auf die aktuellen Bedürfnisse der Stakeholder abgestimmt sind.
- *Priorisierung der Anforderungen*: Die funktionalen Anforderungen werden nach ihrer Priorität geordnet, um den Fokus auf die wichtigsten Funktionen zu legen. Dabei wird die MoSCoW-Methode[1] verwendet.
- *Dokumentation nach Standards*: Alle Anforderungen werden konsistent und detailliert dokumentiert, um Missverständnisse zu vermeiden und eine klare Grundlage für die Umsetzung zu schaffen.

[1] Die MoSCoW-Methode wird verwendet, um Anforderungen nach ihrer Priorität zu klassifizieren. Dabei werden die Kategorien 'Must have', 'Should have', 'Could have', und 'Won't have' genutzt, um eine klare Priorisierung der Anforderungen vorzunehmen (Ali Khan et al., 2015, S. 54).

2.1 Übersicht funktionale Anforderungen

In der folgenden Tabelle sind die funktionalen Anforderungen übersichtlich dargestellt. Jede Anforderung ist einer Kategorie zugeordnet und wurde gemäß der MoSCoW-Methode priorisiert, um den Fokus auf die wichtigsten Funktionen zu legen.

Tab. 1 – Übersicht funktionale Anforderungen

ID	Kategorie	Anforderung	Beschreibung	Priorität
Req_1	Essentials	Aufgabenverwaltung	Nutzer können Aufgaben erstellen, bearbeiten und löschen.	1
Req_2	Essentials	Priorisierung innerhalb der Matrix	Nutzer können Aufgaben nach Wichtigkeit, Dringlichkeit und Fitness priorisieren (Standardwerte werden lt. Req_8 gesetzt).	1
Req_3	GUI	Drag-and-Drop	Aufgaben können per Drag-and-Drop zwischen den sieben Feldern verschoben werden.	2
Req_4	GUI	Fälligkeitsdatum	Nutzer können ein Fälligkeitsdatum angeben.	2
Req_5	GUI	Benutzeroberfläche	Bereitstellung einer übersichtlichen und intuitiven Benutzeroberfläche.	1
Req_6	GUI	Such- und Filterfunktion	Nutzer können Aufgaben nach verschiedenen Kriterien suchen und filtern.	3
Req_7	Erweiterung	Benachrichtigung	Nutzer werden auf bevorstehende Fälligkeiten hingewiesen.	3
Req_8	Erweiterung	Standardprioritäten	Nutzer kann die automatisch gesetzten Standardprioritäten festlegen.	3
Req_9	Erweiterung	Archiv	Nutzer kann erledigte Aufgaben archivieren lassen.	3
				1 = Must have, *2 = Should have* *3 = Could have* *4 = Won't have*

Quelle: eigene Darstellung

2.2 Details funktionale Anforderrungen

Im folgenden Abschnitt werden die funktionalen Anforderungen detailliert beschrieben. Diese Beschreibung umfasst spezifische technische Details, Eingabevalidierungen und klare Akzeptanzkriterien, um sicherzustellen, dass jede Anforderung präzise und vollständig umgesetzt wird.

Req_1: Aufgabenverwaltung

Beschreibung: Nutzer können Aufgaben erstellen, bearbeiten und löschen. Beim Erstellen einer Aufgabe muss der Nutzer mindestens einen Titel angeben. Die Aufgabe kann zudem optional mit einer Beschreibung versehen werden. Das Fälligkeitsdatum und die Prioritäten können ebenfalls ausgewählt werden.

Eingabevalidierung: Der Titel ist ein Pflichtfeld und darf nicht leer sein. Fälligkeitsdatum darf nicht in der Vergangenheit liegen.

Technische Details: Speicherung erfolgt in einer lokalen SQLite-Datenbank.

Akzeptanzkriterien:

- Nutzer kann eine Aufgabe mit Titel erfolgreich erstellen.
- Wenn der Titel nicht eingegeben wird, wird eine Fehlermeldung angezeigt.
- Nutzer kann Beschreibung, Fälligkeitsdatum und Prioritäten festlegen.
- Dir Prioritäten werden standardmäßig lt. geltenden Einstellungen (Req_8) gesetzt.
- Wenn das Fälligkeitsdatum in der Vergangenheit liegt, wir eine Fehlermeldung angezeigt.
- Nutzer kann bestehende Aufgaben bearbeiten, und alle Änderungen werden in der Datenbank gespeichert.
- Nutzer kann eine Aufgabe löschen, die dann aus der Datenbank entfernt wird.

Req_2: Priorisierung innerhalb der Matrix

Beschreibung: Nutzer können Aufgaben nach Wichtigkeit, Dringlichkeit und Fitness priorisieren. Diese Priorisierung erfolgt beim Erstellen der Aufgabe mittels Dropdown-Menüs für jede Dimension (Importance, Urgency, Fitness). Standardwerte werden automatisch gesetzt, wenn keine Priorität explizit ausgewählt wird (siehe Req_8).

Eingabevalidierung: Nur vordefinierte Dropdown-Werte (High, Low) sind zulässig.

Technische Details: Speicherung der Prioritäten als Enum-Werte in der Datenbank.

Akzeptanzkriterien:

- Nutzer kann eine Aufgabe mit den entsprechenden Prioritäten bewerten.
- Dropdown-Menüs enthalten die Werte „High“ und „Low“ für jede Dimension.
- Standardwerte werden lt. Req_8 gesetzt, wenn keine Auswahl erfolgt.

- Fehlerhafte Eingaben oder das Belassen leerer Felder für die Priorisierung sind nicht möglich.

Req_3: Drag-and-Drop

Beschreibung: Nutzer können bereits erstellte Aufgaben, in der Sung-Diagramm-Ansicht, durch Drag-and-Drop in die verschiedenen Kategorien des Sung-Diagramms verschieben. Die Verschiebung erfolgt per Maus oder Touch-Geste, abhängig vom verwendeten Gerät. Aufgaben, bei denen alle Prioritäten (Wichtigkeit, Dringlichkeit, Fitness) auf "Low" gesetzt werden sollen, können in die "Low"-Spalte am Rand des Diagramms verschoben werden.

Eingabevalidierung: Eine Aufgabe kann nur innerhalb der Kategorien des Sung-Diagramms (einschließlich der "Low"-Spalte am Rand) verschoben werden. Das Verschieben außerhalb des Diagramm-Bereichs ist nicht möglich.

Technische Details: Die GUI ist für Drag-and-Drop auf Touch- und Desktop-Geräten optimiert. Bei Aufgaben mit allen "Low"-Prioritäten wird die Aufgabe visuell in der "Low"-Spalte dargestellt.

Akzeptanzkriterien:

- Nutzer kann Aufgaben per Drag-and-Drop in verschiedene Kategorien verschieben.
- Die Drag-and-Drop-Funktion funktioniert sowohl mit Maus als auch per Touch-Geste.
- Aufgaben mit allen Prioritäten auf "Low" werden automatisch in die dafür vorgesehene Spalte verschoben.

Req_4: Fälligkeitsdatum

Beschreibung: Nutzer können ein Fälligkeitsdatum für jede Aufgabe angeben. Das Datum darf nur in der Zukunft liegen.

Eingabevalidierung: Vergangene Daten sind nicht zulässig. Das Fälligkeitsdatum muss ein gültiges Kalenderdatum sein.

Technische Details: Es wird eine Kalenderauswahl bereitgestellt, die die Eingabe eines korrekten Datums erleichtert.

Akzeptanzkriterien:

- Nutzer kann das Fälligkeitsdatum mittels einer Kalenderauswahl setzen.
- Ein Fehler wird angezeigt, wenn ein Datum in der Vergangenheit gewählt wird.

Req_5: Benutzeroberfläche

Beschreibung: Die Anwendung bietet eine übersichtliche und intuitive Benutzeroberfläche. Oben befindet sich eine Navigationsleiste, über die der Nutzer auf die verschiedenen Module der Anwendung zugreifen kann, wie z.B. das Sung-Diagramm, die Aufgabenliste, Einstellungen, das Archiv oder

Benachrichtigungen. Alle wichtigen Funktionen sollen leicht erreichbar und visuell ansprechend dargestellt werden. Barrierefreiheit ist aktuell nicht geplant, könnte jedoch als zukünftiges Update berücksichtigt werden.

Technische Details: Die Benutzeroberfläche wird mit einem modernen UI-Framework umgesetzt (z.B. PyQt oder Tkinter).

Akzeptanzkriterien:

- Alle Hauptfunktionen sind innerhalb von zwei Klicks erreichbar.
- Die Anwendung passt sich an verschiedene Bildschirmgrößen (Desktop, Laptop) an.

Req_6: Such- und Filterfunktion

Beschreibung: Nutzer können Aufgaben nach verschiedenen Kriterien (z.B. Fälligkeitsdatum, Priorität) suchen und filtern. Die Such- und Filterfunktion wird in einer eigenen Ansicht (Aufgabenliste) angeboten, um eine bessere Übersicht zu gewährleisten. Filtereinstellungen werden gespeichert, sodass sie bei einem späteren Start der Anwendung wiederverwendet werden können.

Technische Details: Die Suche erfolgt mittels SQL-Abfragen auf der lokalen SQLite-Datenbank. Filtereinstellungen werden in einer separaten Konfigurationstabelle gespeichert.

Akzeptanzkriterien:

1. Nutzer kann Aufgaben nach allen vorhandenen Attributen filtern.
2. Die Suchanfragen liefern Ergebnisse in Echtzeit (weniger als 1 Sekunde Reaktionszeit).
3. Gespeicherte Filtereinstellungen werden beim nächsten Start der Anwendung automatisch übernommen.

Req_7: Benachrichtigung

Beschreibung: Nutzer werden über bevorstehende Fälligkeiten von Aufgaben, per Push-Benachrichtigung, benachrichtigt. Der Zeitraum für die Benachrichtigung kann konfiguriert werden. Die Benachrichtigungsfunktion kann ein- oder ausgeschalten werden. E-Mail-Benachrichtigungen könnten in einem zukünftigen Update ergänzt werden.

Technische Details: Benachrichtigungen werden mittels lokaler Push-Benachrichtigungen in Windows realisiert.

Akzeptanzkriterien:

- Nutzer erhält eine Benachrichtigung, wenn das Fälligkeitsdatum einer Aufgabe bevorsteht.
- Nutzer kann die Benachrichtigungen in den Einstellungen konfigurieren (ein-, ausschalten und Zeitraum festlegen).

Req_8: Standardprioritäten

Beschreibung: Nutzer kann die automatisch gesetzten Standardprioritäten festlegen (z.B. alle neuen Aufgaben auf "Low").

Technische Details: Die Werte werden in einer Benutzereinstellungen-Datenbank gespeichert.

Akzeptanzkriterien:

- Nutzer kann Standardprioritäten über das Einstellungsmenü festlegen.
- Diese Standardwerte werden für neu erstellte Aufgaben übernommen.
- Standardmäßig werden alle Werte auf „Low“ gesetzt, wenn der Nutzer keine andere Einstellung vornimmt.

Req_9: Archiv

Beschreibung: Nutzer kann erledigte Aufgaben archivieren lassen, anstatt sie zu löschen, um sie später einsehen zu können. Aufgaben im Archiv können zu einem späteren Zeitpunkt auch wieder reaktiviert werden.

Technische Details: Archivierte Aufgaben werden in einer separaten Datenbanktabelle gespeichert. Aufgaben können aus dem Archiv wieder aktiviert und in den aktiven Aufgabenbereich zurückverschoben werden.

Akzeptanzkriterien:

- Nutzer kann eine Aufgabe als „erledigt“ markieren, woraufhin sie ins Archiv verschoben wird.
- Archivierte Aufgaben bleiben über die Archiv-Ansicht verfügbar und können reaktiviert werden.

2.3 User Stories zu funktionalen Anforderungen

User Stories beschreiben Anforderungen aus der Sicht der Endbenutzer und helfen dabei, die Funktionalitäten der Anwendung verständlich und praxisnah darzustellen (Schatten et al., 2010, S. 19). Sie ermöglichen es, den Nutzen der Funktionen für die Nutzer besser zu verdeutlichen und eine benutzerzentrierte Entwicklung zu gewährleisten

Tab. 2 – User-Stories zu funktionalen Anforderungen

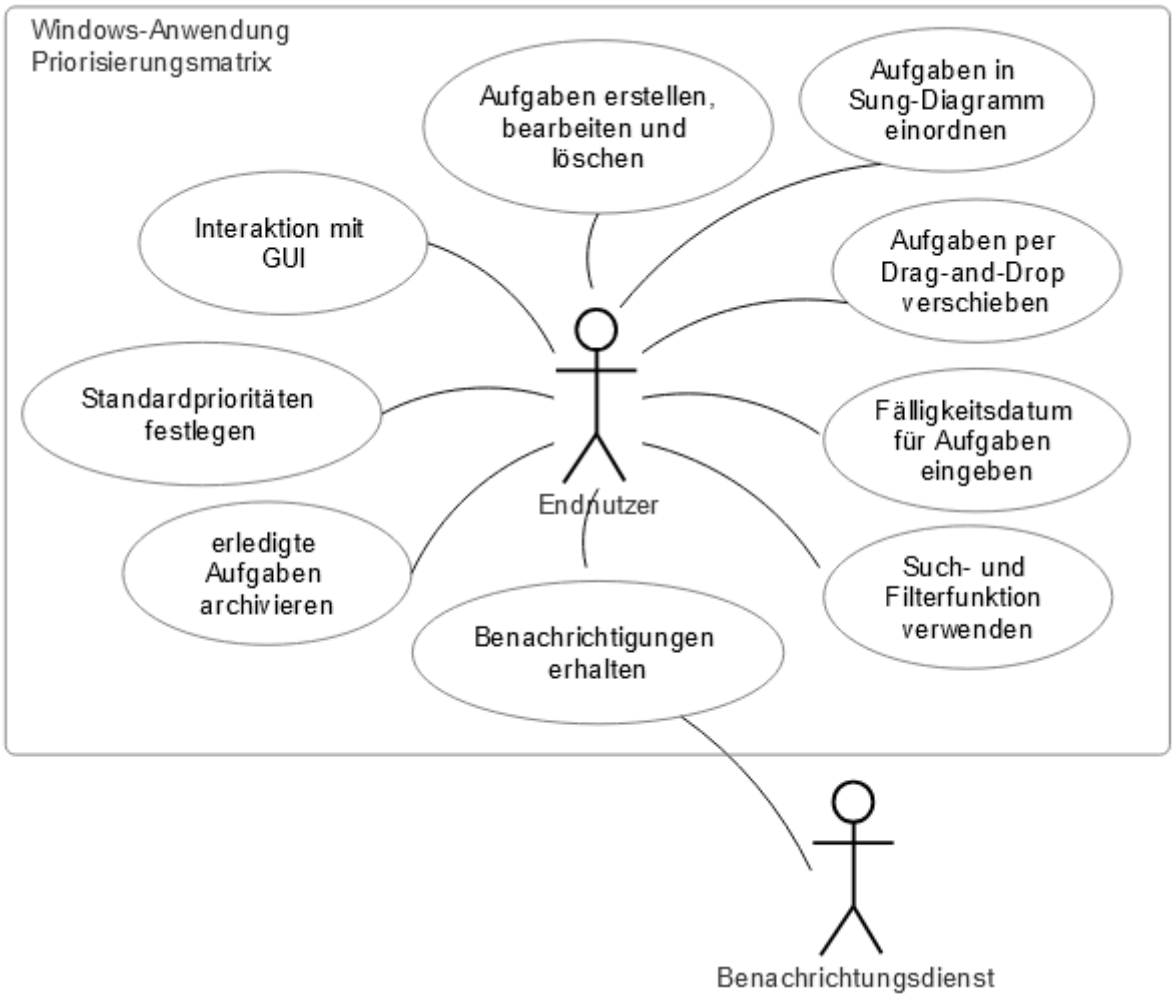
ID	Anforderung	User-Story
Req_1	Aufgabenverwaltung	Als Nutzer möchte ich Aufgaben erstellen, bearbeiten und löschen können, damit ich meine Aufgaben flexibel anpassen und verwalten kann. Ich möchte, dass diese Aktionen intuitiv über Buttons, Kontextmenüs und Textfelder erfolgen.
Req_2	Priorisierung innerhalb der Matrix	Als Nutzer möchte ich meine Aufgaben in die sieben Felder des Sung-Diagramms einordnen, um ihre Dringlichkeit, Wichtigkeit und Fitness genau zu priorisieren. Diese Priorisierung soll durch Dropdown-Menüs erfolgen.
Req_3	Drag-and-Drop	Als Nutzer möchte ich meine Aufgaben per Drag-and-Drop zwischen den sieben Feldern des Sung-Diagramms und der Low-Spalte verschieben können, um sie besser priorisieren zu können. Ich möchte, dass die Aufgabe nach dem Loslassen der Maus an der neuen Position bleibt.
Req_4	Fälligkeitsdatum	Als Nutzer möchte ich ein Fälligkeitsdatum für jede Aufgabe angeben können, damit ich den Überblick darüber behalte, bis wann eine Aufgabe erledigt sein muss. Es soll eine Kalenderansicht geben, die mir die Auswahl des Datums erleichtert.
Req_5	Benutzeroberfläche	Als Nutzer möchte ich eine übersichtliche und intuitive Benutzeroberfläche haben, damit ich die Anwendung einfach und ohne großen Aufwand bedienen kann. Die Navigation soll über eine Navigationsleiste erfolgen, die mir Zugriff auf alle Module der Anwendung (Sung-Diagramm, Aufgabenliste, Einstellungen, Archiv, Benachrichtigungen) ermöglicht.
Req_6	Such- und Filterfunktion	Als Nutzer möchte ich meine Aufgaben nach verschiedenen Kriterien (z.B. Fälligkeitsdatum, Priorität) suchen und filtern können, damit ich schnell die Informationen finde, die ich benötige. Die Filter sollen gespeichert werden, damit sie beim nächsten Start der Anwendung weiterhin aktiv sind.
Req_7	Benachrichtigung	Als Nutzer möchte ich Benachrichtigungen für bevorstehende Fälligkeiten erhalten, damit ich rechtzeitig daran erinnert werde, Aufgaben fristgerecht zu erledigen. Ich möchte in den Einstellungen festlegen können, ob Benachrichtigungen aktiviert sind und welchen Zeitraum sie abdecken sollen.
Req_8	Standardprioritäten	Als Nutzer möchte ich die automatisch gesetzten Standardprioritäten für neue Aufgaben festlegen können, damit ich die Priorisierung an meine persönliche Arbeitsweise anpassen und dadurch meine Aufgaben schneller und effizienter organisieren kann.
Req_9	Archiv	Als Nutzer möchte ich die Möglichkeit haben, erledigte Aufgaben automatisch archivieren zu lassen, damit ich eine bessere Übersicht über aktuelle Aufgaben behalte und bei Bedarf auf abgeschlossene Aufgaben zurückgreifen kann.

Quelle: eine Darstellung

2.4 Use-Case-Diagramm zu funktionalen Anforderungen

Das folgende Use-Case-Diagramm visualisiert die funktionalen Anforderungen. Es zeigt die Interaktionen der Endnutzer mit den verschiedenen Funktionen der Anwendung und stellt die zentralen Anwendungsfälle dar, die für die Umsetzung der Anforderungen relevant sind.

Abb. 2 – Use-Case-Diagramm zu funktionalen Anforderungen und Endnutzer



Quelle: eigene Darstellung

3. Nicht-Funktionale Anforderungen

Die nicht-funktionalen Anforderungen legen die Qualitätseigenschaften und Bedingungen fest, unter denen die Anwendung funktionieren soll. Sie sind ebenso entscheidend für den Erfolg des Projekts, da sie sicherstellen, dass die Anwendung nicht nur die gewünschten Funktionen erfüllt, sondern auch zuverlässig und nutzerfreundlich ist.

Tab. 3 – Nicht-Funktionale Anforderungen

ID	Kategorie	Anforderung	Beschreibung	Priorität
qReq_1	Qualität	Benutzerfreundlich	Die Anwendung soll eine einfache und intuitive Benutzeroberfläche haben. Die Benutzeroberfläche soll eine durchschnittliche Erlern-Zeit von unter 15 Minuten aufweisen, gemessen durch Usability-Tests. *Akzeptanzkriterium*: Die Anwendung wird von mindestens 80 % der Testnutzer als leicht verständlich bewertet.	1
qReq_2	Qualität	Leistung	Die Anwendung muss schnell auf Benutzerinteraktionen reagieren. *Akzeptanzkriterium*: Die Anwendung soll auf Benutzeraktionen innerhalb von 200ms reagieren. In Ausnahmefällen sind 1000ms zulässig. *Testverfahren*: Durchführung von automatisierten Performance-Tests, um die Reaktionszeiten zu überprüfen.	2
qReq_3	Qualität	Verfügbarkeit	Die Anwendung muss stabil laufen und jederzeit nutzbar sein, auch unter verschiedenen Bedingungen wie eingeschränkten Systemressourcen. *Akzeptanzkriterium*: Die Anwendung muss eine 99,9 % Verfügbarkeit während der Nutzungszeit haben und auf Speicher- oder CPU-Beschränkungen robust reagieren. *Testmethoden*: Stabilitätstests unter Ressourcenbeschränkungen.	1
qReq_4	Qualität	Kompatibilität	Die Anwendung wird als native Desktop-App für Windows 10/11 entwickelt. *Akzeptanzkriterium*: Die Anwendung muss auf mindestens 99 % der getesteten Windows 10/11-Umgebungen lauffähig sein.	1
qReq_5	Qualität	Sicherheit	Die Anwendung soll den Datenschutz gewährleisten, insbesondere der Benutzerdaten. Bzgl. Benutzer sollen nur der Benutzername (Nickname) und das Passwort gespeichert werden. Das Passwort wird sicher gehasht, um die Sicherheit der	1

			Nutzerdaten zu gewährleisten. Die Speicherung entspricht den Vorgaben der DSGVO. *Akzeptanzkriterium*: Durchführung eines Sicherheits-Audits, das sicherstellt, dass die Passwörter korrekt gehasht und keine unverschlüsselten sensiblen Daten gespeichert werden. Zudem werden statische Code-Analyse-Tools verwendet, um sicherheitsrelevante Schwachstellen zu identifizieren und zu beheben.	
qReq_6	Qualität	Erweiterbarkeit	Die App soll so konzipiert werden, dass Erweiterungen einfach möglich sind. Es sollen geeignete Schnittstellen und Architekturmuster wie Modularisierung und Abstraktionsschichten verwendet werden, um zukünftige Änderungen zu erleichtern. *Akzeptanzkriterium*: Die App soll innerhalb eines Monats um ein neues Modul erweitert werden können, ohne größere Änderungen an der bestehenden Architektur.	3
				1 = Must have, *2 = Should have* *3 = Could have* *4 = Won't have*

Quelle: eigene Darstellung

4. Glossar

Drag-and-Drop: Eine Benutzerinteraktionsmethode, bei der ein Objekt mit der Maus ausgewählt und an eine andere Stelle gezogen wird.

Eisenhower-Matrix: Eine Methode zur Aufgabenpriorisierung, die Aufgaben in vier Quadranten basierend auf Dringlichkeit und Wichtigkeit einteilt.

GUI (Graphical User Interface): Eine grafische Benutzeroberfläche, die es den Nutzern ermöglicht, über visuelle Elemente wie Buttons und Menüs mit der Anwendung zu interagieren.

JSON: Ein Format zur strukturierten Übertragung von Daten zwischen Systemen, das für den Benachrichtigungsdienst verwendet wird .

MoSCoW-Methode: Eine Methode zur Priorisierung von Anforderungen. Die Kategorien sind: Must have (muss erfüllt werden), Should have (sollte erfüllt werden), Could have (könnte erfüllt werden), und Won't have (wird nicht erfüllt).

Shift-Left-Ansatz: Ein Prinzip im Software-Engineering, das darauf abzielt, Qualität und Tests frühzeitig in den Entwicklungsprozess zu integrieren. Dadurch werden Fehler und Risiken möglichst früh erkannt und behoben, was zu einer effizienteren Entwicklung und einer höheren Gesamtqualität führt. Der Ansatz fördert die frühe Einbindung von Stakeholdern, eine iterative Überprüfung von Anforderungen und eine frühzeitige Qualitätssicherung.

SQLite: Eine relationale Datenbank, die für die Speicherung der Anwendungsdaten verwendet wird. Sie eignet sich besonders für Desktop-Anwendungen, da keine serverseitige Konfiguration erforderlich ist

Sung-Diagramm: Eine erweiterte Version der Eisenhower-Matrix, die Aufgaben in sieben Felder statt in die klassischen vier Quadranten einteilt, um eine differenziertere Priorisierung zu ermöglichen.

Use Case: Eine Beschreibung der typischen Interaktionen zwischen dem Benutzer und der Anwendung, um eine bestimmte Funktionalität zu erreichen.

Literaturverzeichnis

Ali Khan, J., Ur Rehman, I., Hayat Khan, Y., Javed Khan, I., & Rashid, S. (2015). Comparison of Requirement Prioritization Techniques to Find Best Prioritization Technique. *International Journal of Modern Education and Computer Science*, *7*(11), 53–59. https://doi.org/10.5815/ijmecs.2015.11.06

Schatten, A., Biffl, S., Demolsky, M., Gostischa-Franta, E., Östreicher, T., & Winkler, D. (2010). *Best Practice Software-Engineering: Eine Praxiserprobte Zusammenstellung Von Komponenten-orientierten Konzepten, Methoden und Werkzeugen*. Spektrum Akademischer Verlag GmbH. http://ebookcentral.proquest.com/lib/badhonnef/detail.action?docID=511284

Sommerville, I. (2012). *Software Engineering*. Pearson Deutschland GmbH. http://ebookcentral.proquest.com/lib/badhonnef/detail.action?docID=5133710

Portfolio – Phase 2

IU Internationale Hochschule Fernstudium

Studiengang: Master of Science Informatik

Kurs: Projekt – Software Engineering (DLMCSPSE01_D)

Spezifikationsdokument

Windows Anwendung:
Priorisierungsmatrix nach dem Sung-Diagramm

Gregor Hubmann

Matrikelnummer: 9196398

Hornstraße 185

5423 St. Koloman

Abgabedatum (Phase 2): 24.11.2024

Änderungen nach Phase 1:
Keine Änderungen vorgenommen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Das gegenständliche Spezifikationsdokument beschreibt die technischen Eigenschaften und das Verhalten der Anwendung. Es dient dazu, die zentralen Geschäftsobjekte, Geschäftsprozesse, Geschäftsregeln, Systemschnittstellen und Benutzerschnittstellen der Anwendung detailliert zu definieren. Diese Spezifikation stellt sicher, dass die Anforderungen aus dem Anforderungsdokument technisch präzise und nachvollziehbar umgesetzt werden können.

Das Dokument beginnt mit der Beschreibung des Datenmodells, welches die wichtigsten Geschäftsobjekte und deren Beziehungen zueinander darstellt. Die zentralen Geschäftsprozesse werden anschließend erläutert, gefolgt von einer detaillierten Beschreibung der Geschäftsregeln, die die Konsistenz und Qualität der Anwendung sicherstellen. Weiterhin werden die technischen Schnittstellen beschrieben, die die Kommunikation mit externen Systemen ermöglichen. Schließlich umfasst das Spezifikationsdokument eine detaillierte Beschreibung der Benutzerschnittstellen, um die Struktur und das Verhalten der wichtigsten Dialoge darzustellen.

1. Datenmodell

Das Datenmodell der Anwendung beschreibt die zentralen Geschäftsobjekte und deren Beziehungen zueinander (Sommerville, 2012, S. 167). Es bildet die Grundlage für die Datenhaltung und strukturiert die relevanten Informationen, die die Anwendung zur Verwaltung und Priorisierung von Aufgaben benötigt. Die wichtigsten Geschäftsobjekte sind Tasks (Aufgaben), Categories (Kategorien) und User (Benutzer).

In der vorliegenden Konzeption liegt der Fokus auf der Modellierung der Attribute und Beziehungen der Geschäftsobjekte. Die Methoden, die später die Geschäftslogik implementieren (z. B. zum Erstellen, Bearbeiten oder Löschen von Aufgaben), wurden zu diesem Zeitpunkt bewusst weggelassen, um eine flexible und übersichtliche Datenstruktur zu gewährleisten. Diese Methoden werden im Architekturdokument spezifiziert, das eine detaillierte Darstellung der Klassen und ihrer Funktionalitäten bietet.

1.1 Geschäftsobjekte

Die Geschäftsobjekte bilden die Grundlage der Anwendung und definieren die zentralen Datenstrukturen. Die Geschäftsobjekte sind Task, Category und User. Jedes dieser Objekte hat spezifische Attribute und Beziehungen, die ihre Funktion innerhalb der App beschreiben:

Die Attribute der einzelnen Geschäftsobjekte beschreiben die jeweiligen Daten, die innerhalb der Anwendung gespeichert und verarbeitet werden. Die Beziehungen verdeutlichen, wie die einzelnen Objekte miteinander verknüpft sind.

Tab. 1 - Geschäftsobjekte

Geschäftsobjekt	Attribute	Beschreibung	Beziehungen
Task		Eine Aufgabe, die der Nutzer erstellt und im Sung-Diagramm priorisiert	Eine Task gehört zu einer Category
	Title (String)	Kurze Beschreibung der Aufgabe	
	Description (String, optional)	Detaillierte Information über die Aufgabe	
	Due Date (Date)	Datum, bis zu dem die Aufgabe erledigt sein soll	
	Importance (Enum: High, Low)	Wichtigkeit der Aufgabe	
	Urgency (Enum: High, Low)	Dringlichkeit der Aufgabe	
	Fitness (Enum: High, Low)	Bewertung der eigenen Fähigkeit zur Erledigung der Aufgabe	
	Status (Enum: Open, In Progress, Completed)	Aktueller Stand der Aufgabe	
Category		Bezeichnung der Kategorie (z.B. High Importance & High Urgency)	Eine Category kann mehrere Tasks enthalten
	Name (String)	Bezeichnung der Kategorie (z. B. High Importance & High Urgency & High Fitness)	
	Description (String, optional)	Erklärung der Kategorie	
User		Der Nutzer der Anwendung, der Aufgaben erstellt, bearbeitet und priorisiert	Ein User kann mehrere Tasks erstellen
	Username (String)	Der Name des Nutzers	
	PasswordHash (String)	Zur Authentifizierung, Hash-Wert	

Quelle: eigene Darstellung

1.2 UML-Klassendiagramm

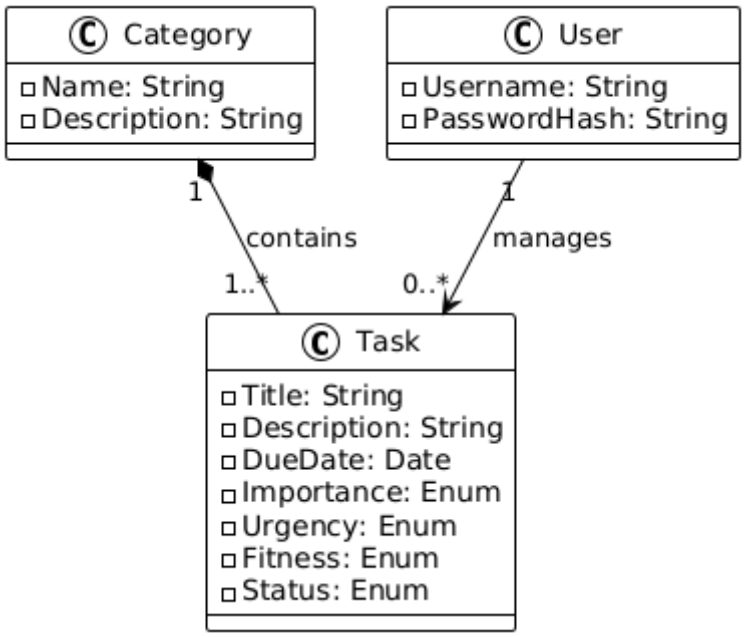
Das UML-Klassendiagramm stellt die Struktur der zentralen Geschäftsobjekte der Anwendung dar und zeigt die Beziehungen zwischen diesen Objekten auf. Die Klassen im Diagramm sind User, Task und Category.

- User repräsentiert den Benutzer der Anwendung, der Aufgaben erstellt und verwaltet.
- Task stellt eine Aufgabe dar, die der Benutzer im Sung-Diagramm priorisiert. Jede Aufgabe hat Attribute wie Title, DueDate, Importance, Urgency, und Fitness, die ihre Eigenschaften beschreiben.
- Category beschreibt die Priorisierung der Aufgaben basierend auf den verschiedenen Kombinationen von Importance, Urgency, und Fitness.

Die Beziehungen zwischen den Klassen sind ebenfalls dargestellt:

- Ein User kann mehrere Tasks verwalten, was als Assoziation im Diagramm dargestellt ist.
- Jede Task gehört zu genau einer Category, während eine Category mehrere Tasks enthalten kann. Diese Beziehung wird durch Aggregation dargestellt.

Abb. 1 – UML-Klassendiagramm



Quelle: eigene Darstellung

2. Geschäftsprozesse

Die Geschäftsprozesse beschreiben die zentralen Abläufe und Nutzerinteraktionen innerhalb der Anwendung. Die Geschäftsprozesse umfassen das Erstellen und Kategorisieren von Aufgaben, das Bearbeiten und Verschieben von Aufgaben, die Benachrichtigung zu fälligen Aufgaben sowie die Anzeige der Aufgabenübersicht.

2.1 Aufgabe erstellen und kategorisieren

Beschreibung: Der Nutzer erstellt eine neue Aufgabe, die er mit einem Titel, einer Beschreibung (optional) und einem Fälligkeitsdatum (optional) versehen kann. Anschließend wählt der Nutzer die entsprechenden Prioritäten für Importance, Urgency, und Fitness. Nach der Eingabe der Prioritäten wird die Aufgabe entweder in eine der sieben Kategorien des Sung-Diagramms oder, bei allen Werten auf „Low“, in eine separate Spalte am Rand eingeordnet. Die Standardwerte für die Prioritäten können vom Nutzer in den Einstellungen angepasst werden.

Ziel: Sicherstellen, dass jede Aufgabe gleich bei der Erstellung sinnvoll kategorisiert wird, damit sie korrekt im Sung-Diagramm erscheint.

Ablauf:

1. Nutzer klickt auf „Neue Aufgabe erstellen“.
2. Nutzer gibt die Attribute der Aufgabe ein (Title, Description (optional), Due Date).
3. Nutzer wählt die Kategorien für Importance, Urgency und Fitness.
4. Aufgabe wird der entsprechenden Kategorie zugeordnet und im Sung-Diagramm angezeigt.

2.2 Aufgabe bearbeiten

Beschreibung: Der Nutzer kann eine bestehende Aufgabe bearbeiten. Dabei können Attribute wie Titel, Beschreibung, Fälligkeitsdatum, oder die Kategorie geändert werden.

Ziel: Nutzer sollen Aufgaben flexibel anpassen können, z. B. wenn sich die Urgency oder Importance ändert.

Ablauf:

1. Nutzer wählt eine vorhandene Aufgabe aus.
2. Nutzer bearbeitet die Attribute (Title, Description (optional), Due Date).
3. Nutzer kann die Kategorien (Importance, Urgency, Fitness) anpassen.
4. Die Änderungen werden gespeichert und die Aufgabe wird im Sung-Diagramm aktualisiert.

2.3 Aufgabe verschieben

Beschreibung: Eine Aufgabe wird innerhalb des Sung-Diagramms in eine andere Kategorie verschoben. Dies kann durch Drag-and-Drop geschehen, falls sich z. B. die Urgency oder Importance einer Aufgabe verändert.

Ziel: Aufgaben flexibel und intuitiv neu priorisieren zu können, wenn sich die Umstände ändern.

Ablauf:

1. Nutzer zieht eine Aufgabe per Drag-and-Drop in eine andere Kategorie.
2. Die Attribute (Importance, Urgency, Fitness) werden entsprechend angepasst.
3. Die neue Position im Sung-Diagramm wird gespeichert.

2.4 Benachrichtigungen zu Aufgaben

Beschreibung: Der Nutzer erhält eine Benachrichtigung, wenn das Fälligkeitsdatum einer Aufgabe näher rückt. Diese Benachrichtigungen helfen dabei, rechtzeitig an anstehende Aufgaben erinnert zu werden. Der Zeitraum ab wann Benachrichtigungen gesendet werden, kann vom Nutzer in den Einstellungen angepasst werden. Die Funktion kann vom Nutzer in den Einstellungen ein- und ausgeschalten werden.

Ziel: Nutzer sollen keine wichtigen Aufgaben vergessen und rechtzeitig daran erinnert werden.

Ablauf:

1. Das System prüft automatisch täglich die Fälligkeitsdaten aller Aufgaben.
2. Wenn eine Aufgabe im entsprechenden Zeitraum fällig wird, wird eine Benachrichtigung erstellt.
3. Der Nutzer erhält eine Benachrichtigung auf der Benutzeroberfläche.

2.5 Aufgabenübersicht anzeigen

Beschreibung: Der Nutzer kann sich eine Übersicht aller Aufgaben anzeigen lassen, sowohl sortiert nach Kategorien im Sung-Diagramm als auch in einer Listenansicht.

Ziel: Eine klare und strukturierte Übersicht über alle Aufgaben zu bieten, um eine gute Planung und Priorisierung zu ermöglichen.

Ablauf:

1. Nutzer öffnet die Aufgabenübersicht.
2. Aufgaben werden entweder als Sung-Diagramm oder als Liste angezeigt.
3. Der Nutzer kann Filter anwenden, um z. B. nur Aufgaben einer bestimmten Kategorie anzuzeigen.

2.6 Archivierung von erledigten Aufgaben

Beschreibung: Der Nutzer kann erledigte Aufgaben archivieren bzw. automatisch archivieren lassen, um die Übersicht über aktive Aufgaben zu bewahren. Archivierte Aufgaben bleiben zur späteren Einsicht erhalten, können aber nicht mehr aktiv bearbeitet werden.

Ziel: Die klare Trennung von aktiven und erledigten Aufgaben zu ermöglichen und so die Übersichtlichkeit zu verbessern.

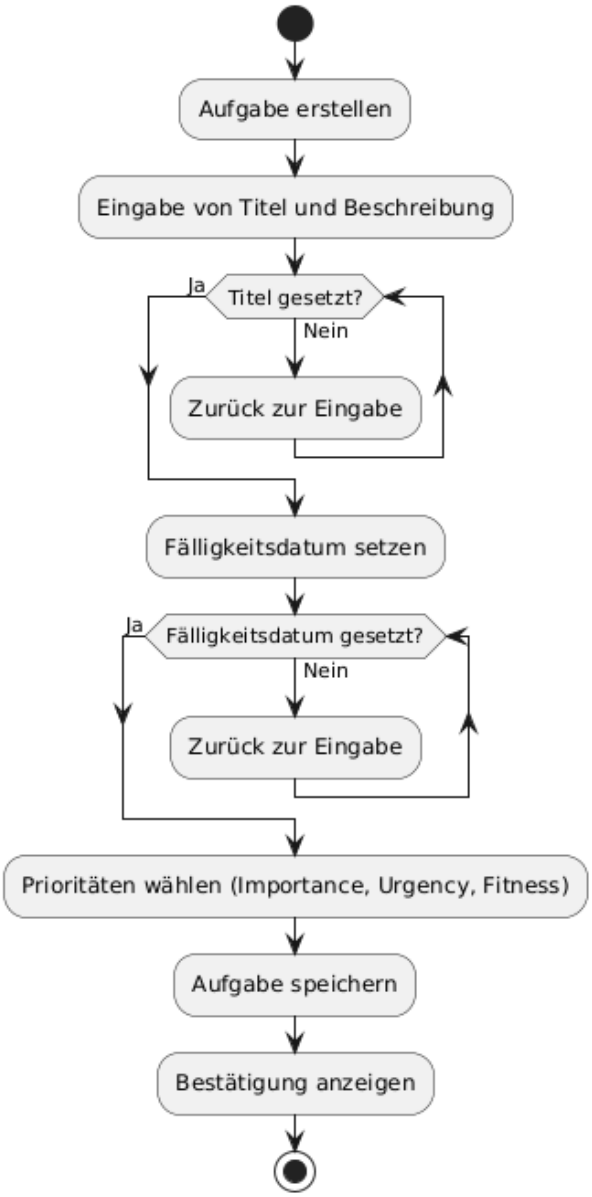
Ablauf:

1. Der Nutzer markiert eine Aufgabe als „Erledigt“.
2. Die Anwendung überprüft, ob die automatische Archivierung aktiviert ist.
3. Falls aktiviert: Die Aufgabe wird automatisch ins Archiv verschoben.
4. Falls deaktiviert: Die Aufgabe bleibt im Aufgabenbereich und kann manuell archiviert werden.
5. Der Nutzer kann das Archiv öffnen, um archivierte Aufgaben einzusehen oder wiederherzustellen.

2.7 UML-Aktivitätsdiagramm zu Kernprozess - Aufgabe erstellen und kategorisieren

Das folgende UML-Aktivitätsdiagramm visualisiert den zentralen Kernprozess "Aufgabe erstellen und kategorisieren" der Anwendung. Dieser Prozess beschreibt den Ablauf von der Erstellung einer neuen Aufgabe bis hin zur Speicherung im System.

Abb. 2 – UML-Aktivitätsprogramm – Aufgabe erstellen und kategorisieren



Quelle: eigene Darstellung

3. Geschäftsregeln

Die Geschäftsregeln definieren die Bedingungen und Einschränkungen, die für die Anwendung gelten und sicherstellen, dass die Prozesse konsistent und korrekt ablaufen. Sie dienen als Grundlage für die Implementierung der zentralen Abläufe und gewährleisten die Einhaltung bestimmter Vorgaben und Regeln.

Fälligkeitsdatum muss gesetzt sein

- Regel: Jede Aufgabe (Task) muss ein Fälligkeitsdatum (Due Date) besitzen.
- Zweck: Diese Regel stellt sicher, dass alle Aufgaben termingerecht geplant werden können und ermöglicht die Generierung von Benachrichtigungen.

Jede Aufgabe muss kategorisiert werden

- Regel: Eine Task muss immer einer Category zugeordnet sein, basierend auf der Kombination von Importance, Urgency und Fitness.
- Zweck: Diese Regel sorgt dafür, dass jede Aufgabe sinnvoll im Sung-Diagramm eingeordnet wird, um den Nutzen der Priorisierung sicherzustellen.

Ein Benutzer kann nur seine eigenen Aufgaben bearbeiten

- Regel: Jeder User kann nur die von ihm erstellten Tasks bearbeiten und einsehen.
- Zweck: Diese Regel dient der Datensicherheit und stellt sicher, dass nur der Besitzer der Aufgabe Änderungen vornehmen kann.

Aufgabenpriorität darf jederzeit geändert werden

- Regel: Nutzer dürfen die Priorität einer Aufgabe (d.h. Importance, Urgency, und Fitness) jederzeit ändern.
- Zweck: Diese Regel stellt sicher, dass Aufgaben flexibel angepasst werden können, wenn sich die Umstände ändern, und unterstützt damit eine dynamische Planung.

Eindeutiger Titel für Aufgaben

- Regel: Der Titel (Title) einer Aufgabe muss innerhalb eines Nutzers eindeutig sein.
- Zweck: Diese Regel hilft dem Nutzer, Aufgaben zu unterscheiden, und verhindert Verwirrung durch doppelte Titel.

Aufgabenstatus

- Regel: Der Status (Status) einer Aufgabe kann nur eine der folgenden Zustände haben: Open, In Progress, Completed.
- Zweck: Diese Regel stellt sicher, dass der Fortschritt jeder Aufgabe klar definiert ist und leicht nachverfolgt werden kann.

Standardwerte für Kategorisierung

- Regel: Eine neu erstellte Aufgabe erhält standardmäßig die Prioritäten Importance, Urgency und Fitness jeweils auf Low (bzw. gemäß vorgenommenen Einstellungen), falls der Nutzer keine expliziten Werte auswählt.
- Zweck: Diese Regel sorgt dafür, dass keine unvollständigen Aufgaben im System vorhanden sind und erhöht die Benutzerfreundlichkeit.

Archivierung von Aufgaben

- Regel: Erledigte Aufgaben sollen entweder automatisch oder manuell archiviert oder gelöscht werden, um die Übersichtlichkeit zu erhöhen.
- Zweck: Diese Regel stellt sicher, dass die Benutzeroberfläche nicht mit erledigten Aufgaben überladen wird und die Übersichtlichkeit gewahrt bleibt.

Benachrichtigungskonfiguration

- Regel: Der Nutzer kann die Benachrichtigungen für Aufgaben aktivieren oder deaktivieren sowie den Benachrichtigungszeitraum einstellen.
- Zweck: Diese Regel dient dazu, eine flexible Benachrichtigungsfunktion bereitzustellen, die den individuellen Bedürfnissen des Nutzers entspricht.

Eingabevalidierung

- Regel: Alle Pflichtfelder (Titel, Fälligkeitsdatum) müssen ausgefüllt werden. Eingaben werden überprüft, um SQL-Injections und Pufferüberläufe zu verhindern.
- Zweck: Diese Regel stellt sicher, dass die Eingaben der Nutzer validiert werden, um Sicherheitsrisiken zu minimieren und die Datenintegrität zu gewährleisten.

4. Systemschnittstellen

Die Systemschnittstellen der Anwendung beschreiben die Verbindungen zu anderen Systemkomponenten, die für den reibungslosen Betrieb notwendig sind (Sommerville, 2012, S. 65). Diese Schnittstellen legen fest, wie Daten ausgetauscht, gespeichert und verarbeitet werden. Für die Anwendung wurden zwei zentrale Schnittstellen definiert: der Benachrichtigungsdienst, der den Nutzer rechtzeitig an bevorstehende Aufgaben erinnert, und die Datenbankspeicherung, die alle Aufgaben und Nutzerdaten nachhaltig sichert.

4.1 Benachrichtigungsdienst

Zweck: Der Benachrichtigungsdienst dient dazu, den Nutzer rechtzeitig über anstehende Aufgaben zu informieren. Dies stellt sicher, dass der Nutzer keine wichtigen Aufgaben verpasst, insbesondere wenn ein Fälligkeitsdatum erreicht ist.

Verwendetes Protokoll: HTTP/HTTPS – Die Kommunikation zwischen der Anwendung und dem Benachrichtigungsdienst erfolgt über das HTTP- oder HTTPS-Protokoll.

Datenformat: JSON – Die Daten zur Aufgabe, wie Titel und Fälligkeitsdatum, werden im JSON-Format übertragen. Dies ermöglicht eine einfache und strukturierte Übertragung der Informationen.

Beschreibung: Die Anwendung kann lokal einen Benachrichtigungsdienst bereitstellen oder auch eine Verbindung zu einem externen Dienst herstellen, der die Benachrichtigungen übernimmt. Sobald das Fälligkeitsdatum einer Aufgabe näher rückt, sendet die App eine HTTP-Anfrage an den Benachrichtigungsdienst. Der Dienst verarbeitet die Anfrage und generiert eine Benachrichtigung, die lokal angezeigt wird.

4.2 Speicherung in der Datenbank

Zweck: Die Speicherung der Aufgaben und Nutzerdaten in einer Datenbank dient dazu, alle erfassten Informationen dauerhaft zu sichern und für spätere Abfragen bereitzuhalten. Dies ermöglicht eine nachhaltige Verwaltung der Aufgaben und deren Prioritäten.

Verwendetes Protokoll: SQLite – Für die lokale Speicherung wird SQLite verwendet, eine leichtgewichtige relationale Datenbank, die sich besonders gut für Desktop-Anwendungen eignet.

Datenformat: SQL-basierte Datenstrukturen – Die Daten werden in SQL-Tabellen gespeichert, die die zentralen Geschäftsobjekte wie Tasks, Categories, und Users abbilden.

Beschreibung: Für die Speicherung der Daten wird eine lokale SQLite-Datenbank verwendet, die direkt in der Anwendung eingebunden wird. Die Datenbank speichert alle Informationen zu Aufgaben, inklusive Titel, Beschreibung, Fälligkeitsdatum, und den gewählten Prioritäten. Der Zugriff auf die Datenbank erfolgt über standardisierte SQL-Abfragen, die sowohl das Erstellen, Lesen, Aktualisieren als auch das Löschen von Aufgaben ermöglichen (CRUD-Operationen). Dies stellt eine robuste und nachhaltige Lösung für die Datenverwaltung innerhalb der Anwendung dar.

4.3 Schnittstelle zwischen GUI und Backend

Zweck: Die Schnittstelle zwischen der grafischen Benutzeroberfläche (GUI) und dem Backend ermöglicht die Verarbeitung von Nutzeraktionen, wie das Erstellen oder Bearbeiten von Aufgaben, und sorgt dafür, dass die Ergebnisse dem Nutzer direkt angezeigt werden. Dadurch wird eine nahtlose Benutzererfahrung gewährleistet.

Beschreibung: Die GUI sendet bei Nutzerinteraktionen Anfragen an das Backend der Anwendung. Diese Anfragen werden von der Backend-Logik verarbeitet, die wiederum auf die Datenbank zugreift, um die benötigten Daten zu lesen, zu aktualisieren oder zu löschen. Die Ergebnisse werden zurück an die GUI gesendet, sodass der Nutzer die Änderungen sofort sehen kann.

Verwendetes Protokoll: Da es sich um eine Desktop-Anwendung handelt, erfolgt die Kommunikation zwischen GUI und Backend lokal über interne Funktionsaufrufe, was eine direkte und performante Interaktion ermöglicht.

Datenformat: Die Daten werden als Python-Objekte, einfache JSON-Datenstrukturen oder SQL-Abfragen innerhalb der Anwendung übertragen. Dies gewährleistet eine einfache Handhabung und minimiert die Komplexität der Datenverarbeitung.

5. Benutzerschnittstellen

Die Benutzerschnittstellen der Anwendung definieren die Art und Weise, wie der Nutzer mit der Anwendung interagiert (Schatten et al., 2010, S. 32). Die Anwendung bietet verschiedene Schnittstellen, um Aufgaben zu erstellen, zu priorisieren und die persönlichen Einstellungen anzupassen. Der Fokus liegt dabei auf einer übersichtlichen Struktur, die die Priorisierung anhand des Sung-Diagramms unterstützt und dem Nutzer eine nahtlose Verwaltung seiner Aufgaben ermöglicht. Darüber hinaus wird auf den Dialogfluss und die Eingabevalidierung eingegangen, um eine konsistente und fehlerfreie Nutzung der App sicherzustellen.

5.1 GUI

Eine klare und benutzerfreundliche Gestaltung der grafischen Benutzeroberfläche (GUI) ist entscheidend, um eine effiziente und intuitive Nutzung zu ermöglichen.

Hauptansicht der Anwendung

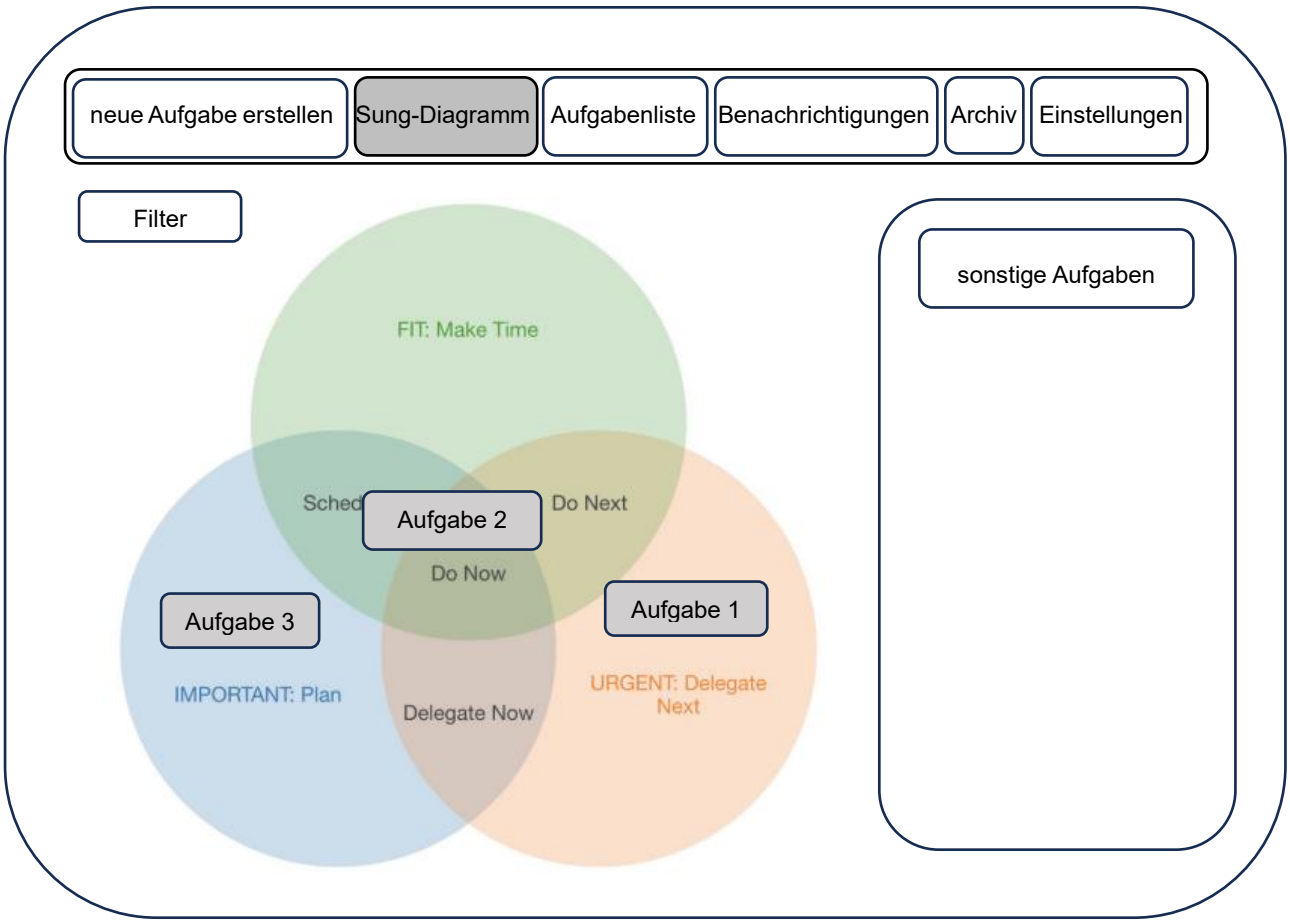
Beschreibung: Die Hauptansicht der Anwendung zeigt das Sung-Diagramm oder die Aufgabenliste, in das die Aufgaben kategorisiert werden können. Die Hauptansicht bietet den Nutzern eine visuelle Übersicht ihrer Aufgaben.

Bestandteile:
Navigationsleiste: Oben befindet sich eine Navigationsleiste, über die der Nutzer auf die verschiedenen Module der App zugreifen kann, wie neue Aufgabe erstellen, Sung-Diagramm, Aufgabenliste, Einstellungen, Archiv oder Benachrichtigungen.

Sung-Diagramm: Ein Bereich, der die sieben Kategorien des Sung-Diagramms visualisiert. Nutzer können ihre Aufgaben per Drag-and-Drop in die verschiedenen Felder des Diagramms verschieben. Neben den Hauptkategorien des Sung-Diagramms gibt es eine zusätzliche Spalte, die Aufgaben auflistet, deren Prioritäten alle auf „Low“ gesetzt sind.

Aufgabenliste: Eine Liste mit allen Aufgaben, inklusive einer Such- und Filterfunktion, um Aufgaben schnell zu finden und anzuzeigen.

Abb. 3 – Skizze – Hauptansicht der Anwendung



Quelle: eigene Darstellung

5.1.1 Dialog zum Erstellen und Bearbeiten von Aufgaben

Beschreibung: Der Erstellen-/Bearbeiten-Dialog wird verwendet, um neue Aufgaben hinzuzufügen oder bestehende Aufgaben zu bearbeiten. Hier werden die relevanten Daten der Aufgabe eingegeben oder angepasst.

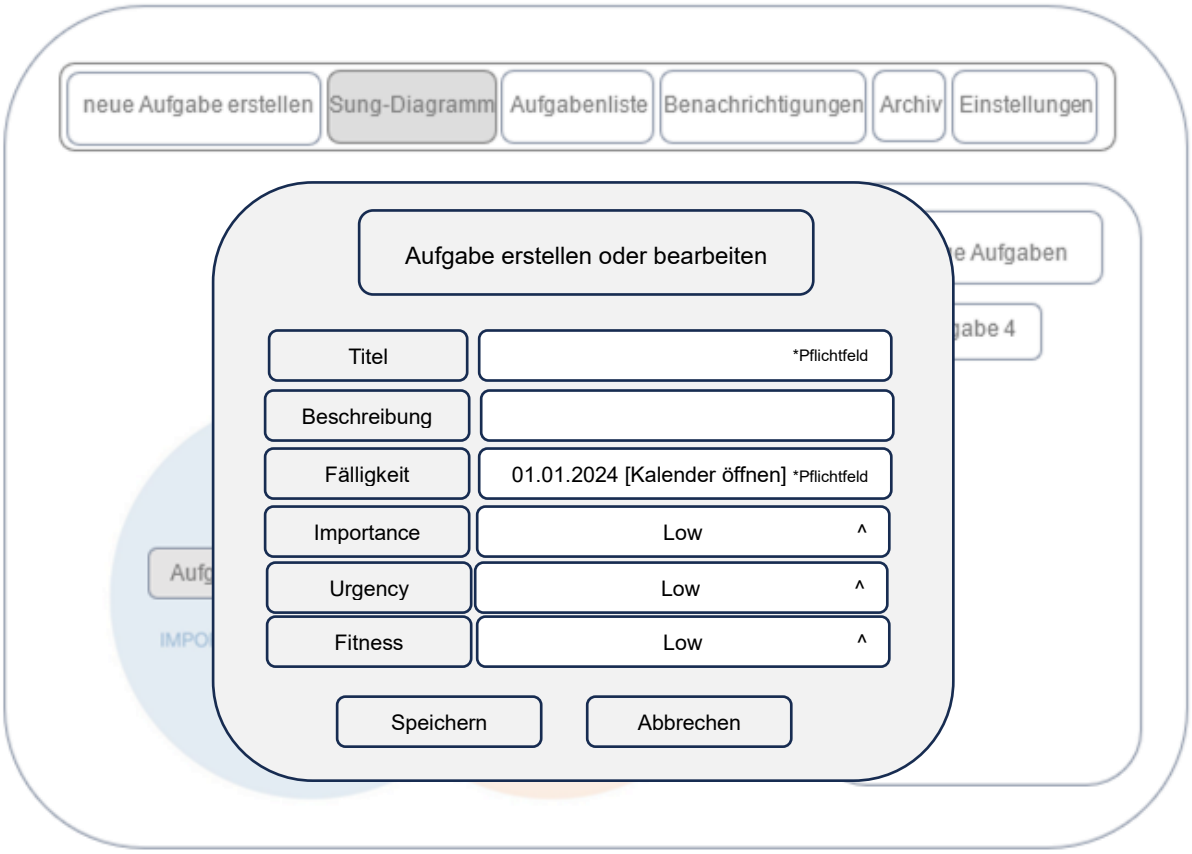
Bestandteile:
Titel und Beschreibung: Ein Pflichtfeld für den Titel und ein optionales Feld für eine detaillierte Beschreibung der Aufgabe.

Fälligkeitsdatum: Der Nutzer muss ein Fälligkeitsdatum festlegen, das die Grundlage für Benachrichtigungen bildet.

Prioritätenauswahl: Felder zur Auswahl der Importance, Urgency, und Fitness der Aufgabe. Standardmäßig sind alle Werte auf Low gesetzt (kann in den Einstellungen geändert werden), sollten aber angepasst werden.

Speichern und Abbrechen: Der Nutzer kann die Eingaben speichern oder den Prozess abbrechen.

Abb. 4 – Skizze – Dialog zur Erstellung und Bearbeitung von Aufgaben



Quelle: eigene Darstellung

5.1.2 Benachrichtigungen

Beschreibung: Der Benachrichtigungsbereich dient dazu, den Nutzer über bevorstehende Fälligkeitstermine oder wichtige Aufgaben zu informieren.

Bestandteile:
Benachrichtigungsfenster: Pop-up-Fenster oder eine seitliche Anzeige innerhalb der App, in der alle fälligen Aufgaben dargestellt werden.

Benachrichtigungseinstellungen: In den Einstellungen der Anwendung kann der Nutzer entscheiden, ob und ab welchem Zeitraum zur Fälligkeit einer Aufgabe er Benachrichtigungen erhalten möchte.

5.1.3 Einstellungen

Beschreibung: Der Einstellungsbereich ermöglicht es dem Nutzer, seine Präferenzen für die Nutzung der App anzupassen.

Bestandteile:
Benachrichtigungseinstellungen: Der Nutzer kann entscheiden, ob Benachrichtigungen aktiviert oder deaktiviert werden sollen, und kann den Zeitraum der Benachrichtigung wählen.

Standardprioritäten einstellen: Der Nutzer kann festlegen, welche Standardwerte für Importance, Urgency, und Fitness gesetzt werden sollen, wenn er eine neue Aufgabe erstellt. Dies bietet eine personalisierte Anpassung an seine Arbeitsweise.

Aufgabenarchivierung: Der Nutzer kann Aufgaben, die erledigt sind, archivieren, anstatt sie zu löschen. Standardmäßig ist diese Funktion aktiviert.

5.1.4 Archiv

Beschreibung: Der Archiv-Reiter bietet den Nutzern die Möglichkeit, archivierte Aufgaben zu sehen. Dies erleichtert den Überblick über bereits erledigte Arbeiten und ermöglicht es dem Nutzer, auf alte Aufgaben zuzugreifen, wenn diese erneut relevant werden oder als Referenz dienen sollen.

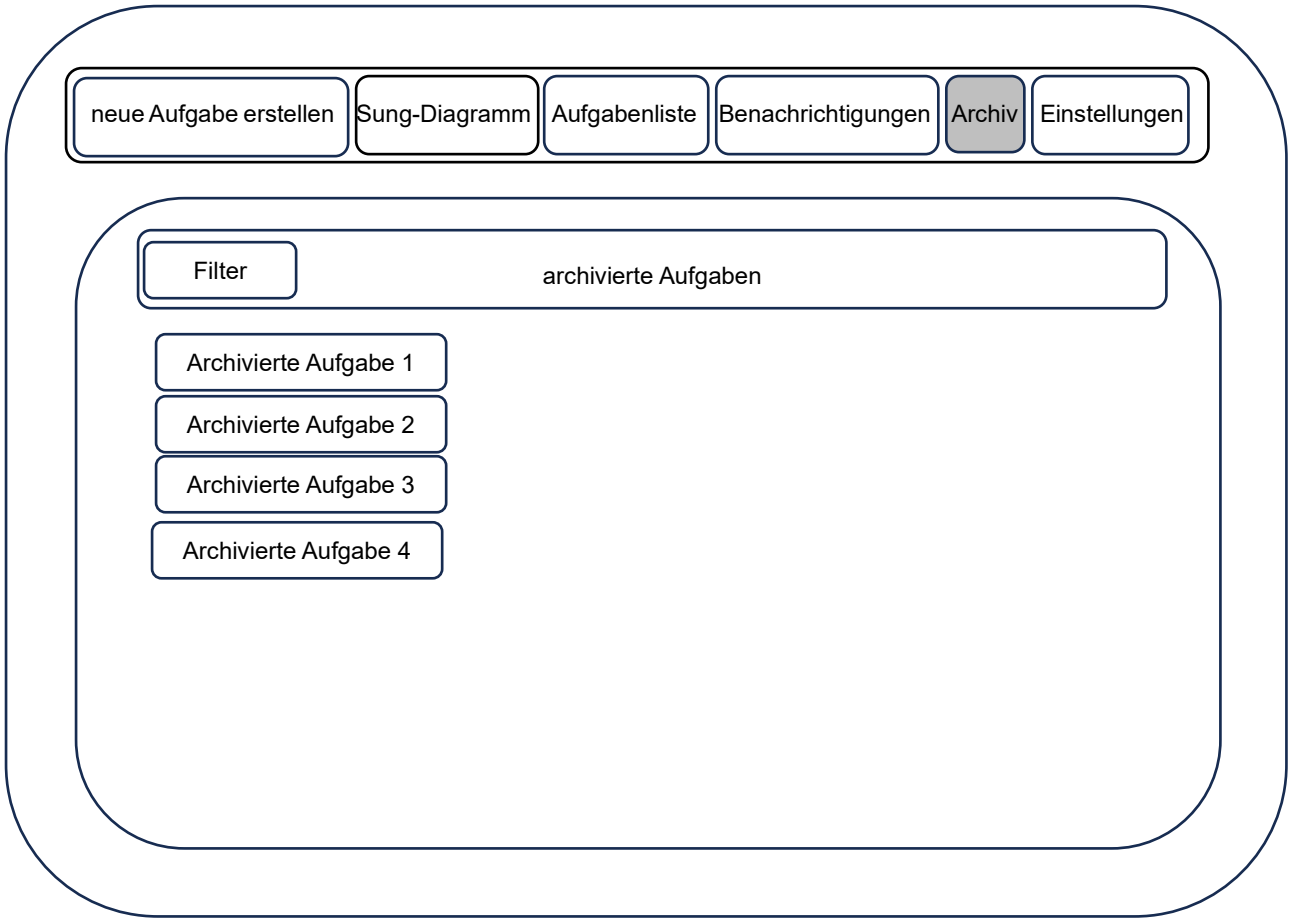
Bestandteile:

Archivierte Aufgabenliste: Eine Übersicht der archivierten Aufgaben. Die Liste enthält grundlegende Informationen wie Titel, Fälligkeitsdatum und Erstellungsdatum.

Reaktivieren von Aufgaben: Der Nutzer hat die Möglichkeit, archivierte Aufgaben wieder zu reaktivieren, sodass diese erneut in die Hauptansicht aufgenommen werden und ggf. neu priorisiert werden können.

Such- und Filterfunktion: Nutzer können nach archivierten Aufgaben suchen und diese nach bestimmten Kriterien wie Priorität, Fälligkeitsdatum oder Kategorie filtern.

Abb. 5 – Skizze – Archivierte Aufgaben



Quelle: eigene Darstellung

5.2 Beschreibung der Dialogflüsse

Der Dialogfluss beschreibt die Navigation und die Interaktion des Nutzers mit der Anwendung. Der Fokus liegt auf einer intuitiven Bedienung, um die Nutzung so einfach wie möglich zu gestalten. Im Folgenden werden die wichtigsten Dialogflüsse der Anwendung beschrieben:

5.2.1 Aufgabe erstellen

Der Nutzer klickt in der Navigationsleiste auf „Neue Aufgabe erstellen“. Daraufhin öffnet sich der Erstellen-/(Bearbeiten)-Dialog, in dem der Nutzer grundlegende Informationen wie den Titel, eine Beschreibung (optional), das Fälligkeitsdatum und die Prioritäten (Importance, Urgency und Fitness) der neuen Aufgabe eingeben kann. Sobald der Nutzer die Informationen eingegeben hat, kann er die Aufgabe speichern, wodurch sie in die Hauptansicht übernommen und im Sung-Diagramm oder der Aufgabenliste angezeigt wird. Der Nutzer hat auch die Möglichkeit, den Prozess abzubrechen, wodurch keine Aufgabe erstellt wird.

5.2.2 Aufgabe bearbeiten

In der Sung-Diagramm- und Aufgabenliste-Ansicht kann der Nutzer eine bestehende Aufgabe auswählen, um sie zu bearbeiten. Dabei wird der (Erstellen-)/Bearbeiten-Dialog geöffnet, in dem alle

Details der Aufgabe (Titel, Beschreibung, Fälligkeitsdatum, Prioritäten) geändert werden können. Nach den Änderungen kann der Nutzer die geänderte Aufgabe speichern, wodurch sie direkt aktualisiert wird. Alternativ kann der Bearbeitungsvorgang abgebrochen werden, um die Änderungen zu verwerfen.

5.2.3 Aufgabe priorisieren

In der Hauptansicht, die das Sung-Diagramm visualisiert, kann der Nutzer eine Aufgabe per Drag-and-Drop in die gewünschte Kategorie verschieben. Dadurch wird die Aufgabe automatisch entsprechend der neuen Priorität (Wichtigkeit, Dringlichkeit, Fitness) klassifiziert. Aufgaben, bei denen alle Prioritäten auf „Low“ gesetzt sind, werden in die spezielle „Low“-Spalte am Rand des Sung-Diagramms verschoben, um sie gesondert darzustellen. Diese Funktion kann mit Maus oder per Touch-Geste erfolgen.

5.2.4 Aufgabenarchivierung

Sobald eine Aufgabe erledigt ist, kann der Nutzer sie als abgeschlossen markieren. Die Aufgabe wird dann automatisch in das Archiv verschoben und nicht mehr in der Hauptansicht angezeigt (wenn in den Einstellungen die Archivierung aktiviert ist). Im Archiv-Reiter hat der Nutzer eine Übersicht über alle archivierten Aufgaben und kann diese bei Bedarf reaktivieren, sodass sie wieder in die Hauptansicht aufgenommen und weiterbearbeitet werden können.

5.2.5 Aufgaben suchen und filtern

Der Nutzer kann in der Aufgabenliste eine Such- und Filterfunktion verwenden, um bestimmte Aufgaben schnell zu finden. Die Filterung kann nach verschiedenen Kriterien wie Fälligkeitsdatum, Priorität, Kategorie oder Status (Offen, In Bearbeitung, Erledigt) erfolgen. Gespeicherte Filtereinstellungen bleiben erhalten und werden beim nächsten Start der Anwendung wieder übernommen, um dem Nutzer eine konsistente und personalisierte Nutzung zu ermöglichen.

5.2.6 Benachrichtigung ansehen

Wenn eine Aufgabe das Fälligkeitsdatum erreicht, öffnet sich ein Benachrichtigungsfenster, das den Nutzer rechtzeitig informiert. Der Nutzer kann die Benachrichtigung quittieren, um die Aufgabe als zur Kenntnis genommen zu markieren, oder direkt zur Aufgabenbearbeitung wechseln, um entsprechende Anpassungen vorzunehmen. Der Zeitraum, ab dem eine Benachrichtigung erfolgen soll, kann in den Einstellungen der Anwendung angepasst werden.

5.2.7 Einstellungen anpassen

Der Nutzer kann den Einstellungsbereich über die Navigationsleiste öffnen. Dort können Benachrichtigungen aktiviert oder deaktiviert sowie der Zeitraum für die Benachrichtigung konfiguriert

werden. Zudem kann der Nutzer die Standardprioritäten für neu erstellte Aufgaben festlegen und die Aufgabenarchivierung aktivieren, um Aufgaben nach Abschluss zu archivieren, statt sie zu löschen.

5.3 Eingabevalidierung

Die Eingabevalidierung stellt sicher, dass alle notwendigen Daten korrekt und vollständig vom Nutzer eingegeben werden, um die Integrität der Aufgabenverwaltung zu gewährleisten. Die folgenden Regeln gelten für die Eingabevalidierung:

Titel und Fälligkeitsdatum: Diese Felder sind Pflichtfelder und müssen vor dem Speichern ausgefüllt sein. Eine leere Eingabe führt zu einer Fehlermeldung, und der Nutzer wird aufgefordert, die entsprechenden Informationen zu ergänzen.

Prioritätenauswahl: Standardwerte werden gesetzt (alle auf "Low" oder gemäß den gewählten Einstellungen), um zu verhindern, dass diese Felder leer bleiben. Der Nutzer hat jedoch die Möglichkeit, diese Werte manuell zu ändern.

Datumseingabe: Das Fälligkeitsdatum muss in der Zukunft liegen. Eine Prüfung stellt sicher, dass keine abgelaufenen Termine gesetzt werden. Falls der Nutzer ein Datum in der Vergangenheit wählt, wird eine Fehlermeldung angezeigt, und der Nutzer muss ein gültiges Fälligkeitsdatum eingeben.

Textlänge: Der Titel der Aufgabe darf eine bestimmte Länge (z.B. 50 Zeichen) nicht überschreiten, um sicherzustellen, dass die Übersichtlichkeit in der Aufgabenliste erhalten bleibt.

Beschreibung: Das Beschreibungsfeld ist optional, aber wenn es ausgefüllt wird, darf die Länge der Beschreibung einen bestimmten Maximalwert nicht überschreiten (z.B. 500 Zeichen), um die Performance und Lesbarkeit zu gewährleisten.

Verhinderung von SQL-Injections: Alle Nutzereingaben, die zur Datenbankkommunikation verwendet werden, werden mittels Prepared Statements behandelt, um die Gefahr von SQL-Injection-Angriffen zu vermeiden.

Pufferüberläufe verhindern: Eingaben des Nutzers, insbesondere für Felder wie Titel oder Beschreibung, werden auf eine maximal erlaubte Länge begrenzt. Die Anwendung führt eine Überprüfung der Eingabelängen durch, um sicherzustellen, dass die festgelegten Grenzwerte nicht überschritten werden und es zu keinem Buffer Overflow kommt.

Literaturverzeichnis

Schatten, A., Biffl, S., Demolsky, M., Gostischa-Franta, E., Östreicher, T., & Winkler, D. (2010). *Best Practice Software-Engineering: Eine Praxiserprobte Zusammenstellung Von Komponenten-orientierten Konzepten, Methoden und Werkzeugen*. Spektrum Akademischer Verlag GmbH. http://ebookcentral.proquest.com/lib/badhonnef/detail.action?docID=511284

Sommerville, I. (2012). *Software Engineering*. Pearson Deutschland GmbH. http://ebookcentral.proquest.com/lib/badhonnef/detail.action?docID=5133710

Portfolio – Phase 2

IU Internationale Hochschule Fernstudium

Studiengang: Master of Science Informatik

Kurs: Projekt – Software Engineering (DLMCSPSE01_D)

Architekturdokument

Windows Anwendung:
Priorisierungsmatrix nach dem Sung-Diagramm

Gregor Hubmann

Matrikelnummer: 9196398

Hornstraße 185

5423 St. Koloman

Abgabedatum (Phase 2): 24.11.2024

Änderungen nach Phase 1:

Dokument wurde in Phase 2 neu erstellt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Das gegenständliche Architekturdokument beschreibt die Struktur und das technische Design der Anwendung. Ziel ist es, eine klare Übersicht über die Hauptkomponenten, ihre Rollen und Verantwortlichkeiten sowie ihre Interaktionen zu geben. Das Dokument umfasst eine Technologieübersicht, die Wahl und Begründung der verwendeten Technologien und Werkzeuge erklärt, sowie eine Architekturübersicht, die die grundlegende Struktur und die Beziehungen zwischen den Komponenten darstellt. Anhand eines UML-Klassendiagramms und eines Sequenzdiagramms wird das Zusammenspiel der Klassen und der Ablauf eines zentralen Systemvorgangs visualisiert.

1. Technologieübersicht

Für die Umsetzung der Anwendung, wurde Python als Programmiersprache ausgewählt. Die Wahl der Technologie basiert auf der Verfügbarkeit geeigneter Bibliotheken zur Realisierung der Anforderungen und auf einer einfachen Implementierung und Wartbarkeit der Anwendung.

1.1 Programmiersprache

Die Anwendung wird in *Python* entwickelt. Python wurde aufgrund seiner vielseitigen Bibliotheken und seiner guten Lesbarkeit ausgewählt. Es ermöglicht eine schnelle und effiziente Umsetzung der Anforderungen und erleichtert zukünftige Anpassungen. Zudem ist Python gut geeignet für den Desktop-Bereich und bietet umfangreiche Unterstützung für GUI- und Datenbankanwendungen.

1.2 GUI-Framework

Für die grafische Benutzeroberfläche wird *tkinter* verwendet. tkinter ist ein leichtgewichtiges und in Python integriertes Framework, das grundlegende Funktionen für die Erstellung von Benutzeroberflächen bietet. Die verfügbaren Widgets und Layout-Optionen ermöglichen die Umsetzung der Drag-and-Drop-Funktionalitäten sowie die Gestaltung einer benutzerfreundlichen Oberfläche.

1.3 Datenbank

Die Anwendungsdaten werden mit *SQLite* gespeichert. SQLite ist eine leichtgewichtige und eingebettete Datenbank, die sich ideal für Desktop-Anwendungen eignet. Die Integration mit Python ist unkompliziert, und SQLite ermöglicht eine persistente Datenspeicherung ohne zusätzliche Konfigurationsaufwände. Diese Wahl unterstützt die geplante, lokale Datennutzung und erfüllt die Anforderungen an die Datenverwaltung innerhalb der Anwendung.

1.4 Benachrichtigungen

Die Benachrichtigungsfunktionalität wird durch die *NotificationManager*-Klasse bereitgestellt, die basierend auf Benutzereinstellungen und Aufgabenfälligkeit Benachrichtigungen plant. Anstatt Push-Benachrichtigungen über eine externe Bibliothek wie plyer zu senden, werden die geplanten Benachrichtigungen intern verwaltet und über GUI-Elemente (z. B. Messagebox) ausgegeben. Dies

ermöglicht eine effiziente und einfache Erinnerung an bevorstehende Aufgaben, ohne zusätzliche Abhängigkeiten einzuführen.

1.5 Versionsverwaltung

Die Versionsverwaltung erfolgt mit *GitHub*, was die Kontrolle und Nachverfolgbarkeit der Entwicklungsfortschritte sicherstellt. Um den Entwicklungsprozess einfach zu halten, wird auf ein festes Workflow-Modell wie Git-Flow verzichtet, da der zusätzliche Aufwand für ein Einzelprojekt nicht erforderlich ist. Stattdessen erfolgt eine grundlegende Versionierung, die die Änderungen und Fortschritte im Projekt dokumentiert.

1.6 Tests und Qualitätssicherung

Die Qualitätssicherung wird durch *pytest* unterstützt. Pytest dient als Test-Framework und ermöglicht die Automatisierung von Unit- und Integrationstests, was die zuverlässige Implementierung der funktionalen Anforderungen sicherstellt.

1.7 Paketverwaltung und Virtualisierung

Zur Verwaltung von Abhängigkeiten und zur Sicherstellung der Portabilität der Anwendung wird eine virtuelle Umgebung mit *venv* genutzt. Dies ermöglicht die isolierte Verwaltung von Paketen und minimiert potenzielle Versionskonflikte bei der Nutzung der Anwendung auf verschiedenen Systemen.

2. Architekturübersicht

Die Anwendung folgt einer Schichtenarchitektur, um eine klare Trennung der verschiedenen Funktionsbereiche zu gewährleisten (Schatten et al., 2010, S. 211). Diese Architektur erleichtert sowohl die Implementierung, Wartung als auch die Erweiterbarkeit der Anwendung, da jede Schicht spezifische Aufgaben erfüllt und somit unabhängig von den anderen Schichten weiterentwickelt oder angepasst werden kann. Die Architektur besteht aus den folgenden drei Hauptschichten:

2.1 Präsentationsschicht

Diese Schicht bildet die Benutzeroberfläche (GUI) und wird durch tkinter realisiert. Sie stellt alle Elemente bereit, die der Benutzer zur Interaktion mit der Anwendung benötigt, einschließlich der Drag-and-Drop-Funktionalität im Sung-Diagramm, Filtermöglichkeiten und Benachrichtigungen. Die Präsentationsschicht kommuniziert ausschließlich mit der Anwendungsschicht und leitet Benutzereingaben weiter, ohne die Geschäftslogik direkt zu beinhalten. Die strikte Trennung der Präsentationslogik verbessert die Flexibilität bei der Gestaltung und Anpassung der Benutzeroberfläche.

2.2 Anwendungsschicht

Die Anwendungsschicht enthält die Geschäftslogik der Anwendung. Sie steuert den Datenfluss zwischen der Präsentations- und der Datenzugriffsschicht und ist verantwortlich für alle Funktionen der

Aufgaben-Priorisierung und -Verwaltung. Die Geschäftslogik bestimmt, wie Benutzeraktionen (z.B. das Priorisieren oder Archivieren von Aufgaben) verarbeitet werden. Diese Schicht ist zentral für die Umsetzung der funktionalen Anforderungen, wie die Einteilung der Aufgaben in Kategorien des Sung-Diagramms.

2.3 Datenzugriffsschicht

Die Datenzugriffsschicht ist für den Zugriff auf die SQLite-Datenbank verantwortlich. Sie umfasst alle Operationen zur Speicherung, Abfrage und Aktualisierung von Daten. Diese Schicht abstrahiert den direkten Datenbankzugriff und stellt der Anwendungsschicht klar definierte Schnittstellen zur Verfügung, wodurch die Datenintegrität und -sicherheit gewährleistet wird. Änderungen an der Datenstruktur oder an der Implementierung der Datenbankzugriffe können auf diese Weise ohne Einfluss auf die Geschäftslogik erfolgen.

2.4 Begründung der Schichtenarchitektur

Die Schichtenarchitektur wurde aufgrund der folgenden Vorteile gewählt:

Modularität und Wartbarkeit: Jede Schicht ist klar voneinander getrennt und erfüllt spezifische Aufgaben (Schatten et al., 2010, S. 211). Dies vereinfacht und strukturiert die Implementierung, Wartung und Anpassung, da Änderungen in einer Schicht keine direkten Auswirkungen auf andere Schichten haben.

Wiederverwendbarkeit und Erweiterbarkeit: Die Struktur ermöglicht die einfache Erweiterung der Anwendung, z.B. durch das Hinzufügen neuer Funktionen in der Anwendungsschicht oder Anpassungen an der Benutzeroberfläche (Schatten et al., 2010, S. 211).

Testbarkeit: Die Schichtenarchitektur erleichtert das Testen, da jede Schicht isoliert getestet werden kann (Schatten et al., 2010, S. 211). Insbesondere die Geschäftslogik in der Anwendungsschicht lässt sich unabhängig von der Benutzeroberfläche und der Datenzugriffsschicht testen.

3. Struktur

In diesem Abschnitt wird die Struktur der Anwendung beschrieben. Die Hauptkomponenten und ihre Rollen werden zunächst einzeln vorgestellt, gefolgt von ihren Attributen und Methoden. Anschließend werden die Beziehungen und Abhängigkeiten zwischen den Komponenten definiert, um die Interaktionen und die Datenflüsse in der Anwendung zu verdeutlichen. Der GUIController wird als zentrale Schnittstelle für die Benutzerinteraktion beschrieben und erklärt, wie er die verschiedenen Geschäftsobjekte steuert. Abschließend veranschaulicht ein UML-Klassendiagramm die Struktur und Beziehungen aller Komponenten und bietet eine visuelle Übersicht der gesamten Anwendungsarchitektur.

3.1 Hauptkomponenten - Übersicht

Die Hauptkomponenten der Anwendung strukturieren die Aufgabenverwaltung, Benutzereinstellungen und die Benutzeroberfläche in klar definierten Modulen, die die Funktionalität der Anwendung vollständig abdecken. Für eine Übersicht über alle Komponenten wird auf das UML-Klassendiagramm verwiesen.

Zu den Hauptkomponenten zählen:

Task: Verwaltung der einzelnen Aufgaben.

User: Verwaltung der Benutzerinformationen.

NotificationManager: Steuerung und Versand von Benachrichtigungen.

GUIController: Vermittlung der Interaktionen zwischen Benutzeroberfläche und Geschäftslogik.

ArchiveManager: Verwaltung des Archivs für erledigte Aufgaben.

SettingsManager: Speicherung und Anpassung der Benutzereinstellungen.

3.2 Hauptkomponenten – Rollen und Attribute

3.2.1 Task

Repräsentiert eine einzelne Aufgabe, die der Benutzer erstellt und im Sung-Diagramm priorisiert.

Attribute:

- title (String): Titel der Aufgabe.
- description (String, optional): Detaillierte Beschreibung der Aufgabe.
- due_date (Date): Fälligkeitsdatum der Aufgabe.
- importance (Priority Enum: HIGH, LOW): Priorität der Aufgabe nach Wichtigkeit.
- urgency (Priority Enum: HIGH, LOW): Priorität der Aufgabe nach Dringlichkeit.
- fitness (Priority Enum: HIGH, LOW): Einschätzung der eigenen Fähigkeiten zur Erledigung der Aufgabe.
- status (Status Enum: OPEN, IN_PROGRESS, COMPLETED): Aktueller Bearbeitungsstand der Aufgabe.
- completed_date (Date, optional): Das Datum, an dem die Aufgabe als erledigt markiert wurde. Es wird verwendet, um Aufgaben für automatische Archivierung oder Löschung zu erkennen.
- id (int, optional): Eindeutige Identifikation der Aufgabe.

Methoden:

- __init__
  Initialisiert eine neue Aufgabe mit den angegebenen Attributen.
  Standardwerte: status=Status.OPEN, completed_date=None, task_id=None.

- edit_task
  Ermöglicht das Bearbeiten der Attribute einer Aufgabe.
  Parameter:
    title (String, optional): Neuer Titel der Aufgabe.
    due_date (Date, optional): Neues Fälligkeitsdatum.
    importance (Priority, optional): Neue Wichtigkeit.
    urgency (Priority, optional): Neue Dringlichkeit.
    fitness (Priority, optional): Neue Fitness-Einschätzung.
    description (String, optional): Neue Beschreibung.
  Überprüft, ob neue Werte übergeben wurden, und aktualisiert die entsprechenden Attribute.
- mark_as_completed
  Markiert eine Aufgabe als abgeschlossen, indem der status auf Status.COMPLETED gesetzt wird. Setzt das completed_date auf das aktuelle Datum (date.today()).

Funktion: Diese Klasse stellt die Grunddatenstruktur für Aufgaben dar und speichert alle relevanten Details für die Priorisierung und Verwaltung im Sung-Diagramm. Sie unterstützt Bearbeitung und Statusänderungen der Aufgaben.

3.2.2 User

Repräsentiert den Benutzer, der die Anwendung nutzt.

Attribute:

- username (String): Benutzername des Nutzers.
- password_hash (String): Gespeichertes Passwort des Nutzers als Hash-Wert.

Methoden:

- __init__
  Initialisiert eine neue Benutzerinstanz mit einem Benutzernamen und einem Passwort.
  Das Passwort wird direkt nach SHA-256 gehasht und als password_hash gespeichert.
- hash_password
  Hashing-Methode, die das Passwort mit dem SHA-256-Algorithmus sicher verschlüsselt.
  Parameter:
    password (*String*): Das Klartext-Passwort.
  Rückgabe:
    SHA-256-Hash des Passworts als Hex-String.
- check_password
  Prüft, ob ein eingegebenes Passwort mit dem gespeicherten Hash übereinstimmt.
  Parameter:
    password (*String*): Das zu überprüfende Klartext-Passwort.
  Rückgabe:

True, wenn das eingegebene Passwort dem gespeicherten Hash entspricht, andernfalls *False*.

Funktion: Speichert Benutzerinformationen, authentifiziert den Benutzer und verwaltet Benutzerdaten.

3.2.3 NotificationManager

Verarbeitet die Benachrichtigungen für den Benutzer, um auf bevorstehende Fälligkeiten aufmerksam zu machen.

Attribute:

- settings_manager (*SettingsManager*): Verwalter der Benutzereinstellungen, der unter anderem die Benachrichtigungsintervalle und Aktivierungsstatus speichert.

Funktion: Der NotificationManager durchsucht die Aufgabenliste und nutzt die Benutzereinstellungen, um Benachrichtigungen für bevorstehende Fälligkeiten basierend auf dem konfigurierten Intervall zu planen. Dabei wird überprüft, ob Benachrichtigungen aktiviert sind und ob eine Aufgabe innerhalb des festgelegten Zeitraums fällig wird.

Methoden:

- schedule_notifications(tasks):
  Beschreibung: Plant Benachrichtigungen für Aufgaben basierend auf den Benutzereinstellungen.
  Parameter: tasks (Liste von Task-Objekten), die überprüft werden sollen.
  Rückgabewert: Eine Liste von Aufgaben, für die Benachrichtigungen anstehen.

3.2.4 GUIController

Verbindet die Benutzeroberfläche mit der Geschäftslogik, verwaltet Benutzeraktionen und aktualisiert die grafische Darstellung. Er stellt die Hauptschnittstelle zwischen dem Benutzer und den zugrunde liegenden Logik- und Datenzugriffsschichten dar.

Attribute:

- root (tk.Tk): Hauptfenster der tkinter-Benutzeroberfläche.
- current_user (str): Der Benutzername des aktuell angemeldeten Benutzers.
- db_path (str): Der Pfad zur SQLite-Datenbank.
- tasks (List[Task]): Eine Liste aller geladenen Aufgaben.
- task_elements (Dict[int, tk.Canvas]): Eine Zuordnung zwischen Aufgaben-IDs und ihren grafischen Elementen im Venn-Diagramm.
- selected_task (Dict): Die aktuell ausgewählte Aufgabe mit zugehörigem grafischen Element.
- settings_manager (SettingsManager): Verwaltung der Benutzereinstellungen.
- archive_manager (ArchiveManager): Verwaltung archivierter Aufgaben.

- notification_manager (NotificationManager): Verwaltung und Planung von Benachrichtigungen.
- filter_controller (FilterController): Verwaltung von Filterfunktionen in der Aufgabenliste.
- drag_drop_handler (DragDropHandler): Steuerung der Drag-and-Drop-Funktionalität im *Venn-Diagramm.*

Methoden:

- Erstellen und Aktualisieren der GUI:
  create_widgets(): Erstellt die GUI-Elemente, einschließlich des Venn-Diagramms, der Schaltflächen und der Listen.
  draw_venn_diagram(): Zeichnet das Venn-Diagramm, das die Aufgaben nach Prioritäten organisiert.
  update_task_venn_diagram(): Aktualisiert das Venn-Diagramm basierend auf den aktuellen Aufgaben und bindet Drag-and-Drop-Events.
- Aufgabenmanagement:
  add_task(): Öffnet den TaskEditor, um eine neue Aufgabe hinzuzufügen.
  edit_task(): Öffnet den TaskEditor für die Bearbeitung der ausgewählten Aufgabe.
  delete_task(): Löscht die ausgewählte Aufgabe aus der Datenbank und der grafischen Darstellung.
  mark_task_completed(): Markiert eine ausgewählte Aufgabe als erledigt und archiviert sie bei Bedarf.
  mark_task_open(): Markiert eine abgeschlossene Aufgabe erneut als offen.
- Archivverwaltung:
  archive_selected_task(task_to_archive=None): Archiviert die ausgewählte Aufgabe, falls sie abgeschlossen ist.
  show_archive(): Öffnet den ArchiveViewer, um archivierte Aufgaben anzuzeigen.
- Filter- und Task-Anzeige:
  load_tasks(filters=None): Lädt Aufgaben aus der Datenbank basierend auf den angegebenen Filtern.
  update_task_listbox(): Aktualisiert die Listenansicht für Aufgaben mit niedriger Priorität.
  low_listbox_select(event): Behandelt die Auswahl einer Aufgabe in der Liste für Aufgaben mit niedriger Priorität.
- Benachrichtigungen:
  schedule_notifications(): Plant Benachrichtigungen für bevorstehende Fälligkeiten und zeigt diese an.
- Drag-and-Drop-Handling:
  drag_or_select_task(event, task_id): Bestimmt, ob eine Aufgabe ausgewählt oder verschoben wird.

_handle_click_or_drag(event, task_id): Finalisiert die Entscheidung zwischen Auswahl und Drag-and-Drop.

Funktion: Agiert als zentrale Schnittstelle zwischen der Benutzeroberfläche und der Geschäftslogik. Er ermöglicht das Erstellen, Bearbeiten und Löschen von Aufgaben, die Verwaltung archivierter Aufgaben und die Planung von Benachrichtigungen. Zusätzlich stellt er sicher, dass Drag-and-Drop und andere Benutzeraktionen korrekt in die Anwendung integriert werden.

3.2.5 ArchiveManage

Verwaltet archivierte Aufgaben, die zuvor als abgeschlossen markiert wurden. Er ermöglicht das Speichern, Abrufen und gegebenenfalls das automatische Löschen dieser Aufgaben, um die Übersichtlichkeit der aktiven Aufgaben zu gewährleisten.

Attribute:

- db_path (str): Der Pfad zur SQLite-Datenbank, in der archivierte Aufgaben gespeichert werden.

Methoden:

- Tabellenverwaltung:
  _create_archived_tasks_table(): Erstellt die Tabelle archived_tasks in der SQLite-Datenbank, falls sie nicht existiert. Diese Tabelle speichert abgeschlossene Aufgaben.
- Aufgabenarchivierung:
  archive_task(task): Verschiebt eine abgeschlossene Aufgabe in das Archiv, indem sie in der Tabelle archived_tasks gespeichert wird.
  Anforderung: Die Aufgabe muss den Status COMPLETED haben.
  Fehlerbehandlung: Löst eine ValueError aus, wenn die Aufgabe nicht abgeschlossen ist.
- Automatische Archivierung:
  auto_archive_task(task, days_until_archive): Archiviert automatisch Aufgaben, die seit einer bestimmten Anzahl von Tagen abgeschlossen sind.
- Automatische Löschung:
  auto_delete_task(task, days_until_delete): Löscht Aufgaben aus dem Archiv, die eine festgelegte Zeitspanne überschritten haben.

Funktion: Strukturiert archivierte Aufgaben und hält das aktive Aufgabenmanagement übersichtlich. Er bietet Funktionen zur automatischen Archivierung und Löschung, die helfen, die Datenbank zu organisieren und Speicherplatz zu verwalten. Dies ermöglicht dem Nutzer, abgeschlossene Aufgaben bei Bedarf wieder aufzurufen oder automatisch zu entfernen.

3.2.6 SettingsManager

Verwaltet die Benutzereinstellungen der Anwendung, einschließlich Standardprioritäten, Benachrichtigungsoptionen, automatischer Archivierung und Löschung von Aufgaben. Diese Klasse ermöglicht die Anpassung und Speicherung von Einstellungen für eine personalisierte Nutzungserfahrung.

Attribute:

- db_path (str): Der Pfad zur SQLite-Datenbank, in der die Einstellungen gespeichert werden.*Methoden:*

Methoden:

- Tabelleninitialisierung:
  _initialize_settings_table(): Stellt sicher, dass die Tabelle settings in der Datenbank existiert. Falls nicht, wird sie erstellt.
- Einstellungsverwaltung:
  save_settings(notification_interval, auto_archive, auto_delete, notifications_enabled, default_priorities): Speichert oder aktualisiert die Benutzereinstellungen in der Datenbank.
  Parameter: Umfasst Benachrichtigungsintervalle, automatische Archivierungs- und Löschoptionen sowie Standardprioritäten.
  get_settings(): Ruft die aktuellen Benutzereinstellungen aus der Datenbank ab und gibt sie als Dictionary zurück.
  Rückgabe: Standardwerte, falls keine Einstellungen vorhanden sind.
- Prioritätseinstellungen:
  update_default_priorities(priorities): Aktualisiert die Standardprioritäten für neue Aufgaben.
  Parameter: Dictionary mit den Prioritäten für importance, urgency und fitness.
- Benachrichtigungseinstellungen:
  update_notifications_enabled(enabled): Aktiviert oder deaktiviert Benachrichtigungen.
  Parameter: Boolean-Wert, um Benachrichtigungen ein- oder auszuschalten.
  update_notification_interval(interval): Aktualisiert das Benachrichtigungsintervall in Tagen.
  Parameter: Ganzzahliger Wert für die Anzahl der Tage.
- Archivierungs- und Löschoptionen:
  update_auto_archive_setting(setting): Aktiviert oder deaktiviert die automatische Archivierung.
  Parameter: Boolean-Wert.
  update_auto_delete_setting(setting): Aktiviert oder deaktiviert die automatische Löschung.
  Parameter: Boolean-Wert.

Funktion: Ermöglicht es, die Anwendung an die Präferenzen der Benutzer anzupassen. Er stellt sicher, dass Benachrichtigungen, Prioritäten und automatische Prozesse wie Archivierung und

Löschung flexibel verwaltet werden können. Die Klasse abstrahiert den Zugriff auf die Datenbank und bietet eine klare Trennung zwischen Geschäftslogik und Datenpersistenz.

3.3 Hauptkomponenten – Beziehungen und Abhängigkeiten

Task und User

Beziehung: Aggregation

Beschreibung: Eine User-Instanz kann mehrere Task-Instanzen erstellen und verwalten. Der Benutzer ist der Eigentümer der Aufgaben und kann diese bearbeiten, priorisieren oder löschen. Durch die Aggregation wird verdeutlicht, dass Aufgaben ohne den Benutzer existieren können, jedoch von ihm verwaltet werden.

Rolle: Die Aggregation zeigt, dass ein Benutzer mehrere Aufgaben besitzen und bearbeiten kann, wobei jede Aufgabe nur einem Benutzer gehört.

Task und ArchiveManager

Beziehung: Assoziation

Beschreibung: Der ArchiveManager interagiert mit Task, um erledigte Aufgaben aus der Hauptliste zu entfernen und in der Datenbanktabelle archived_tasks zu speichern. Archivierte Aufgaben können bei Bedarf wiederhergestellt werden.

Rolle: Der ArchiveManager strukturiert abgeschlossene Aufgaben, um die aktive Aufgabenliste übersichtlich zu halten. Gleichzeitig ermöglicht er es, Aufgaben zu einem späteren Zeitpunkt erneut zu aktivieren.

NotificationManager und Task

Beziehung: Assoziation

Beschreibung: Der NotificationManager prüft die Aufgabenliste auf bevorstehende Fälligkeiten, basierend auf dem due_date jeder Aufgabe. Benachrichtigungen werden erstellt, wenn eine Aufgabe innerhalb des konfigurierten Intervalls fällig wird.

Rolle: Diese lose Verbindung erlaubt es, Benutzer rechtzeitig über anstehende Aufgaben zu informieren, ohne dass eine direkte Kopplung zwischen den Klassen erforderlich ist.

GUIController und die Benutzerinteraktion mit Task, NotificationManager, ArchiveManager und SettingsManager

Beziehung: Assoziation

Beschreibung: Der GUIController agiert als zentrale Vermittlungsschicht zwischen der Benutzeroberfläche und den Geschäftsobjekten. Er interagiert mit Task, NotificationManager,

ArchiveManager und SettingsManager, um Benutzeraktionen wie das Erstellen, Bearbeiten, Priorisieren, Archivieren und Löschen von Aufgaben sowie das Anpassen von Einstellungen und Anzeigen von Benachrichtigungen umzusetzen.

Rolle: Der GUIController sorgt dafür, dass Benutzeraktionen korrekt in die Geschäftslogik übertragen werden und aktualisiert die grafische Oberfläche entsprechend.

SettingsManager und NotificationManager

Beziehung: Assoziation

Beschreibung: Der SettingsManager speichert Benachrichtigungseinstellungen wie Intervalle und Aktivierungsstatus. Diese Einstellungen werden vom NotificationManager abgerufen, um Benachrichtigungen entsprechend den Benutzerpräferenzen zu planen.

Rolle: Diese lose Verbindung ermöglicht eine flexible Anpassung der Benachrichtigungsoptionen durch den Benutzer.

SettingsManager und ArchiveManager

Beziehung: Assoziation

Beschreibung: Der SettingsManager speichert Benutzerpräferenzen für automatische Archivierung und Löschung von Aufgaben. Der ArchiveManager verwendet diese Einstellungen, um abgeschlossene Aufgaben automatisch zu verwalten.

Rolle: Diese Verbindung ermöglicht es, die Archivierungs- und Löschprozesse gemäß den Benutzerpräferenzen zu automatisieren.

In der Anwendung sind die Hauptkomponenten lose gekoppelt, wodurch eine hohe Flexibilität und Wartbarkeit gewährleistet werden. Die zentrale Schnittstelle für Benutzerinteraktionen ist der GUIController, der als Vermittler zwischen der Benutzeroberfläche und der Geschäftslogik agiert. Die Klassen interagieren gezielt, um Benutzereingaben in die Logik zu überführen und Ergebnisse übersichtlich anzuzeigen. Diese Struktur fördert die Erweiterbarkeit und Modularität der Anwendung.

3.4 GUIController

Der GUIController ist eine zentrale Komponente der Anwendung und dient als Schnittstelle zwischen der Benutzeroberfläche und der Geschäftslogik. Er verarbeitet Benutzeraktionen und sorgt dafür, dass Änderungen korrekt in der Datenbank und der GUI widergespiegelt werden. Während die Klasse viele wichtige Aufgaben übernimmt, ist sie möglicherweise überladen. Eine striktere Trennung zwischen der Präsentations- und der Geschäftslogik wurde nicht konsequent umgesetzt, was in der Projektdokumentation detaillierter analysiert wird.

Aufgaben:

- Erstellen, Bearbeiten und Löschen von Aufgaben: Der GUIController ermöglicht Benutzern, Aufgaben zu erstellen, zu bearbeiten und zu löschen. Diese Funktionen werden über Methoden wie add_task(), edit_task(), und delete_task() bereitgestellt und greifen direkt auf die Datenbank zu.
- Kategorisieren und Priorisieren von Aufgaben: Aufgaben werden in einem Venn-Diagramm basierend auf ihren Prioritätswerten (importance, urgency, fitness) dargestellt. Die Methode update_task_venn_diagram() sorgt dafür, dass die Darstellung der Aufgaben immer aktuell ist.
- Anzeigen von Benachrichtigungen: Über schedule_notifications() wird der Benutzer über bevorstehende Fälligkeiten informiert. Diese Funktion ruft Benachrichtigungsdaten vom NotificationManager ab und zeigt sie in der GUI an.
- Verwalten von archivierten Aufgaben: Aufgaben können über archive_selected_task() archiviert und mit show_archive() angezeigt werden. Die Verwaltung archivierter Aufgaben erfolgt in enger Zusammenarbeit mit dem ArchiveManager.
- Anpassen von Einstellungen: Über die Methode show_settings() können Benutzer Einstellungen wie Benachrichtigungsintervalle oder automatische Archivierung anpassen. Diese Änderungen werden vom SettingsManager verwaltet.
- Drag-and-Drop-Funktionalität: Mit Hilfe des DragDropHandler können Benutzer Aufgaben interaktiv im Venn-Diagramm verschieben. Die Methoden drag_or_select_task() und _handle_click_or_drag() kontrollieren, ob eine Aufgabe verschoben oder ausgewählt wird.

Hinweise zur Struktur

Die GUIController-Klasse ist sehr umfangreich und vereint sowohl Präsentations- als auch Teile der Geschäftslogik. In einer idealen Schichtenarchitektur sollten GUI-Logik und Geschäftslogik strikt getrennt sein, um die Wartbarkeit zu verbessern. Dieser Aspekt wird in der Projektdokumentation ausführlicher diskutiert.

3.5 Interaktion mit den Hauptkomponenten

Task und GUIController

Der GUIController interagiert mit der Task-Klasse, um Benutzeraktionen wie das Erstellen, Bearbeiten und Löschen von Aufgaben zu ermöglichen. Aufgaben werden über die grafische Benutzeroberfläche in das System eingefügt, angepasst und entfernt. Der GUIController ordnet die Aufgaben im Sung-Diagramm anhand ihrer Prioritäten (Wichtigkeit, Dringlichkeit und Fitness) ein und aktualisiert die Darstellung entsprechend.

NotificationManager und GUIController

Der NotificationManager überwacht die Aufgaben auf bevorstehende Fälligkeiten und sendet Benachrichtigungen. Der GUIController empfängt diese Benachrichtigungen und zeigt sie in der Benutzeroberfläche an. Gleichzeitig können Benutzer über den GUIController die Benachrichtigungseinstellungen wie Intervalle und Aktivierungsstatus anpassen. Änderungen werden vom GUIController an den SettingsManager weitergeleitet, der die Konfiguration verwaltet.

SettingsManager und GUIController

Der GUIController ermöglicht es Benutzern, die Anwendungseinstellungen zu ändern, die der SettingsManager verwaltet. Dazu gehören:

- Benachrichtigungseinstellungen (z. B. Intervall, Aktivierung)
- Automatische Archivierung und Löschung: Benutzer können konfigurieren, ob und wann Aufgaben automatisch archiviert oder gelöscht werden sollen.

Der GUIController fungiert hierbei als Vermittler, um Änderungen von der Benutzeroberfläche an den SettingsManager weiterzugeben, welcher diese in der Datenbank speichert.

ArchiveManager und GUIController

Der GUIController stellt Funktionen bereit, mit denen Benutzer Aufgaben archivieren oder archivierte Aufgaben wiederherstellen können. Er bietet:

- Manuelle Archivierung: Benutzer können abgeschlossene Aufgaben direkt ins Archiv verschieben.
- Wiederherstellen von archivierten Aufgaben: Archivierte Aufgaben können bei Bedarf in die aktive Aufgabenliste zurückgeführt werden.
- Automatische Archivierung: Der GUIController zeigt an, wenn diese Funktion aktiviert ist, und überlässt die Umsetzung dem ArchiveManager.

Der ArchiveManager übernimmt die Datenhaltung und Logik für Archivierungsvorgänge, während der GUIController als Benutzeroberflächenschnittstelle fungiert.

Zusammenfassung der Interaktionen

Der GUIController ist die zentrale Schaltstelle für die Benutzerinteraktion und vermittelt zwischen:

- Task: Verwaltung und Darstellung von Aufgaben.
- NotificationManager: Planung und Anzeige von Benachrichtigungen.
- SettingsManager: Anpassung und Speicherung von Benutzereinstellungen.
- ArchiveManager: Archivierung und Wiederherstellung von Aufgaben.

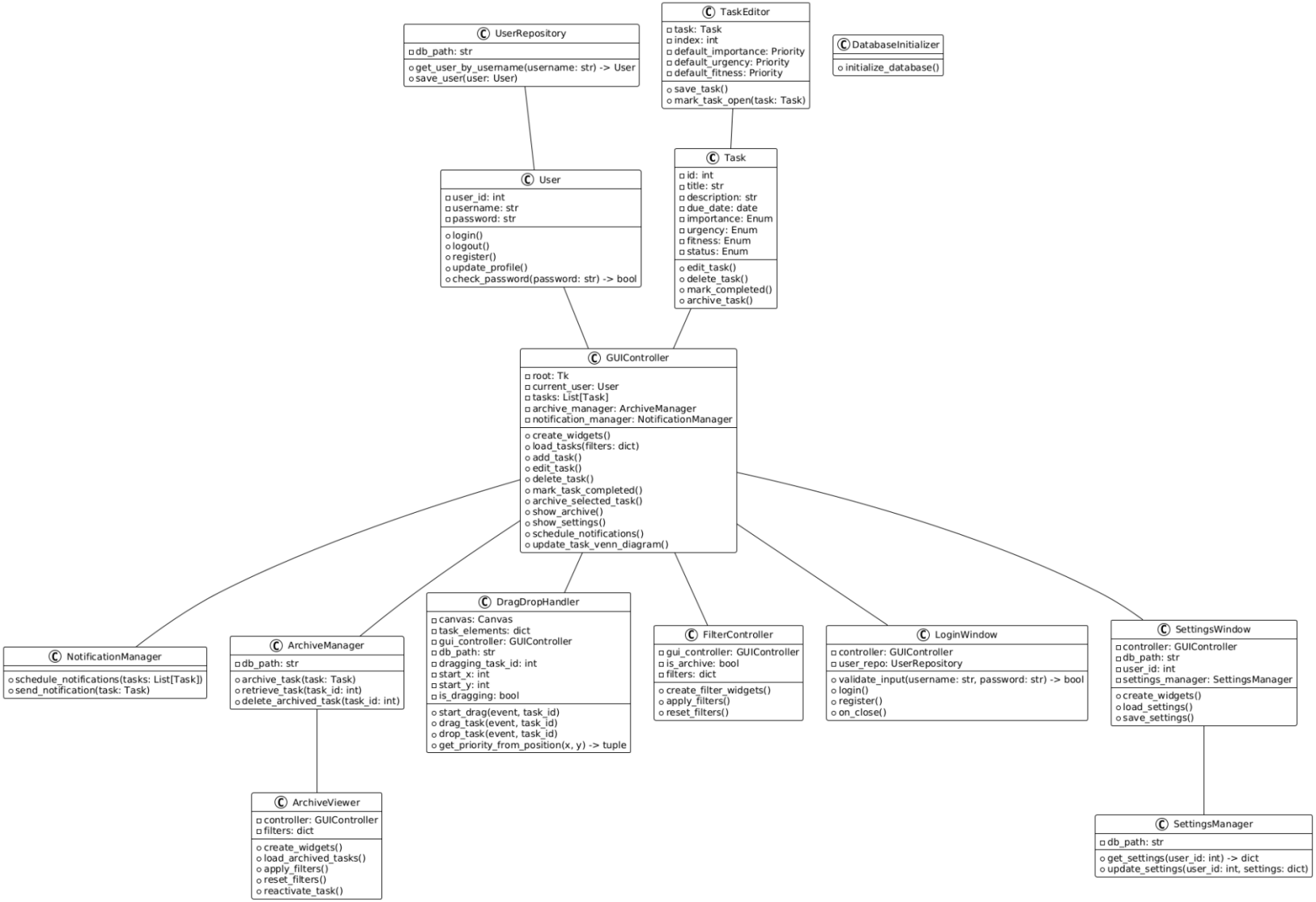
3.6 Übersicht der Benutzeraktionen

Der GUIController fungiert als zentrale Vermittlungsschicht zwischen der Benutzeroberfläche und der Geschäftslogik und gewährleistet folgende Funktionen:

- Aufgabenverwaltung
  Benutzer können über die GUI Aufgaben erstellen, bearbeiten, priorisieren, archivieren und löschen. Der GUIController leitet diese Aktionen an die entsprechenden Klassen (Task, ArchiveManager) weiter, aktualisiert die Benutzeroberfläche und sorgt dafür, dass Änderungen sofort sichtbar sind.
- Benachrichtigungen
  Vom NotificationManager generierte Benachrichtigungen zu bevorstehenden Fälligkeiten werden über den GUIController in der Benutzeroberfläche angezeigt. Benutzer können außerdem über die GUI Benachrichtigungseinstellungen, wie Intervalle und Aktivierungsstatus, anpassen. Diese Änderungen werden an den SettingsManager übermittelt.
- Einstellungen verwalten
  Der GUIController bietet Zugriff auf den SettingsManager, sodass Benutzer wichtige Konfigurationen, wie Standardprioritäten, automatische Archivierung oder Löschung von Aufgaben, direkt in der Benutzeroberfläche anpassen können. Änderungen werden gespeichert und wirken sich sofort auf die Geschäftslogik aus.
- Aktualisierung der Benutzeroberfläche
  Der GUIController stellt sicher, dass die Benutzeroberfläche stets aktuell bleibt, indem sie nach jeder Benutzeraktion oder Systemänderung (z. B. automatischer Archivierung) angepasst wird. Rückmeldungen wie Bestätigungen für Archivierungen oder Warnungen bei Fehlern werden klar und direkt angezeigt.

3.7 UML-Klassendiagramm

Abb. 1 – UML-Klassendiagramm (vereinfachte Beziehungsdarstellung aus Gründen der Übersichtlichkeit)



Quelle: eigene Darstellung

Das dargestellte UML-Klassendiagramm bietet einen Überblick über alle Klassen, die in der Anwendung verwendet werden. Es werden sowohl Hauptkomponenten als auch unterstützende Klassen dargestellt, um die Struktur der Anwendung zu verdeutlichen. Aus Gründen der Übersichtlichkeit sind jedoch nicht alle möglichen Beziehungen zwischen den Klassen vollständig abgebildet.

GUIController:
Zentrale Schnittstelle der Benutzeroberfläche.
Verantwortlich für die Koordination zwischen Benutzeraktionen und Geschäftslogik.
Verbindet sich mit nahezu allen anderen Komponenten, darunter SettingsManager, NotificationManager, ArchiveManager, FilterController, und DragDropHandler.

Task:
Repräsentiert eine einzelne Aufgabe.
Enthält Attribute wie Titel, Beschreibung, Fälligkeitsdatum und Prioritäten (Wichtigkeit, Dringlichkeit, Fitness).
Unterstützt Methoden zum Bearbeiten, Löschen und Markieren von Aufgaben als abgeschlossen.

User:
Repräsentiert einen Benutzer der Anwendung.
Bietet Methoden zur Authentifizierung und Profilverwaltung.

TaskEditor:
Eine GUI-Komponente zur Erstellung und Bearbeitung von Aufgaben.
Ermöglicht das Setzen von Standardprioritäten und das Speichern neuer Aufgaben.

SettingsManager:
Verwaltet die Benutzereinstellungen, wie Standardprioritäten, Benachrichtigungsintervalle und automatische Archivierungsoptionen.
Ermöglicht eine flexible Anpassung der Anwendung an individuelle Präferenzen.

NotificationManager:
Verantwortlich für die Planung und Anzeige von Benachrichtigungen.
Ruft Einstellungen vom SettingsManager ab und überprüft Fälligkeiten von Aufgaben.

ArchiveManager:
Verwaltet archivierte Aufgaben.
Bietet Methoden zum Archivieren, Abrufen und automatischen Löschen von Aufgaben.

FilterController:
Handhabt Filterfunktionen zur Anzeige bestimmter Aufgaben in der GUI.
Verbindet sich mit dem GUIController, um gefilterte Aufgabenlisten darzustellen.

DragDropHandler:
Verantwortlich für die Implementierung von Drag-and-Drop-Funktionen im Venn-Diagramm.
Ermöglicht das Verschieben von Aufgaben zwischen verschiedenen Prioritätskategorien.

ArchiveViewer:
GUI-Komponente zur Ansicht und Verwaltung archivierter Aufgaben.
Unterstützt Funktionen wie das Laden, Filtern und Wiederherstellen archivierter Aufgaben.

SettingsWindow:
GUI-Komponente für die Anpassung der Benutzereinstellungen.
Verbindet sich mit dem SettingsManager, um Änderungen an den Einstellungen zu speichern.

LoginWindow:
GUI-Komponente für die Benutzeranmeldung und -registrierung.
Verbindet sich mit dem UserRepository zur Authentifizierung und zum Speichern von Benutzerdaten.

UserRepository:
Kümmert sich um die Persistenz von Benutzerinformationen in der Datenbank.
Unterstützt das Abrufen und Speichern von Benutzerdaten.

DatabaseInitializer:
Initialisiert die Datenbankstruktur der Anwendung.
Stellt sicher, dass Tabellen wie Aufgaben, Benutzer oder Einstellungen existieren.

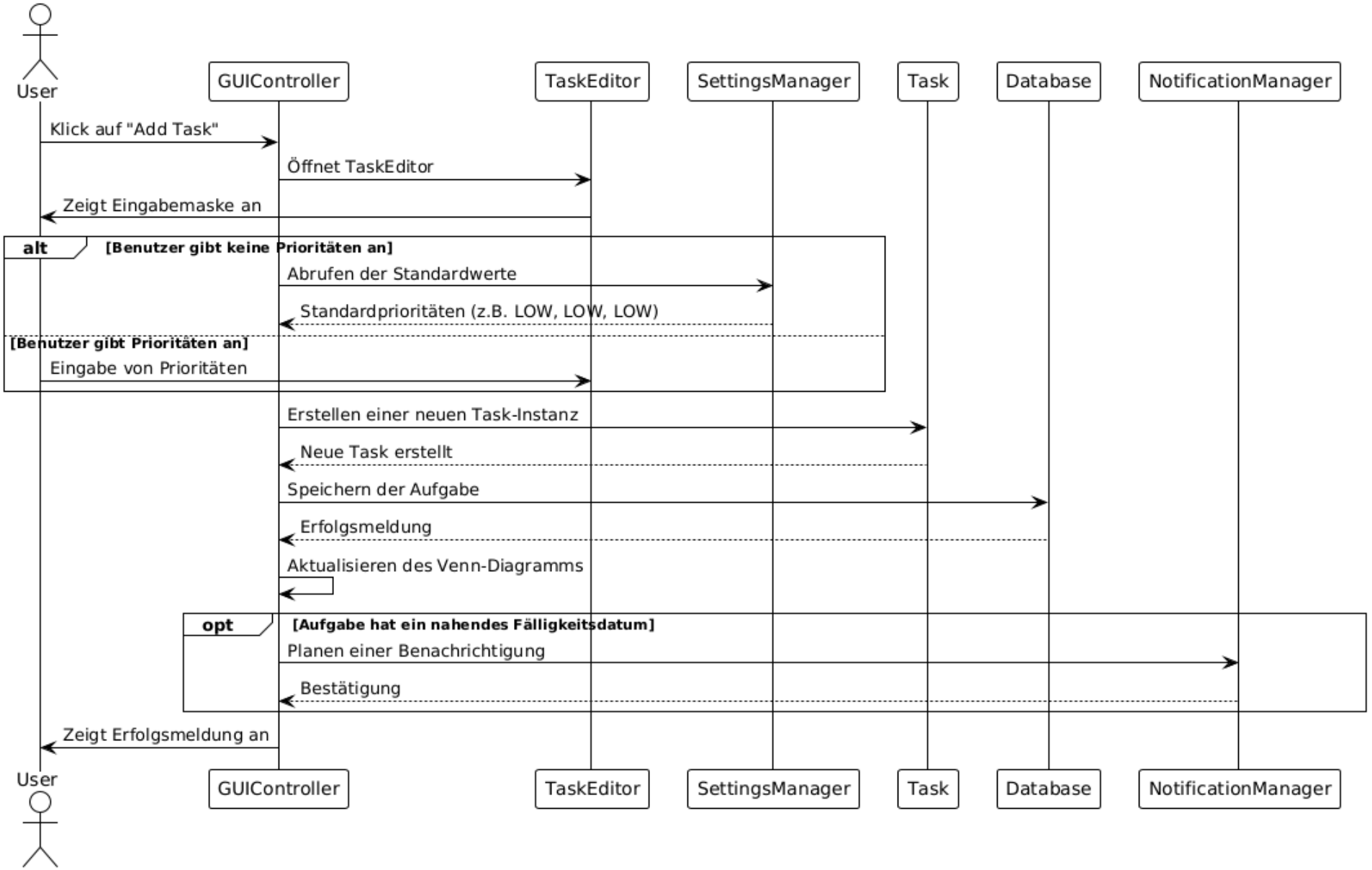
4. Verhalten – UML-Sequenzdiagramm zum Erstellen und Kategorisieren einer Aufgabe

Ablaufbeschreibung:

1. Aktion durch den User (GUI):
2. Der Benutzer initiiert die Aktion durch einen Button-Klick ("Add Task") in der GUI.
3. Aufruf des TaskEditors:
4. Der GUIController öffnet den TaskEditor, um dem Benutzer eine Eingabemaske zur Verfügung zu stellen, in der er Titel, Beschreibung, Fälligkeitsdatum und (falls gewünscht) Prioritäten angeben kann.
5. Abrufen von Standardwerten (SettingsManager):
6. Falls der Benutzer keine Prioritäten angibt, ruft der GUIController die im SettingsManager definierten Standardprioritäten (z. B. LOW, LOW, LOW) ab und verwendet diese als Fallback.
7. Erstellung einer neuen Task-Instanz:
8. Nach der Eingabe erstellt der GUIController eine neue Instanz der Task-Klasse und setzt Attribute wie Titel, Beschreibung, Prioritäten, Fälligkeitsdatum und Standardstatus (OPEN).
9. Speicherung der Aufgabe (TaskRepository/Database):

10. Die erstellte Aufgabe wird vom GUIController an das TaskRepository oder direkt an die Datenbank weitergegeben, wo sie persistiert wird.
11. Kategorisierung der Aufgabe:
12. Der GUIController aktualisiert das Venn-Diagramm oder andere GUI-Elemente, um die Aufgabe basierend auf ihren Prioritäten in die entsprechenden Kategorien (Wichtigkeit, Dringlichkeit, Fitness) einzuordnen.
13. Benutzerfeedback:
14. Der Benutzer erhält eine visuelle oder textliche Bestätigung in der GUI, dass die Aufgabe erfolgreich erstellt und kategorisiert wurde.
15. Aktualisierung der Benachrichtigungen (optional):
16. Der NotificationManager überprüft, ob die neue Aufgabe eine Benachrichtigung auslösen könnte (z. B. aufgrund eines nahenden Fälligkeitsdatums) und plant ggf. eine Benachrichtigung.

Abb. 2 – UML-Sequenzdiagramm zum Erstellen und Kategorisieren einer Aufgabe



Quelle: eigene Darstellung

Literaturverzeichnis

Schatten, A., Biffl, S., Demolsky, M., Gostischa-Franta, E., Östreicher, T., & Winkler, D. (2010). *Best Practice Software-Engineering: Eine Praxiserprobte Zusammenstellung Von Komponenten-orientierten Konzepten, Methoden und Werkzeugen*. Spektrum Akademischer Verlag GmbH. http://ebookcentral.proquest.com/lib/badhonnef/detail.action?docID=511284