Portfolio – Phase 3

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 3): 08.12.2024

Änderungen nach Phase 2:
Use-Case-Diagramm lt. Feedback angepasst (Abb. 3)
6 Projektdokumentation angepasst
8 Status der Implementierung zum Ender der Phase 3 angepasst
9 Benutzeranleitung eingefügt

Inhaltsverzeichnis

## Abbildungsverzeichnis



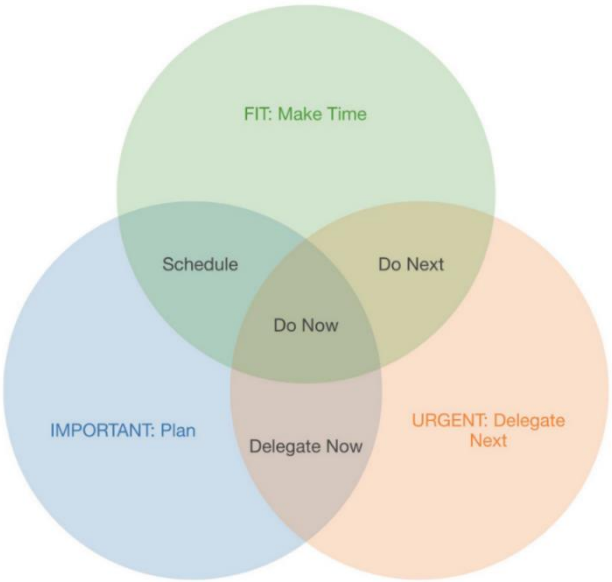
## Tabellenverzeichnis

## 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 |

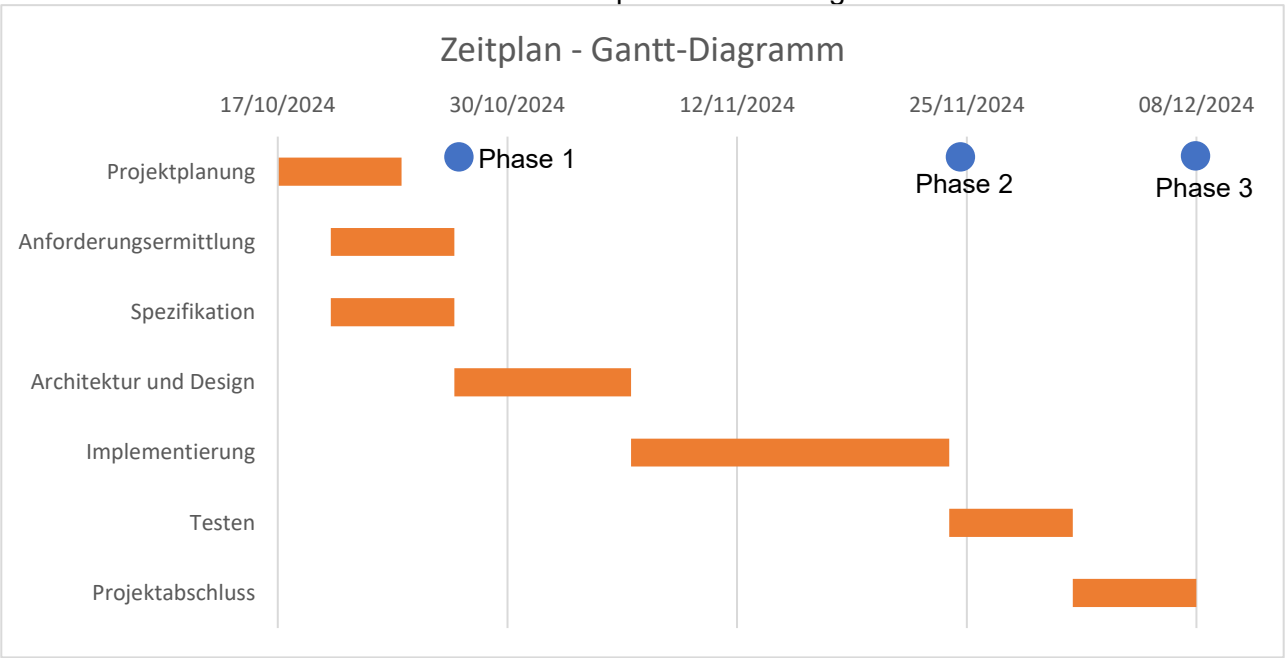
Quelle: eigene Darstellung

## 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 (mit der Ausnahme von Anpassungen aufgrund von Feedback).

- 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 (mit der Ausnahme von Anpassungen aufgrund von Feedback).

- 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 nach Phase 2, aus Gründen der Nachvollziehbarkeit, nicht mehr verändert (mit der Ausnahme von Anpassungen aufgrund von Feedback).

- Testdokument

  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 wurde in Phase 3 neu erstellt.

- 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.

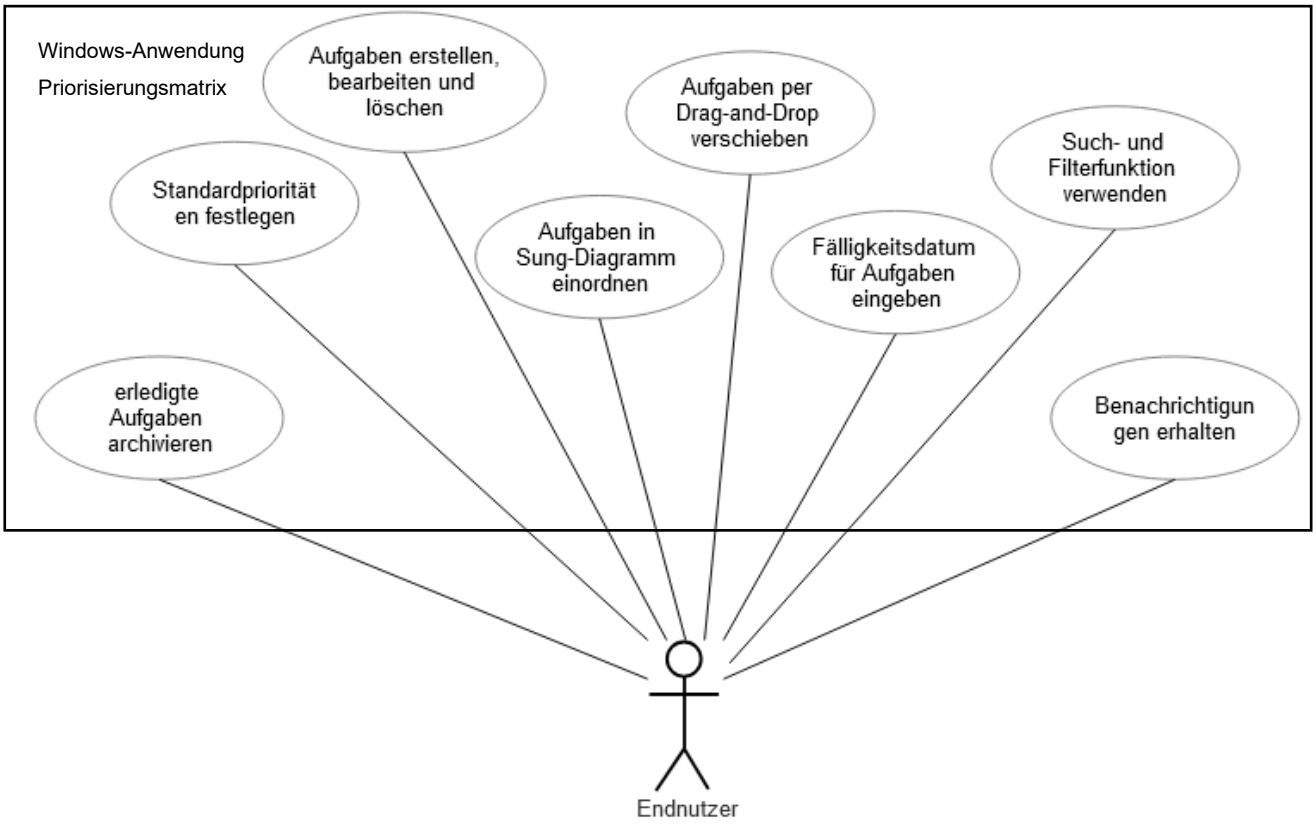
*Schichtenarchitektur:*

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 (Abb. 3 - vgl. Anforderungsdokument, S. 10, Abb. 2) 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 3

Die Implementierung wurde eine Woche vor dem Ender der Phase 3, am 01.12.2024, erneut 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 wurden unten 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.

*Verbesserungspotentiale und offene Punkte:*

Trotz der stabilen Basis für die Kernfunktionalität der Anwendung gibt es verschiedene Aspekte, die weiteres Optimierungspotential aufzeigen:

- Formatierungen der GUI:
    - Aufgaben könnten nicht nur als Text, sondern in ansprechend gestalteten Kästchen dargestellt werden.
    - Einstellungen könnten logisch gruppiert werden, um die Übersichtlichkeit zu erhöhen.

- Zielfelder für Drag-and-Drop:
    - Die Funktionalität ist implementiert, jedoch sind die Zielfelder für Prioritäten nicht optimal angepasst.
- Drag-and-Drop für die LOW-Liste:
    - Die Möglichkeit, Aufgaben per Drag-and-Drop aus der LOW-Liste zu verschieben, ist aktuell noch nicht implementiert.
- Code-Trennung und Schichtenarchitektur:
    - Besonders im Bereich des GUI-Controllers wurde die Schichtenarchitektur nicht immer strikt eingehalten. Eine konsequentere Trennung zwischen den Schichten wäre wünschenswert, war jedoch zeitlich nicht mehr realisierbar.
- Ersetzen von Magic Numbers:
    - Aktuell existieren zum Teil noch hardcodierte Werte. Diese könnten durch Konstanten oder Konfigurationsparameter ersetzt werden.
- Ausgebautes Error-Handling:
    - Insbesondere bei Datenbankanfragen ist das Fehlermanagement ausbaufähig.
- Passwort-Sicherheit:
    - Die Integration von Salting in der Passwort-Hashing-Funktion würde die Sicherheit erhöhen. Dies wurde jedoch als weniger kritisch eingestuft, da die Anwendung für die lokale Nutzung und nicht für große Benutzergruppen ausgelegt ist.
- Optimierung der Geschäftsabläufe:
    - Beispielsweise sollte das Editor-Fenster bei fehlerhaften Eingaben offen bleiben, anstatt sich direkt zu schließen, wie es derzeit der Fall ist.
- Datumsauswahl via Calendar-Picker:
    - Derzeit müssen Benutzer das Fälligkeitsdatum manuell in ein Textfeld eingeben, was zu Fehlern führen kann. Die Integration eines Calendar-Picker-Controls würde die Benutzerfreundlichkeit erheblich verbessern, indem ein intuitiveres und fehlerfreies Auswählen von Daten ermöglicht wird.
- Position des Dialogs zum Anlegen/Bearbeiten von Tasks:
    - Der Dialog für das Anlegen oder Bearbeiten von Aufgaben öffnet sich aktuell immer an einer festen Position, was für einige Benutzer unpraktisch erscheinen kann. Eine dynamischere Platzierung, entweder zentriert auf dem Bildschirm oder in der Nähe des Klickpunktes, könnte die Bedienung deutlich angenehmer gestalten.

- Bearbeiten eines Tasks durch Doppelklick:
    - Es wird derzeit erwartet, dass Benutzer Buttons verwenden, um Tasks zu bearbeiten. Die Einführung einer Doppelklick-Funktion zum direkten Öffnen und Bearbeiten von Tasks würde die Bedienung intuitiver machen und die Effizienz steigern.

Diese Punkte zeigen auf, wo die nächste Iteration ansetzen könnte, um die Anwendung weiter zu verbessern und zu optimieren.

Die Aufgaben der Finalisierungsphase (Implementierung abschließen, Tests durchführen, Benutzerhandbuch erstellen, Reflexion des Projekts) waren sehr umfangreich und hätten mehr Zeit erfordert, um sie noch detaillierter umzusetzen.

## 9. Benutzeranleitung

Diese Benutzeranleitung beschreibt die notwendigen Schritte, um die Anwendung zu installieren und auszuführen. Ziel ist es, den Endnutzer:innen eine klare und einfache Anleitung zu bieten, um die Software erfolgreich einzurichten und zu verwenden. Die Anwendung besteht aus einer ausführbaren Datei (*main.exe*) und einer zugehörigen Datenbank (*database.db*). Beide Dateien müssen korrekt im selben Verzeichnis vorliegen, um eine fehlerfreie Nutzung zu gewährleisten.

### 9.1 Installation und Vorbereitung

1. Erhalt der Anwendung
   Die Anwendung ist entweder als ZIP-Archiv verfügbar oder kann über das GitHub-Repository (https://github.com/craexxn/PrioritizationSung) heruntergeladen werden. In beiden Fällen befinden sich die zwei notwendigen Dateien (*main.exe* und *database.db*) im Ordner */dist/*.
2. Entpacken oder Kopieren der Dateien
    - Falls die Anwendung als ZIP-Archiv vorliegt:
        - Laden Sie das ZIP-Archiv herunter und entpacken Sie es an einen beliebigen Ort auf Ihrem Computer.
        - Stellen Sie sicher, dass die beiden Dateien main.exe und database.db im gleichen Ordner liegen.
    - Falls die Anwendung aus dem GitHub-Repository heruntergeladen wurde:
        - Navigieren Sie zum Ordner /dist/ innerhalb des Repositorys und kopieren Sie die beiden Dateien main.exe und database.db in einen lokalen Ordner Ihrer Wahl.

- Vergewissern Sie sich, dass sich die beiden Dateien im selben Ordner befinden.

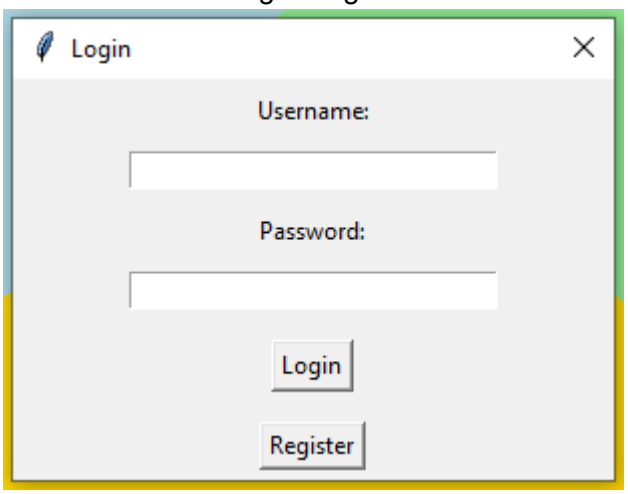
## 9.2 Start der Anwendung

1. Ausführen der *main.exe*
   Um die Anwendung zu starten, doppelklicken Sie auf die Datei main.exe. Nach einem kurzen Ladevorgang (gegebenenfalls beim Erst-Start einmalige Überprüfung/Freigabe durch Anitvirus-Software, danach bitte main.exe neustarten) erscheint das Login-Fenster. Dieses Fenster bietet zwei Optionen:
   - Login: Für bereits registrierte Benutzer
   - Register: Für neue Benutzer, die noch kein Konto haben.

## 9.3 Registrierung eines Benutzers

1. Eingabe der Benutzerinformationen
   - Geben Sie einen Benutzernamen ein, der zwischen 3 und 20 Zeichen lang ist und ausschließlich alphanumerische Zeichen enthält (keine Leerzeichen oder Sonderzeichen).
   - Geben Sie ein Passwort ein, das mindestens 3 Zeichen lang ist.

Abb. 4 – Login/Register-Window




Quelle: eigene Darstellung

2. Bestätigung der Registrierung
   - Klicken Sie auf Register.
   - Bei erfolgreicher Registrierung wird eine Bestätigungsmeldung angezeigt, und Sie werden zurück zum Login-Fenster geleitet.
3. Fehlerbehebung
   - Falls der Benutzername bereits existiert oder die Eingaben nicht den Anforderungen entsprechen, wird eine Fehlermeldung angezeigt. Korrigieren Sie die Eingaben entsprechend.

## 9.4 Login eines Benutzers

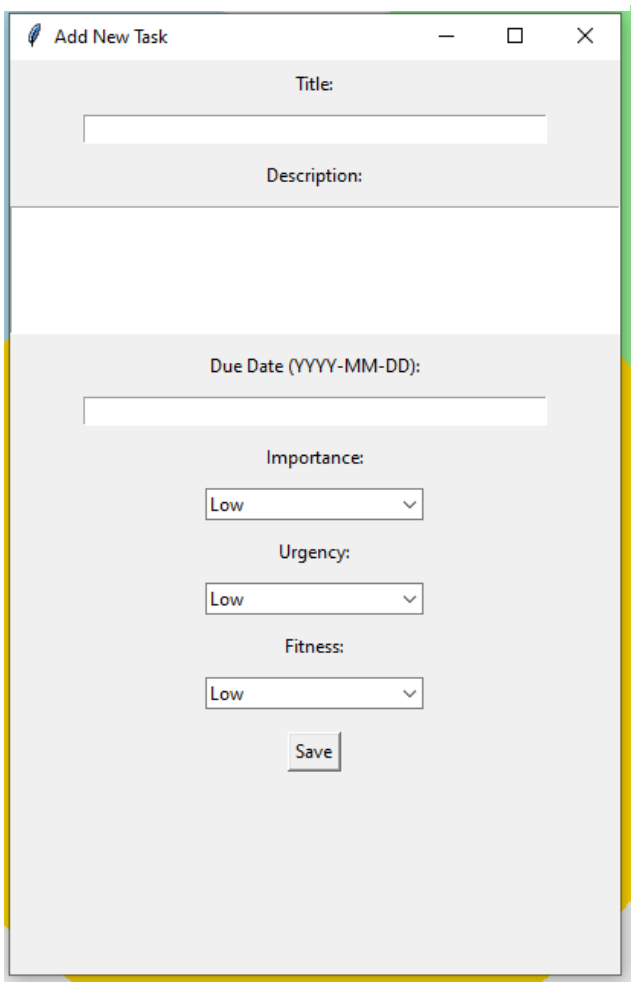
1. Eingabe der Benutzerinformationen
    - Im Login-Fenster geben Sie Ihren Benutzernamen und Ihr Passwort in die entsprechenden Felder ein.
    - Die gleichen Anforderungen wie bei der Registrierung gelten auch hier:
        - Der Benutzername muss zwischen 3 und 20 alphanumerische Zeichen enthalten.
        - Das Passwort muss mindestens 3 Zeichen lang sein.
2. Anmeldung
    - Klicken Sie auf die Schaltfläche Login.
    - Nach erfolgreicher Anmeldung wird das Hauptfenster der Anwendung geöffnet. Sie haben nun Zugriff auf die Aufgabenverwaltung.
3. Fehlermeldungen
    - Sollten die Eingaben nicht korrekt sein, wird eine Fehlermeldung angezeigt. Vergewissern Sie sich, dass der Benutzername und das Passwort korrekt eingegeben wurden.
    - Falls das Problem weiterhin besteht, überprüfen Sie, ob der Benutzername bereits registriert wurde, oder registrieren Sie einen neuen Benutzer.
4. Hinweis
    - Das Passwort wird sicher verschlüsselt in der Datenbank gespeichert. Es kann nicht wiederhergestellt werden. Sollte es vergessen werden, müssen Sie einen neuen Benutzer erstellen.

## 9.5 Erstellung einer neuen Aufgabe

1. Zugriff auf die Funktion zur Aufgabenerstellung
    - Nach dem erfolgreichen Login klicken Sie im Hauptfenster auf die Schaltfläche Add Task, die sich in der unteren Menüleiste befindet.
2. Ausfüllen der Task-Daten
    - Ein neues Fenster öffnet sich, in dem die Details der Aufgabe eingegeben werden können:
        - Titel: Geben Sie einen aussagekräftigen Titel ein (maximal 25 Zeichen).

- Beschreibung: Fügen Sie eine detaillierte Beschreibung hinzu (optional, maximal 250 Zeichen).
- Fälligkeitsdatum: Geben Sie das Datum im Format YYYY-MM-DD ein (z. B. 2024-12-31).
- Prioritäten: Wählen Sie die Wichtigkeit, Dringlichkeit und Fitness der Aufgabe aus den Dropdown-Menüs aus. Die Optionen sind Low und High.

Abb. 5 – Task-Editor

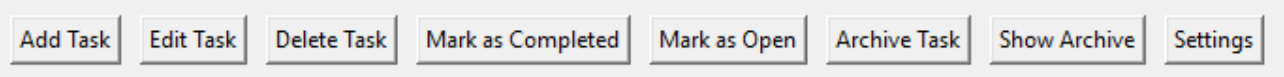



Quelle: eigene Darstellung

3. Speichern der Aufgabe
   - Klicken Sie auf die Schaltfläche Save, um die Aufgabe zu speichern.
   - Die Aufgabe wird in der Aufgabenübersicht im Hauptfenster angezeigt und entsprechend den Prioritäten im Venn-Diagramm positioniert.
4. Fehlermeldungen bei der Eingabe
   - Stellen Sie sicher, dass der Titel und das Fälligkeitsdatum korrekt eingegeben wurden. Falls nicht, wird eine entsprechende Fehlermeldung angezeigt.
5. Hinweis
   - Die Prioritäten bestimmen die Position der Aufgabe im Venn-Diagramm. Aufgaben mit niedrigen Prioritäten werden in der separaten LOW Priority Tasks-Liste rechts angezeigt.

## 9.6 Bearbeitung und Verwaltung von Aufgaben

Abb. 6 – Menu-Buttons




Quelle: eigene Darstellung

*Bearbeiten einer Aufgabe*

1. Auswahl einer Aufgabe
    - Klicken Sie auf eine Aufgabe im Venn-Diagramm oder in der LOW Priority Tasks-Liste, um sie auszuwählen.
2. Bearbeiten der Aufgabe
    - Klicken Sie auf die Schaltfläche Edit Task in der unteren Menüleiste.
    - Im Bearbeitungsfenster können Sie den Titel, die Beschreibung, das Fälligkeitsdatum und die Prioritäten der Aufgabe anpassen.
    - Speichern Sie die Änderungen mit Save.
3. Drag-and-Drop-Funktion
    - Aufgaben im Venn-Diagramm können per Drag-and-Drop zwischen den verschiedenen Prioritätsbereichen verschoben werden.
    - Klicken Sie eine Aufgabe an, halten Sie die Maustaste gedrückt, und ziehen Sie die Aufgabe in den gewünschten Bereich des Diagramms.
    - Die Prioritäten der Aufgabe werden automatisch basierend auf ihrer neuen Position aktualisiert.
    - Hinweis: Aufgaben mit allen Prioritäten auf LOW werden automatisch in die LOW Priority Tasks-Liste verschoben. Eine Verschiebung aus der LOW Priority Tasks-Liste ist derzeit nicht möglich.

*Löschen einer Aufgabe*

1. Auswahl der Aufgabe
    - Wählen Sie eine Aufgabe aus dem Diagramm oder der Liste aus.
2. Löschen
    - Klicken Sie auf die Schaltfläche Delete Task.
    - Es erscheint eine Sicherheitsabfrage, ob die Aufgabe wirklich gelöscht werden soll.
    - Bestätigen Sie mit Yes, um die Aufgabe dauerhaft zu entfernen.

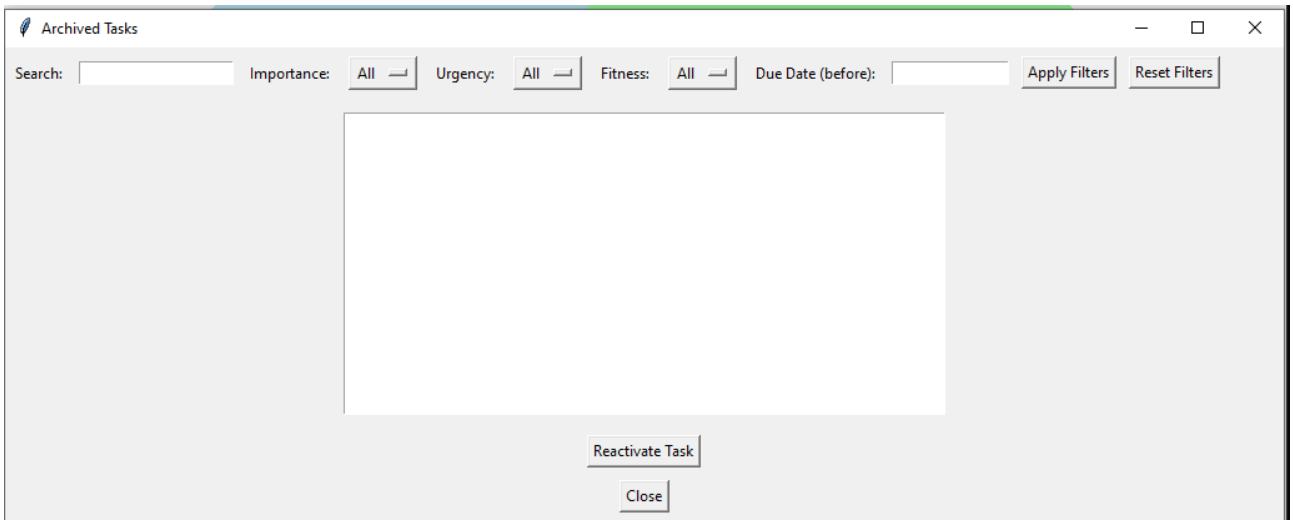
*Statusänderung: Als abgeschlossen oder offen markieren*

1. Markieren als abgeschlossen
    - Wählen Sie eine Aufgabe aus und klicken Sie auf Mark as Completed.
    - Die Aufgabe wird in den COMPLETED-Status versetzt und kann je nach Einstellung im Settings-Window automatisch archiviert werden.
2. Markieren als offen
    - Wählen Sie eine abgeschlossene Aufgabe aus und klicken Sie auf Mark as Open, um sie wieder in den OPEN-Status zu setzen.

*Archivierung von Aufgaben*

1. Manuelle Archivierung
    - Wählen Sie eine abgeschlossene Aufgabe aus und klicken Sie auf Archive Task.
    - Die Aufgabe wird in das Archiv verschoben.
2. Archiv anzeigen
    - Klicken Sie auf Show Archive, um archivierte Aufgaben anzuzeigen und ggf. wieder zu aktivieren.

Abb. 7 – Archive-Viewer




Quelle: eigene Darstellung

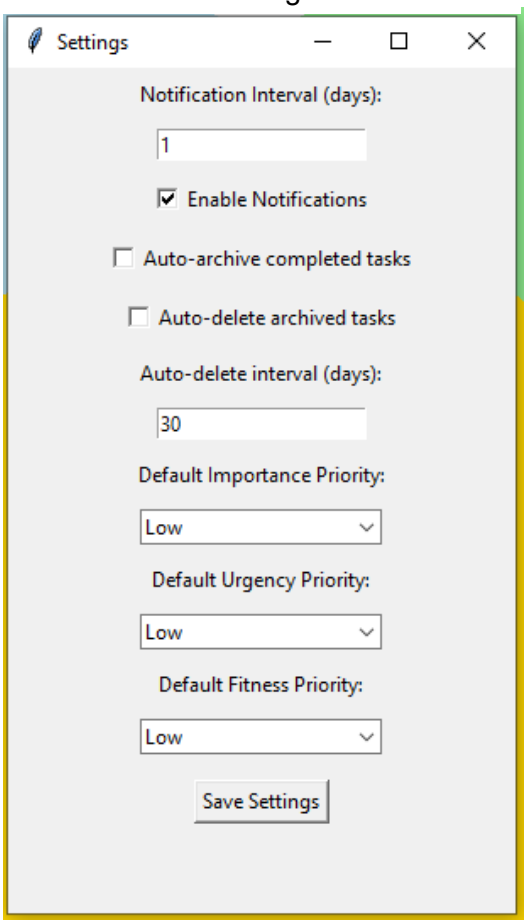
## 9.7 Einstellungen

Das Settings-Window ist über die Schaltfläche Settings erreichbar und ermöglicht die Anpassung der folgenden Parameter:

1. Benachrichtigungen: Aktivieren/Deaktivieren und Festlegen des Intervalls.
2. Automatische Archivierung: Aktivieren Sie, ob abgeschlossene Aufgaben automatisch archiviert werden solle.
3. Automatische Löschung: Legen Sie fest, ob archivierte Aufgaben nach einer bestimmten Zeit gelöscht werden sollen.
4. Standard-Prioritäten: Definieren Sie, welche Prioritäten beim Erstellen neuer Aufgaben voreingestellt sein sollen.

Alle Änderungen werden durch einen Klick auf Save Settings gespeichert. Die Einstellungen sind benutzerspezifisch und wirken sich nur auf den aktuellen Nutzer aus.

Abb. 8 – Settings-Window

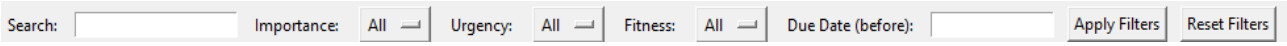



Quelle: eigene Darstellung

## 9.8 Filter-Einstellungen

Die Anwendung bietet umfangreiche Filterfunktionen, um gezielt Aufgaben in der Task- oder Archiv-Ansicht anzuzeigen.

Abb. 9 – Filter-Menu




Quelle: eigene Darstellung

*Filter in der Task-Ansicht*

1. Öffnen der Filteroptionen
    - Die Filteroptionen befinden sich in der Hauptansicht unten und bieten die Möglichkeit, die angezeigten Aufgaben nach bestimmten Kriterien einzuschränken.
2. Filterkriterien
    - Suche (Search): Geben Sie einen Teil des Titels oder den gesamten Titel ein, um die entsprechenden Aufgaben anzuzeigen.
    - Wichtigkeit (Importance): Zeigt nur Aufgaben mit LOW oder HIGH Importance an.

- Dringlichkeit (Urgency): Filtern Sie nach Aufgaben, die als LOW oder HIGH Urgency markiert sind.
- Fitness (Fitness): Filtern Sie nach Aufgaben, die als LOW oder HIGH Fitness priorisiert sind.
- Fälligkeitsdatum (Due Date): Zeigt nur Aufgaben an, die vor einem bestimmten Datum fällig sind.

3. Anwendung der Filter
    - Nach der Auswahl der Filterkriterien wird die Ansicht nach Klicken auf *Apply Filters* aktualisiert und zeigt nur die entsprechenden Aufgaben.
    - Hinweis: Durch Entfernen aller Filter (Klick auf *Reset Filters*) werden wieder alle Aufgaben angezeigt.

*Filter in der Archivansicht*

Die Filterfunktion in der Archivansicht funktioniert genauso wie in der Task-Ansicht, mit der folgenden Ausnahme:

- Position der Filteroptionen: Die Filteroptionen befinden sich in der Archivansicht oben, direkt über der Liste der archivierten Aufgaben.

Hinweis zu den Filter-Einstellungen

Die Filter-Einstellungen werden nicht dauerhaft gespeichert und gelten nur für die aktuelle Sitzung. Möchten Sie häufig verwendete Filtereinstellungen behalten, können zukünftige Updates dies eventuell unterstützen.

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 3

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 3): 08.12.2024

Änderungen nach Phase 2:
Use-Case-Diagramm lt. Feedback angepasst (Abb. 2)

## 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.

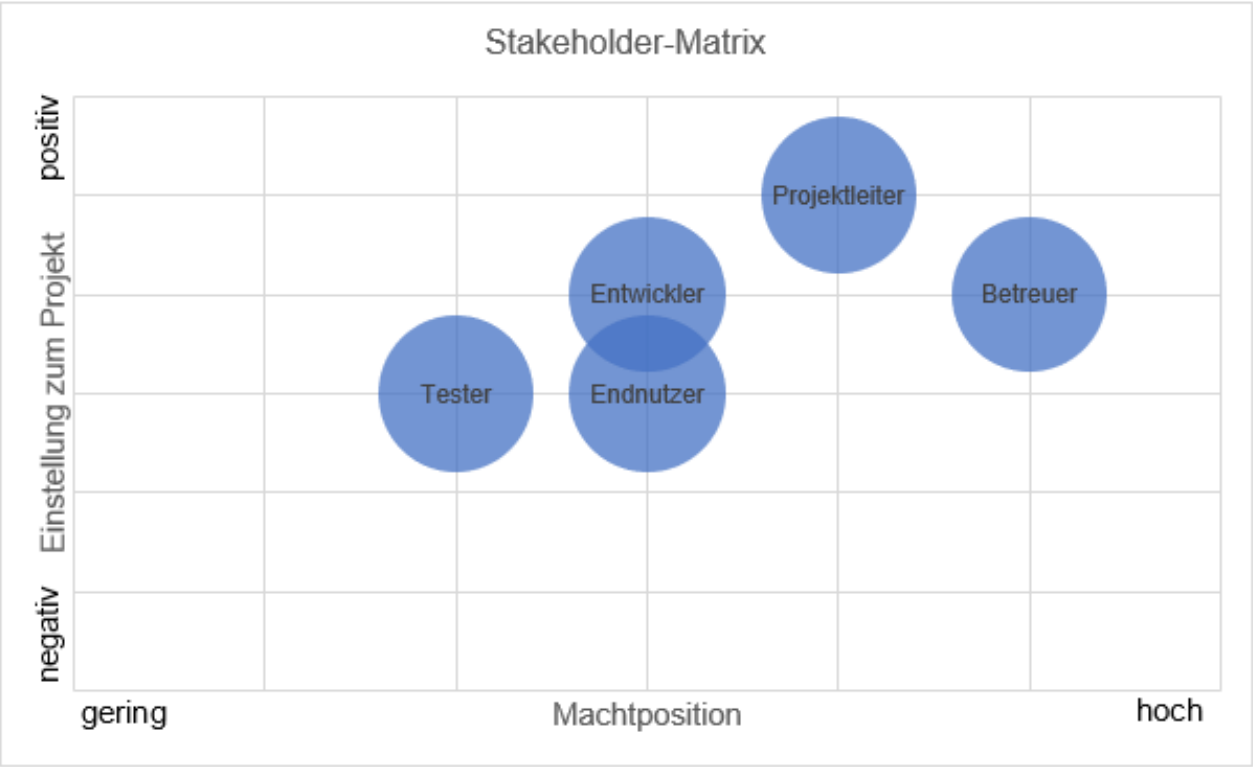
*Privatpersonen mit komplexem Aufgabenmanagement*: Menschen, die auch im privaten Bereich viele Verpflichtungen haben, wie z. B. Eltern, die sowohl berufliche als auch familiäre Aufgaben koordinieren müssen.

*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,*<br>*2 = Should have*<br>*3 = Could have*<br>*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

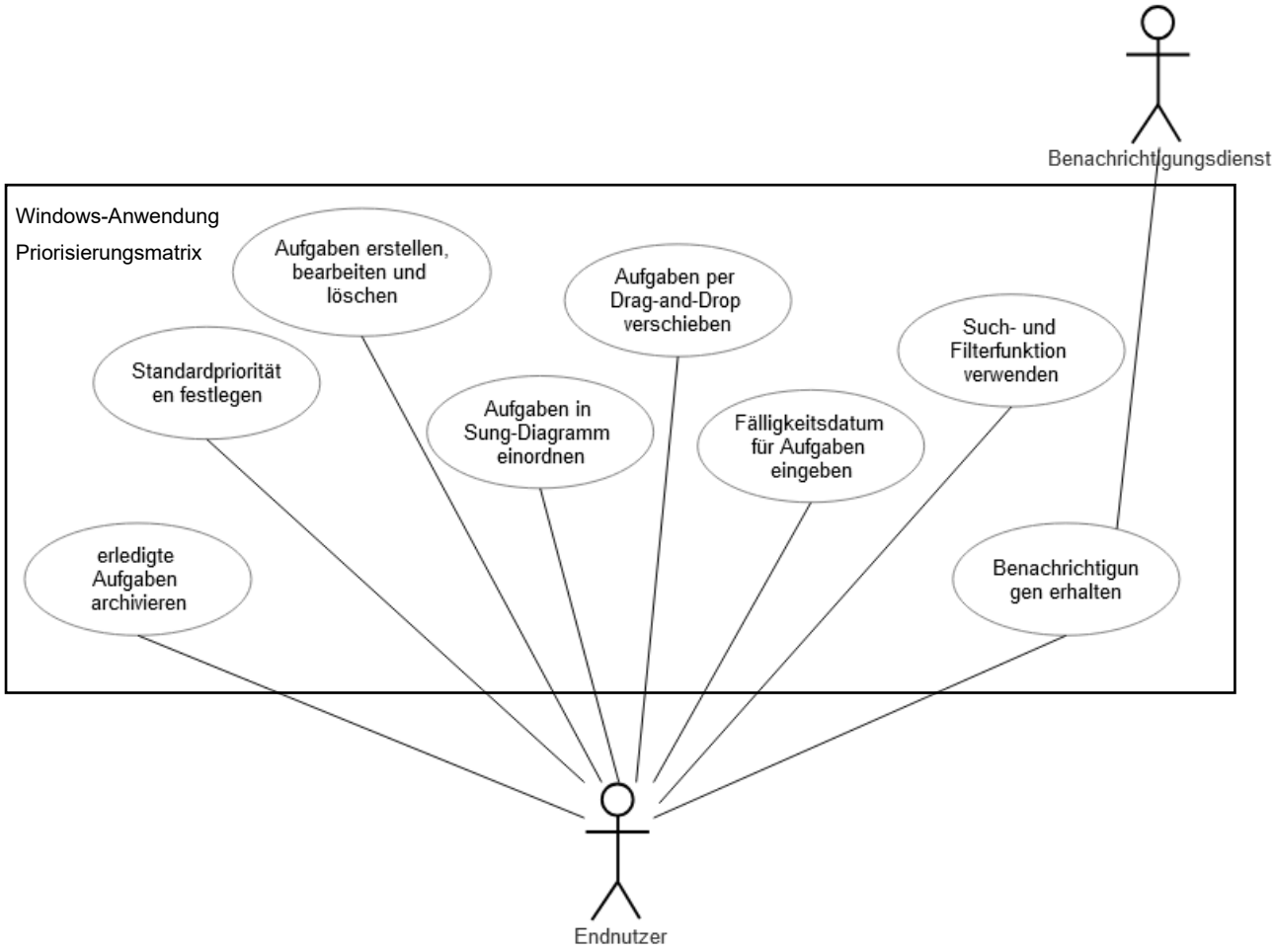
| ID | Anforderung | User-Story |
|---|---|---|
| Req_1 | Aufgabenverwaltung | Als Nutzer möchte ich Aufgaben erstellen, bearbeiten und löschen können, damit ich meine Aufgaben flexibel anpassen und verwalten kann. Ich möchte, dass diese Aktionen intuitiv über Buttons, Kontextmenüs und Textfelder erfolgen. |
| 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.<br>Die Benutzeroberfläche soll eine durchschnittliche Erlern-Zeit von unter 15 Minuten aufweisen, gemessen durch Usability-Tests.<br>*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.<br>*Akzeptanzkriterium*: Die Anwendung soll auf Benutzeraktionen innerhalb von 200ms reagieren. In Ausnahmefällen sind 1000ms zulässig.<br>*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.<br>*Akzeptanzkriterium*: Die Anwendung muss eine 99,9 % Verfügbarkeit während der Nutzungszeit haben und auf Speicher- oder CPU-Beschränkungen robust reagieren.<br>*Testmethoden*: Stabilitätstests unter Ressourcenbeschränkungen. | 1 |
| qReq_4 | Qualität | Kompatibilität | Die Anwendung wird als native Desktop-App für Windows 10/11 entwickelt.<br>*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.<br><br>*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.<br><br>*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,*<br>*2 = Should have*<br>*3 = Could have*<br>*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 3

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 3): 08.12.2024

Änderungen nach Phase 2:
UML-Klassen-Diagramm lt. Feedback angepasst (Abb. 1)

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

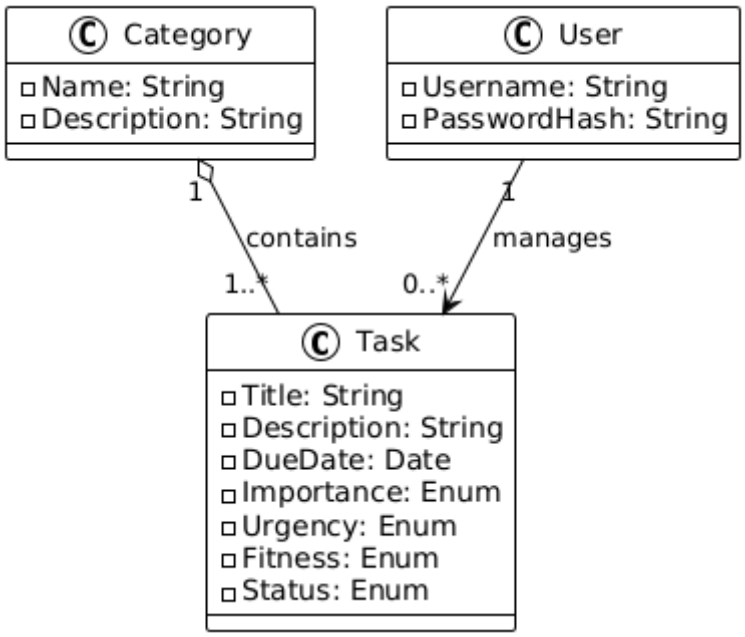
Das UML-Klassendiagramm stellt die Struktur der zentralen Geschäftsobjekte der Anwendung dar und zeigt die Beziehungen zwischen diesen Objekten auf. Die Klassen im Diagramm sind User, Task und Category.

- 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.

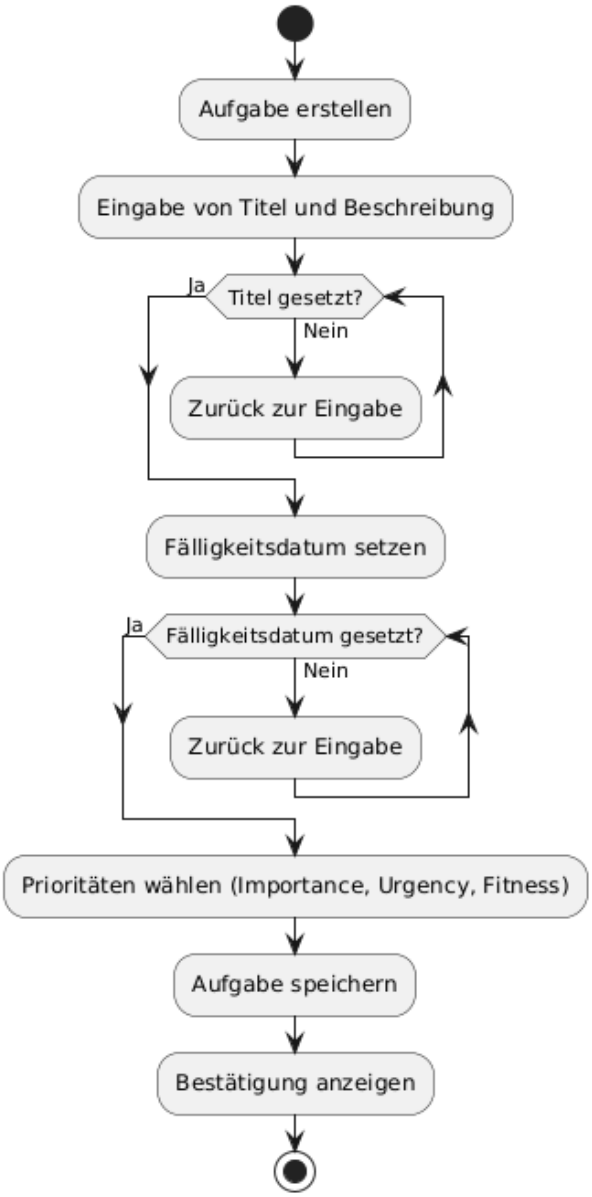
Ablauf:

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

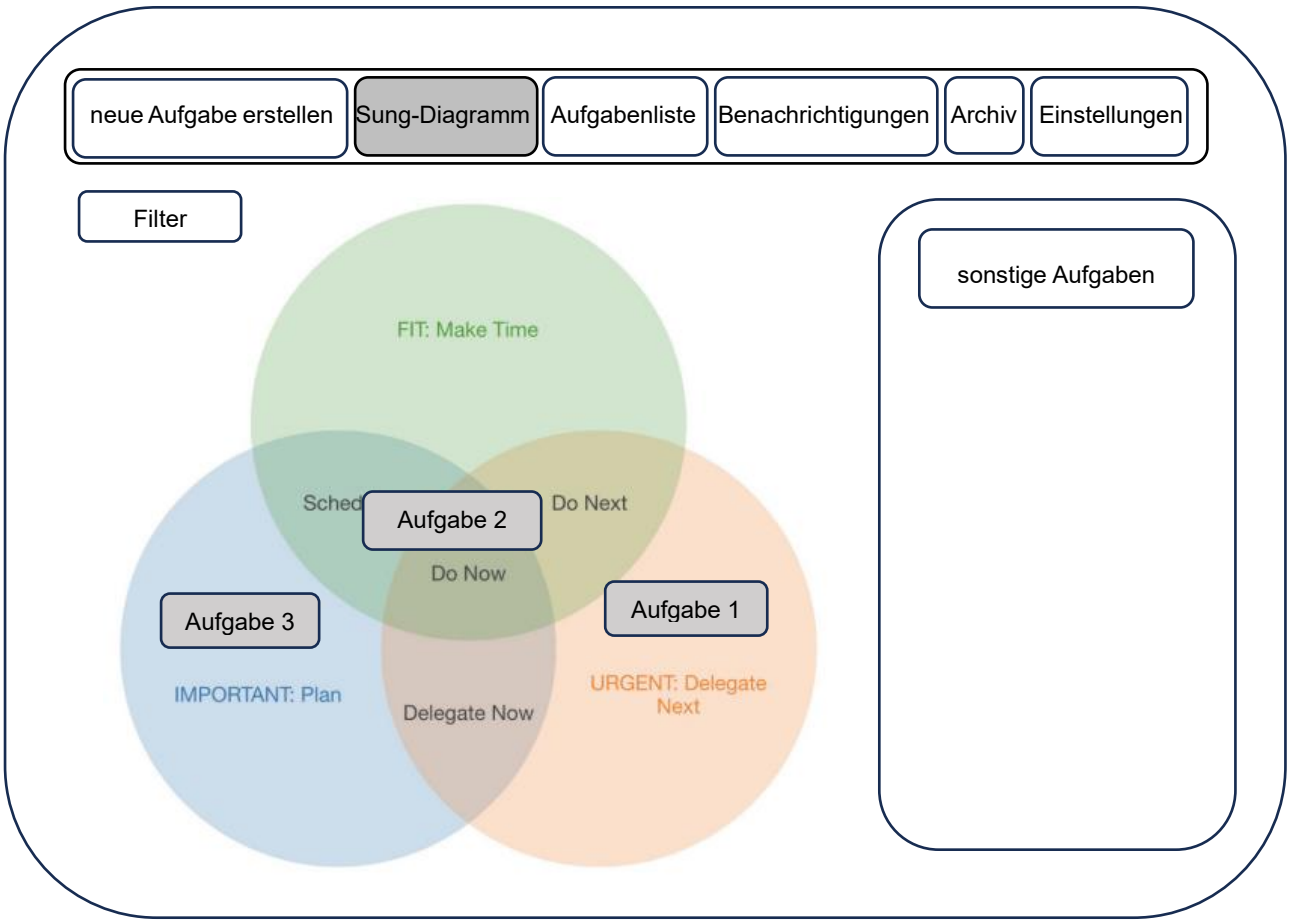
*Beschreibung*: Die Hauptansicht der Anwendung zeigt das Sung-Diagramm oder die Aufgabenliste, in das die Aufgaben kategorisiert werden können. Die Hauptansicht bietet den Nutzern eine visuelle Übersicht ihrer Aufgaben.

*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.

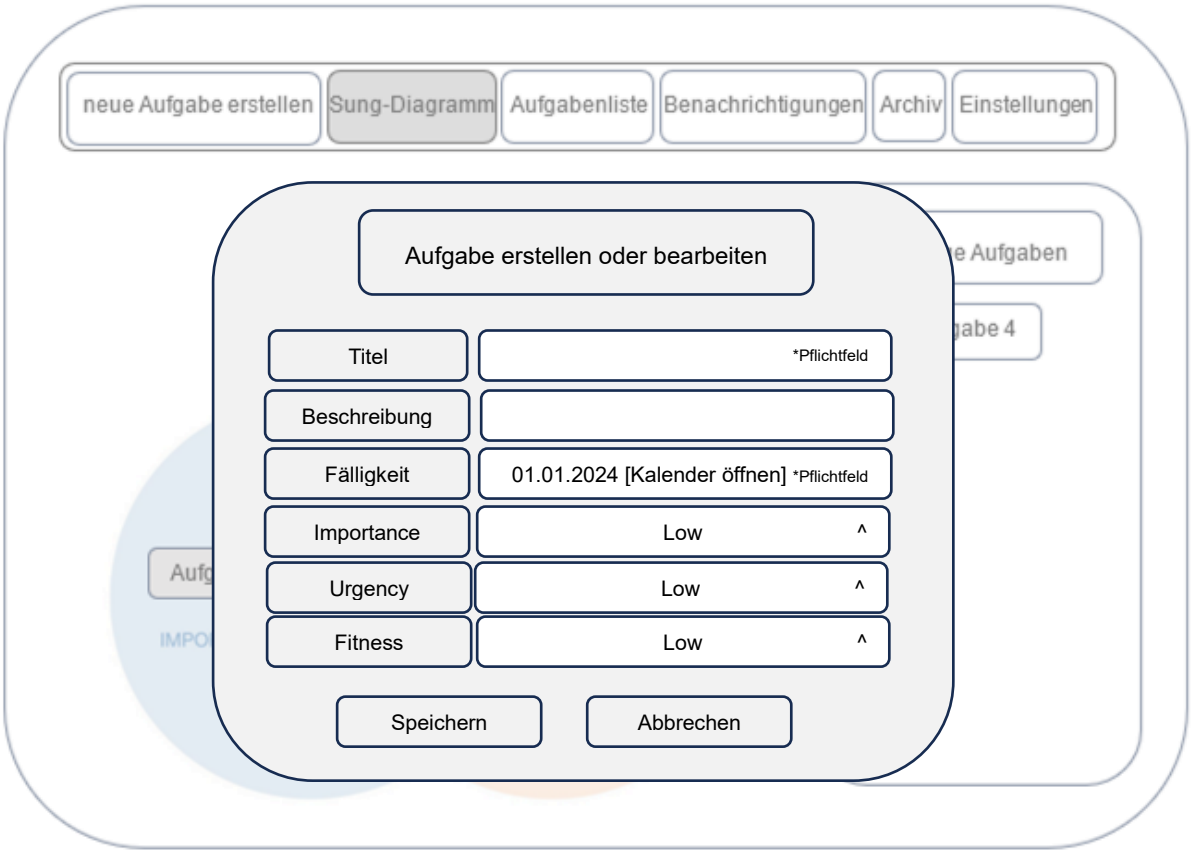
*Bestandteile*:
*Titel und Beschreibung*: Ein Pflichtfeld für den Titel und ein optionales Feld für eine detaillierte Beschreibung der Aufgabe.

*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.

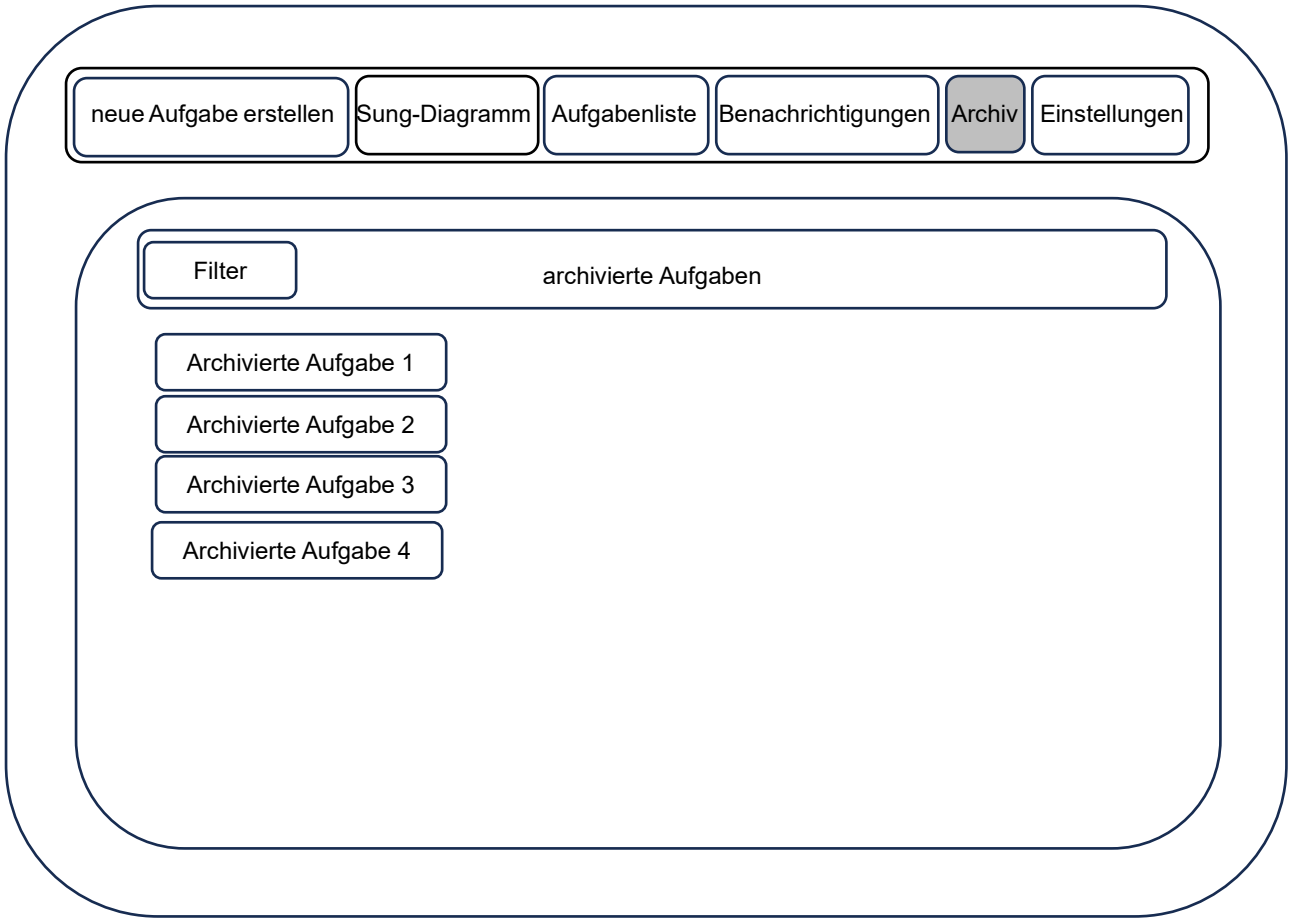
*Bestandteile*:

*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 3

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 3): 08.12.2024

Änderungen nach Phase 2:

Keine Änderungen.

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.

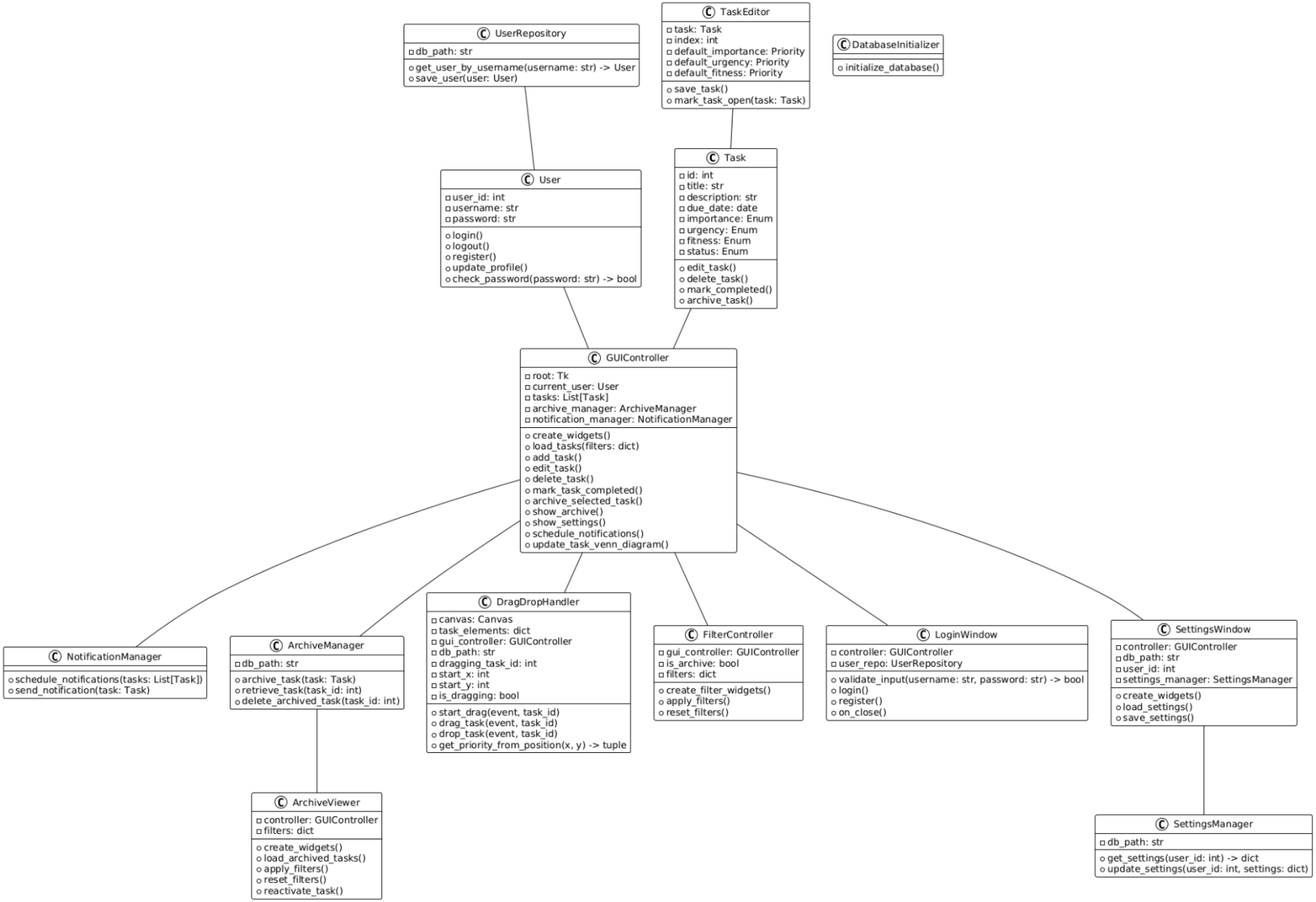
## 3.6 Übersicht der Benutzeraktionen

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.

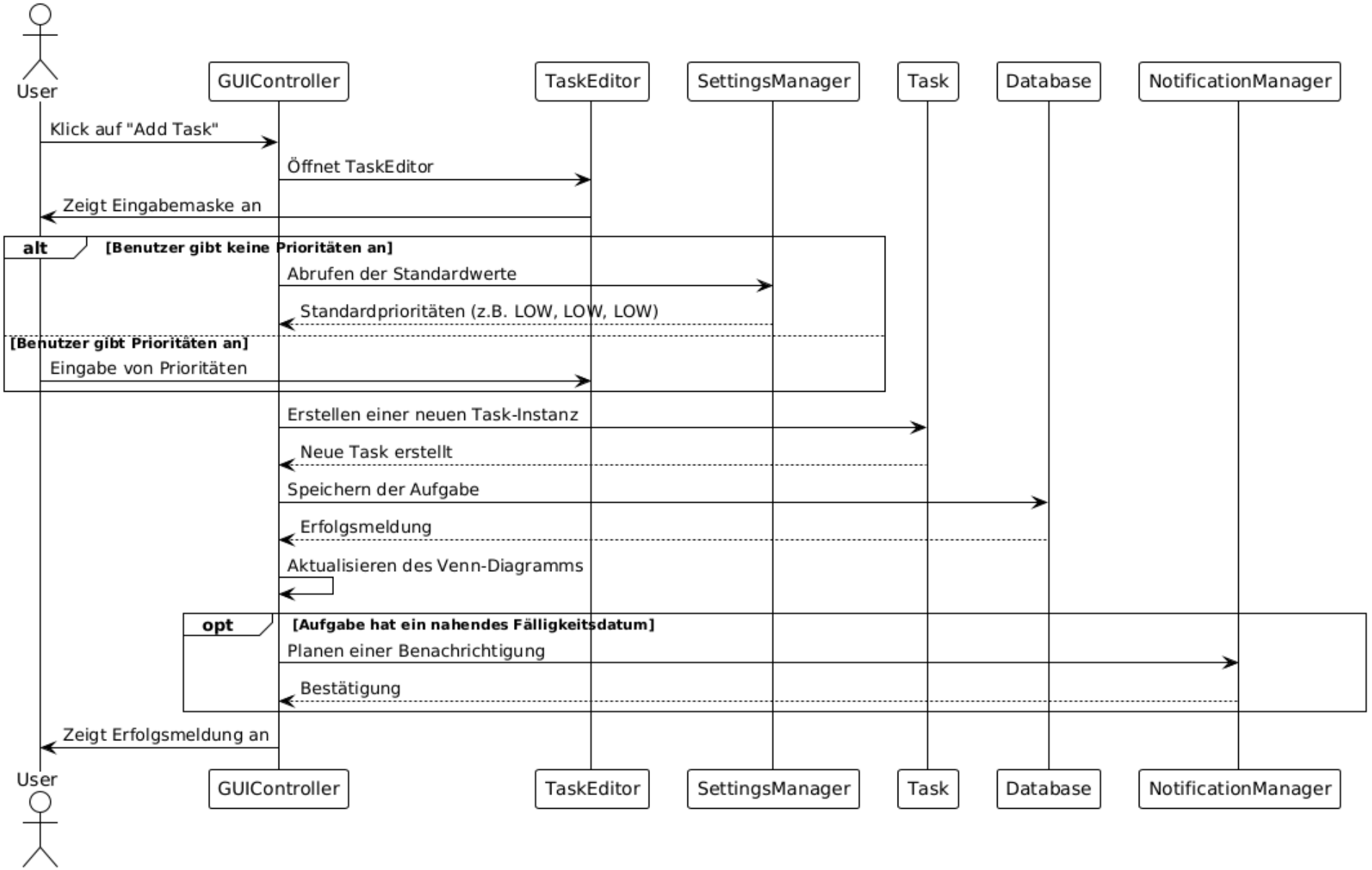
## 4. Verhalten – UML-Sequenzdiagramm zum Erstellen und Kategorisieren einer Aufgabe

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

Portfolio – Phase 3

IU Internationale Hochschule Fernstudium

Studiengang: Master of Science Informatik

Kurs: Projekt – Software Engineering (DLMCSPSE01_D)

# Testdokument

## Windows Anwendung:
## Priorisierungsmatrix nach dem Sung-Diagramm

Gregor Hubmann

Matrikelnummer: 9196398

Hornstraße 185

5423 St. Koloman

Abgabedatum (Phase 3): 08.12.2024

Änderungen nach Phase 2:
Testdokument wurde in Phase 3 neu erstellt.

Inhaltsverzeichnis

## Abbildungsverzeichnis



## Tabellenverzeichnis

# 1. Einleitung

Dieses Testdokument dient als Grundlage für die Qualitätssicherung der Anwendung „*Priorisierungsmatrix nach dem Sung-Diagramm*“. Es beschreibt die geplante Teststrategie, die Teststufen sowie die Durchführung und Ergebnisse der Tests. Ziel ist es, die Funktionalität, Stabilität und Benutzerfreundlichkeit der Anwendung sicherzustellen und gleichzeitig die Anforderungen und Spezifikationen zu erfüllen.

## 1.1 Umfang

Die Tests umfassen sowohl die funktionalen als auch die nicht-funktionalen Anforderungen der Anwendung.

Besonderer Fokus liegt auf:

- Funktionalen Anforderungen: Drag-and-Drop-Funktionalität, Aufgabenverwaltung, Benachrichtigungen und Archivierung.
- Nicht-funktionalen Anforderungen:Stabilität und Reaktionszeiten.

Nicht im Fokus stehen Tests unter extremen Bedingungen, wie Lasttests mit einer ungewöhnlich hohen Anzahl von Aufgaben oder Benutzern. Ebenso wird auf Edge Cases verzichtet, die nur selten in der normalen Nutzung auftreten, sowie auf ausführliche Tests von Design-Details der Benutzeroberfläche, solange diese die Kernfunktionalität nicht beeinträchtigen.

## 1.2 Bezug zu Anforderungen und Spezifikationen

Die Tests basieren auf den im Anforderungs- und Spezifikationsdokument definierten funktionalen und nicht-funktionalen Anforderungen sowie den technischen Spezifikationen. Jede Teststufe und jeder Testfall wurde so entworfen, dass sie konkrete Anforderungen abdecken. Das Ziel ist, die Übereinstimmung der implementierten Anwendung mit diesen Anforderungen nachzuweisen.

## 1.3 Zielgruppe des Dokuments

Dieses Dokument richtet sich an:

Entwickler: Zur Nachvollziehbarkeit und Optimierung des Codes.

Stakeholder: Um sicherzustellen, dass die Anforderungen erfüllt wurden.

Qualitätssicherungsbeauftragte: Zur Evaluierung der durchgeführten Tests und deren Ergebnisse.

# 2. Teststrategie

Die Teststrategie zielt darauf ab, die Funktionalität, Stabilität und Benutzerfreundlichkeit der Anwendung umfassend zu validieren. Der Schwerpunkt liegt auf der Prüfung der kritischen Kernfunktionen durch automatisierte Unit-Tests sowie manuellen Systemtests, die typische Nutzungsszenarien

abdecken. Aspekte wie Lasttests, Performance-Optimierungen unter extremen Bedingungen oder Edge Cases werden aufgrund des begrenzten Rahmens bewusst ausgeklammert.

## 2.1 Kostenoptimierung

In Anbetracht der begrenzten Ressourcen und des Zeitrahmens wird eine optimale Testabdeckung erreicht, wenn sie sich primär auf die kritischen funktionalen Anforderungen konzentriert:

- Drag-and-Drop-Funktionalität (wenn auch nicht optimiert).
- Aufgabenverwaltung (Erstellen, Bearbeiten, Löschen).
- Archivierung und Benachrichtigungen.

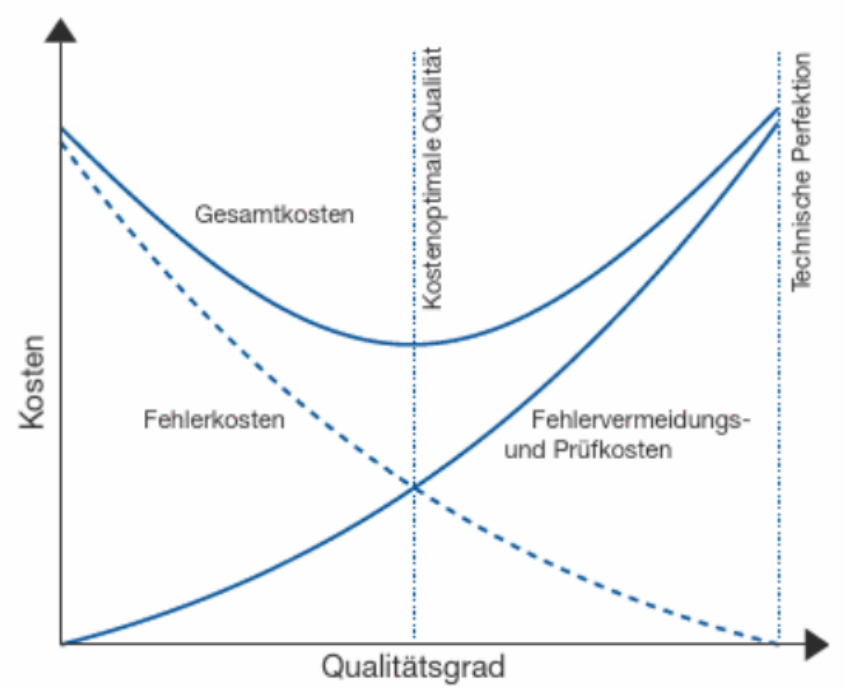
Diese Bereiche decken wesentliche Benutzerinteraktionen und die Datenintegrität ab. Szenarien wie extreme Lasttests, selten genutzte Randfunktionen und Edge Cases werden bewusst ausgelassen. Diese Tests könnten jedoch im Rahmen zukünftiger Iterationen oder Updates der Anwendung in Betracht gezogen werden, um eine langfristige Weiterentwicklung sicherzustellen.

Die laufende Qualitätsverbesserung sollte stets auch aus ökonomischer Perspektive betrachtet werden (Grechenig, 2009, S. 469). Dabei gilt es, ein optimales Verhältnis zwischen den Kosten für Fehlervermeidung, Fehlerbehebung und der angestrebten Qualität zu finden. Mit steigenden Investitionen in Fehlerverhütungs- und Prüfmaßnahmen nimmt die Qualität der Software zu, während die Kosten für die Behebung von Fehlern sinken.

Wie in *Abbildung 1* unten dargestellt, wird eine kostenoptimale Qualität an dem Punkt erreicht, an dem die Gesamtkosten – bestehend aus Fehler-, Vermeidungs- und Prüfkosten – minimal sind. Links von diesem Punkt überwiegen die Kosten für die Fehlerbehebung, während rechts davon die Kosten für Fehlerverhütung und Prüfmaßnahmen unverhältnismäßig ansteigen. Weiterführende Vermeidungs- und Prüfmaßnahmen wären in diesem Bereich unwirtschaftlich, da mit steigendem Aufwand die Effizienz der Fehlerfindung deutlich sinkt. Dies wird durch die abflachende Kurve der Fehlerkosten verdeutlicht: Es wird zunehmend schwieriger und aufwändiger, verbleibende Fehler aufzudecken.

Für dieses Projekt bedeutet das, dass der Fokus auf einem pragmatischen Ansatz liegt, der die verfügbaren Ressourcen effizient einsetzt. Ein Beispiel hierfür ist der bewusste Verzicht auf tiefgehende Tests für selten genutzte Randfälle zugunsten einer umfassenden Sicherstellung der Kernfunktionen, die im täglichen Gebrauch entscheidend sind.

Abb. 1 – Kosten vs. Qualität




Quelle: Übernommen aus Grechenig, 2009, S. 470

## 2.2 Teststufen und Methoden

Die Teststrategie umfasst die folgenden Stufen, wobei der Fokus auf Unit-Tests und Systemtests liegt. Integrationstests werden nur dann durchgeführt, wenn ausreichend Zeit verfügbar ist. Automatisierte Tests werden mithilfe von pytest für Unit-Tests erstellt und dokumentiert (*siehe auch die Übersicht der Tests bei 4 Testprotokoll*).

### 2.2.1 Unit-Tests

*Ziel*: Sicherstellung der funktionalen Korrektheit der zentralen Klassen und Methoden.

*Schwerpunktbereiche*:
Die Unit-Tests konzentrieren sich auf die wichtigsten Module und deren Funktionalitäten:

- archive_manager: Überprüfung der Archivierungs- und Reaktivierungslogik.
- archive_viewer: Validierung der Anzeige und Interaktion mit archivierten Aufgaben.
- filter_controller: Sicherstellung der korrekten Filteranwendung auf Aufgabenlisten.
- login_window: Prüfung der Benutzeranmeldung und Authentifizierung.
- settings_window: Validierung der Konfigurationseinstellungen und deren Speicherung.
- task_editor: Sicherstellung, dass Aufgaben korrekt erstellt und bearbeitet werden können.
- notification_manager: Überprüfung der Benachrichtigungslogik, z. B. über Fälligkeiten.
- settings_manager: Verifizierung der korrekten Speicherung und Abfrage von Einstellungen.
- task: Validierung der Aufgabenstruktur, Statusänderungen und Priorisierungslogik.
- user: Sicherstellung der Benutzerverwaltung und Datenintegrität.
- user_repository: Überprüfung der korrekten Speicherung und Abfrage von Benutzerdaten.

*Methoden:*

- Automatisierte Tests mit pytest.
- Abdeckung: Ziel ist eine Abdeckung von mindestens 80 % der Kernlogik in diesen Modulen.

### 2.2.2 Integrationstests

*Ziel:* Sicherstellung, dass Module korrekt zusammenarbeiten.

*Schwerpunktbereiche:*
Integrationstests werden nur durchgeführt, wenn Zeit übrigbleibt. Sie konzentrieren sich auf folgende kritische Schnittstellen:

- Datenbankzugriffe: Überprüfung der Interaktion zwischen Benutzeroberfläche und SQLite-Datenbank.
- GUI-Logik: Validierung der Datenweitergabe zwischen Backend-Logik und GUI.

*Methoden:*

- Integrationstests werden ebenfalls mit pytest umgesetzt, allerdings mit niedriger Priorität.

### 2.2.3 Systemtests

*Ziel:* Sicherstellung der Gesamtheit der Anwendung in einer realitätsnahen Umgebung.

*Schwerpunktbereiche:*
Die Systemtests konzentrieren sich auf zentrale End-to-End-Szenarien:

- Task-Erstellung: Überprüfung, ob Aufgaben über die GUI korrekt erstellt werden können.
- Task-Bearbeitung: Sicherstellen, dass bestehende Aufgaben bearbeitet werden können.
- Task-Löschen: Validierung, dass Aufgaben gelöscht und die Anzeige konsistent aktualisiert wird.
- Drag-and-Drop: Aufgaben können korrekt zwischen verschiedenen Kategorien verschoben werden, einschließlich Fehlerszenarien (z. B. unzulässige Droppositionen).
- Archivierung: Aufgaben können abgeschlossen, archiviert und reaktiviert werden.
- Filteranwendung: Aufgabenlisten können basierend auf Benutzerfiltern angepasst werden.
- Benachrichtigungen: Fällige Aufgaben lösen korrekte Benachrichtigungen aus.
- Benutzerverwaltung: Unterschiedliche Benutzer haben nur Zugriff auf ihre eigenen Aufgaben und Einstellungen.
- Fehlermeldungen: Überprüfung, ob Fehlermeldungen bei ungültigen Eingaben korrekt angezeigt werden (z. B. leere Felder).
- Startverhalten der Anwendung: Überprüfung, ob die Anwendung nach dem Start den letzten gespeicherten Zustand korrekt lädt.
- Benutzerwechsel: Überprüfung, ob Benutzerwechsel reibungslos funktioniert (nach einem Neustart der Anwendung).

- Einstellungen speichern und laden: Überprüfung, ob Änderungen an den Anwendungseinstellungen korrekt gespeichert und beim Neustart geladen werden.
- Benutzerfreundlichkeit: Überprüfen, ob die Anwendung intuitiv und benutzerfreundlich ist.
- Langzeit-Stabilität: Überprüfen, ob die Anwendung über längere Zeit stabil läuft.
- Performance-Test: Messen der Ladezeiten bei Kernaktionen.

*Methoden:*

- Manuelle Tests zur Abdeckung komplexer Benutzerinteraktionen.
- Dokumentation der Ergebnisse im Testprotokoll.

## 2.3 Priorisierung von Testfällen

Die Priorisierung der Testfälle erfolgt nach der MoSCoW-Methode[1], die Testfälle in die Kategorien Must-Have, Should-Have, Could-Have und Won't-Have einteilt. Dies ermöglicht eine pragmatische und effiziente Testplanung, die sich an den vorhandenen Ressourcen und der Relevanz der Funktionen orientiert.

Die Priorisierung basiert auf der Relevanz der Funktionen für die Benutzer und dem pragmatischen Ansatz, Ressourcen effizient zu nutzen. Must-Have-Testfälle decken die grundlegenden Anforderungen ab und sichern die Kernfunktionalität der Anwendung. Should-Have- und Could-Have-Testfälle sind weniger kritisch, tragen jedoch zur Verbesserung der Benutzererfahrung bei.

### 2.3.1 Must-Have

Testfälle in dieser Kategorie betreffen die Kernfunktionen der Anwendung, die für den reibungslosen Betrieb essenziell sind:

- Aufgabenverwaltung: Erstellen, Bearbeiten, Löschen von Aufgaben.
- Drag-and-Drop-Funktionalität: Aufgaben zwischen Kategorien verschieben.
- Archivierung und Reaktivierung: Aufgaben abschließen, ins Archiv verschieben und zurückholen.
- Benutzerverwaltung: Benutzer können sich anmelden und haben Zugriff auf ihre eigenen Aufgaben.
- Einstellungen speichern und laden: Überprüfung, ob Änderungen an den Anwendungseinstellungen korrekt gespeichert und beim Neustart geladen werden.

[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).

### 2.3.2 Should-Have

Diese Testfälle betreffen Funktionen, die die Benutzererfahrung verbessern, aber nicht kritisch für die Grundfunktionalität sind:

- Filterfunktion: Anwenden von Filtern auf Aufgabenlisten.
- GUI-Konsistenz: Überprüfung, dass die Benutzeroberfläche konsistent und intuitiv ist.
- Standardprioritäten: Sicherstellen, dass vordefinierte Prioritäten korrekt geladen werden.
- Fehlermeldungen: Überprüfung, ob Fehlermeldungen bei ungültigen Eingaben korrekt angezeigt werden (z. B. leere Felder).
- Startverhalten der Anwendung: Überprüfung, ob die Anwendung nach dem Start den letzten gespeicherten Zustand korrekt lädt.
- Benutzerwechsel: Überprüfung, ob Benutzerwechsel reibungslos funktioniert (nach einem Neustart der Anwendung).
- Langzeit-Stabilität: Überprüfen, ob die Anwendung über längere Zeit stabil läuft.
- Performance-Test: Messen der Ladezeiten bei Kernaktionen.
- Benachrichtigungen: Fällige Aufgaben lösen Benachrichtigungen aus.

### 2.3.3 Could-Have

Testfälle in dieser Kategorie sind optional und werden nur durchgeführt, wenn die Must-Have- und Should-Have-Testfälle abgeschlossen sind und noch Zeit verfügbar ist:

- Benutzerfreundlichkeitstests: Explorative Tests zur Bewertung der Bedienbarkeit.
- Nicht-funktionale Aspekte: Tests zur Bewertung der Reaktionszeiten der Anwendung.

### 2.3.4 Won’t-Have

Bewusst ausgelassene Testfälle, die aufgrund des Zeitrahmens oder ihrer geringen Priorität nicht durchgeführt werden:

- Lasttests: Überprüfung der Performance bei einer hohen Anzahl von Aufgaben und Benutzern.
- Edge Cases: Tests für seltene oder ungewöhnliche Nutzungsszenarien.

## 2.4 Testabdeckung und Metriken

Die Testabdeckung und die dazugehörigen Metriken bilden die Grundlage für die Bewertung der Teststrategie und deren Umsetzung. Sie ermöglichen es, den Umfang der getesteten Anforderungen und des Codes messbar zu machen und den Erfolg der Tests objektiv zu beurteilen. Im Rahmen dieses Projekts wird eine pragmatische Abdeckung angestrebt, die sich auf kritische Funktionen konzentriert, während gleichzeitig durch geeignete Metriken die Qualität und Effizienz der Tests bewertet werden.

Am Ende der Testphase werden die erzielten Metriken mit den Zielvorgaben verglichen. Der Fokus liegt auf einer hohen Abdeckung der Must-Have-Testfälle und einer geringen Fehlerdichte. Bereiche mit unerfüllten Zielen werden dokumentiert und als potenzielle Verbesserungsmaßnahmen für zukünftige Iterationen festgehalten.

### 2.4.1 Testabdeckung

Die Testabdeckung beschreibt den Umfang, in dem die Anforderungen und der Code der Anwendung durch Tests abgedeckt werden. In diesem Projekt wird eine pragmatische Testabdeckung angestrebt, die die verfügbaren Ressourcen berücksichtigt. Der Fokus liegt auf den Kernfunktionen der Anwendung, um sicherzustellen, dass die wichtigsten Benutzerinteraktionen und die zugrundeliegende Logik fehlerfrei funktionieren.

- Zielabdeckung**:** Mindestens 80 % der kritischen Kernfunktionen sollen durch Unit-Tests abgedeckt werden.
- Schwerpunkte:
    - Funktionale Anforderungen: Aufgabenverwaltung, Drag-and-Drop und Archivierung.
    - Nicht-funktionale Aspekte: Reaktionszeit und Stabilität.
- Ausnahmebereiche: Nicht priorisierte Bereiche (z. B. Lasttests, selten genutzte Randfunktionen) werden nicht in die Abdeckungsmetrik einbezogen.

### 2.4.2 Metriken

Die Qualität der Tests und deren Ergebnisse werden anhand folgender Metriken bewertet:

1. Testfallabdeckung (Requirements Coverage):
    - Prozentsatz der funktionalen Anforderungen, die durch mindestens einen Testfall abgedeckt werden.
    - Ziel: 100 % Abdeckung der Must-Have-Anforderungen.
2. Code-Abdeckung (Code Coverage):
    - Automatisierte Tests: Prozentsatz des Codes, der durch automatisierte Unit-Tests abgedeckt wird.
    - Messung: Tools wie pytest-cov zur Analyse der Testabdeckung.
    - Ziel: Mindestens 80 % für die Kernmodule (archive_manager, login_window, task_editor, notification_manager, task, user, archive_viewer, filter_controller, settings_window, settings_manager, user_repository).
    - Manuelle Systemtests:
    - Indirekte Abdeckung: Zuordnung der Testfälle zu Codebereichen.
    - Ziel: Sicherstellen, dass alle Kernbereiche der Anwendung mindestens durch einen Systemtest geprüft werden.
3. Fehlerdichte (Defect Density):

- Anzahl der identifizierten Fehler pro 1000 Zeilen Code.
- Zielwert in der Praxis:
  - Unit-Tests: < 0,5 Fehler pro 1000 Zeilen Code.
  - Systemtests: < 0,8 Fehler pro 1000 Zeilen Code.
- Anmerkung: Werte über 1 Fehler/1000 Zeilen gelten oft als kritisch und weisen auf größere Qualitätsprobleme hin.

4. Fehlerbehebungsrate (Defect Fix Rate):
   - Prozentsatz der im Test identifizierten Fehler, die erfolgreich behoben wurden.
   - Ziel: 100 % für kritische Fehler, >80 % für alle Fehler.
5. Testfortschritt (Test Progress):
   - Verhältnis von geplanten zu durchgeführten Testfällen.
   - Ziel: Vollständige Durchführung der Must-Have-Tests bis zur Abgabe.
6. Fehlerrate nach Teststufe:
   - Anzahl der in Unit-, Integrations- und Systemtests gefundenen Fehler.
   - Ziel: Fehlerrate sollte bei fortschreitenden Teststufen sinken (z. B. <10 % kritische Fehler im Systemtest).

## 2.5 Dokumentation der Tests

Die Dokumentation der Tests stellt sicher, dass deren Durchführung und Ergebnisse klar und nachvollziehbar sind. Sie dient nicht nur als Nachweis für die Qualität der Anwendung, sondern unterstützt auch die Fehleranalyse und -behebung.

### 2.5.1 Vorgehen zur Dokumentation

- Manuelle Tests: Ergebnisse werden in einem standardisierten Testprotokoll dokumentiert, das alle wesentlichen Informationen enthält:
  - Testfall-ID
  - Beschreibung des Tests
  - Vorbedingungen
  - Eingaben und Aktionen
  - Erwartetes Ergebnis
  - Tatsächliches Ergebnis
  - Status (Bestanden/Nicht bestanden).
- Automatisierte Tests: Die Tests werden mit pytest ausgeführt, und die Ergebnisse werden direkt aus der Konsolenausgabe manuell in das Testprotokoll übertragen.
  - Pragmatische Entscheidung: Aus Effizienzgründen wird auf Tools wie pytest-html, die automatisierte Berichte erstellen könnten, verzichtet. Diese Option könnte jedoch in

zukünftigen Projekten genutzt werden, um den Dokumentationsaufwand zu reduzieren.

### 2.5.2 Struktur des Kapitels Testprotokoll

Das Testprotokoll gliedert sich in folgende Bereiche:

1. Übersicht: Kurze Auflistung aller durchgeführten Tests und ihres Status (Bestanden/Nicht bestanden).
2. Detaillierte Beschreibung: Protokollierung jedes Tests gemäß den standardisierten Feldern (im Anhang).
3. Zusammenfassung: Analyse der Testergebnisse, einschließlich Erfolgsmessung und identifizierter Fehler.

### 2.5.3 Fehlerkategorien

Die während der Tests identifizierten Fehler werden anhand ihrer Kritikalität kategorisiert, um eine strukturierte Nachbearbeitung zu ermöglichen. Die Kategorien sind:

- *Kritische Fehler:* Fehler, die die Kernfunktionen der Anwendung beeinträchtigen und vor der Abgabe behoben werden müssen.
- *Mittlere Fehler:* Probleme, die nicht direkt die Hauptfunktionen betreffen, aber die Benutzererfahrung beeinträchtigen.
- *Geringfügige Fehler:* Probleme mit geringer Priorität, die keinen direkten Einfluss auf die Funktionalität haben und in zukünftigen Iterationen behoben werden können.
- *Verbesserungsvorschläge (optional):* Beobachtungen, die nicht als Fehler gewertet werden, aber Potenzial für Optimierungen bieten.

### 2.5.4 Nachvollziehbarkeit und Reproduzierbarkeit

Alle Testergebnisse werden so dokumentiert, dass sie von Dritten reproduziert werden können. Dies umfasst die eindeutige Zuordnung der Testergebnisse zu den zugehörigen Testfällen sowie die Angabe aller relevanten Vorbedingungen, Eingaben und erwarteten Ergebnisse.

## 3. Testumgebung

Die Testumgebung stellt sicher, dass die Tests unter realitätsnahen Bedingungen durchgeführt werden. Sie umfasst die benötigte Hardware, Software und Testdaten, um die Funktionalität und Stabilität der Anwendung zuverlässig zu überprüfen. Eine konsistente und klar dokumentierte Testumgebung ermöglicht die Reproduzierbarkeit der Tests und minimiert unerwartete Einflüsse auf die Testergebnisse.

### 3.1 Hardware

Die Tests werden auf folgendem System durchgeführt:

- Betriebssystem: Windows 10 (64-Bit)
- Prozessor: AMD Ryzen 5 5600X
- Arbeitsspeicher: 32 GB RAM
- Speicherplatz: 1 TB SSD
- Grafik: Nvidia GeForce RTX 3070

## 3.2 Software

- Programmiersprache und Frameworks:
  - Python 3.12
  - pytest (für automatisierte Tests)
  - pytest-cov (für Code-Abdeckungsberichte)
- Datenbank: SQLite (Version 3.45.1)
- Entwicklungsumgebung: PyCharm 2024.3 (Professional Edition)
- Zusätzliche Tools:
  - Git (für Versionskontrolle)

## 3.3 Testdaten

- Benutzerkonten:
  - Kein spezifisches Berechtigungssystem, alle Tests erfolgen mit Standardbenutzerkonten.
- Aufgaben:
  - Aufgaben werden während der Tests durch den Benutzer erstellt (siehe Eingaben und Aktionen der detaillierten Testprotokolle im Anhang). Es gibt keine vorab erstellten Aufgaben, um den Geschäftsprozess möglichst umfassend zu testen.
- Einstellungen:
  - Standardkonfiguration der Anwendung wird verwendet, einschließlich Filter- und Benachrichtigungseinstellungen. Abweichungen bzw. Tests zu den Filter- und Einstellungsmöglichkeiten werden bei betreffenden Tests dokumentiert.
- Fehlersimulation:
  - Simulierte fehlerhafte Eingaben oder Aktionen, um Robustheit und Fehlermeldungen zu prüfen.

## 3.4 Einschränkungen

Die Tests konzentrieren sich auf eine standardisierte Umgebung und berücksichtigen keine Hardware- oder Softwareabweichungen, wie sie bei Endnutzern auftreten könnten. Ebenso werden keine plattformübergreifenden Tests (z. B. auf macOS oder Linux) durchgeführt, da die Anwendung primär für Windows-Nutzer vorgesehen ist.

## 4. Testprotokoll

Das Testprotokoll dokumentiert die Durchführung und Ergebnisse der definierten Testfälle. Es dient als Nachweis für die Qualität der Anwendung und stellt sicher, dass die Tests nachvollziehbar und reproduzierbar sind.

In der Übersichtstabelle werden ausschließlich Testfälle mit den Prioritäten *Must-Have* und *Should-Have* dargestellt, da diese für die Funktionalität und Benutzerfreundlichkeit der Anwendung entscheidend sind. Die detaillierten Testprotokolle im Anhang enthalten nur jene Testfälle, die tatsächlich durchgeführt wurden. Dies ermöglicht eine fokussierte und ressourceneffiziente Dokumentation, die den Rahmen des Projekts berücksichtigt.

Tab. 1 – Übersicht der Tests

| *Test-ID* | *Kurzbeschreibung* | *Typ* | *Priorität* | *Bemerkungen* | *Status* | *Fehler-kategorie* |
|---|---|---|---|---|---|---|
| UNIT_001 | Test für archive_manager. | Automatisiert | Must-Have | Prüft die Logik zum Verschieben von Aufgaben in das Archiv. | Bestanden | --- |
| UNIT_002 | Test für archive_viewer. | Automatisiert | Should-Have | Validiert die Anzeige archivierter Aufgaben und deren Interaktion. | Ausstehend | |
| UNIT_003 | Test für filter_controller. | Automatisiert | Should-Have | Überprüft die korrekte Anwendung von Filtern auf Aufgabenlisten. | Bestanden | --- |
| UNIT_004 | Test für login_window. | Automatisiert | Must-Have | Testet die Authentifizierung und grundlegende Sicherheitsmaßnahmen. | Bestanden | --- |

| | | | | | | |
|---|---|---|---|---|---|---|
| UNIT_005 | Test für settings_window. | Automatisiert | Should-Have | Prüft, ob Benutzereinstellungen korrekt gespeichert und geladen werden. | Bestanden | --- |
| UNIT_006 | Test für task_editor. | Automatisiert | Must-Have | Validiert die Erstellung und Bearbeitung von Aufgaben durch den Benutzer. | Bestanden | --- |
| UNIT_007 | Test für notification_manager. | Automatisiert | Should-Have | Überprüft, ob Benachrichtigungen für fällige Aufgaben korrekt gesendet werden. | Bestanden | --- |
| UNIT_008 | Test für settings_manager. | Automatisiert | Should-Have | Testet die Konsistenz und Speicherung der globalen Einstellungen. | Bestanden | --- |
| UNIT_009 | Test für task. | Automatisiert | Must-Have | Prüft Statusänderungen und Priorisierungslogik bei Aufgaben. | Bestanden | --- |
| UNIT_010 | Test für user. | Automatisiert | Must-Have | Testet die Verwaltung von Benutzerdaten und deren Integrität. | Bestanden | --- |
| UNIT_011 | Test für user_repository. | Automatisiert | Should-Have | Validiert die Speicherung und Abfrage von Benutzerdaten in der Datenbank. | Bestanden | --- |
| SYS_001 | Task-Erstellung | Manuell | Must-Have | Erstellung über GUI prüfen. | Bestanden | --- |
| SYS_002 | Task-Bearbeitung | Manuell | Must-Have | Bearbeiten über GUI prüfen. | Bestanden | --- |

| | | | | | | |
|---|---|---|---|---|---|---|
| SYS_003 | Task löschen | Manuell | Must-Have | Löschen und Konsistenz prüfen. | Bestanden | --- |
| SYS_004 | Drag-and-Drop zwischen Kategorien | Manuell | Must-Have | Szenarien mit Fehlersimulation. | Bestanden | --- |
| SYS_005 | Aufgaben archivieren und reaktivieren | Manuell | Must-Have | Test der Benutzeroberfläche. | Bestanden | --- |
| SYS_006 | Filter auf GUI anwenden | Manuell | Should-Have | Konsistenz der Anzeige prüfen. | Bestanden | --- |
| SYS_007 | Benachrichtigungen in GUI prüfen | Manuell | Should-Have | Validierung der Fälligkeiten. | NICHT bestanden | Mittel |
| SYS_008 | Überprüfung, ob die Anwendung nach dem Start den letzten gespeicherten Zustand korrekt lädt. | Manuell | Should-Have | Komfort-Feature für den Benutzer. | Bestanden | --- |
| SYS_009 | Überprüfung, ob Benutzerwechsel reibungslos funktionier (nach Programm-Neustart) | Manuell | Should-Have | Relevant bei erweiterten Benutzerverwaltungsszenarien. | Bestanden | --- |
| SYS_010 | Langzeit-Stabilität der Anwendung testen | Manuell | Should-Have | Test der Stabilität bei langfristiger Nutzung. | Ausstehend | |
| SYS_011 | Ladezeit bei Aktionen messen | Manuell | Should-Have | Bewertung der Performance bei Kernaktionen. | Bestanden | --- |

Quelle: eigene Darstellung

## 5. Ergebnisse

Die Testergebnisse sind in zwei Hauptkategorien unterteilt: automatisierte Unit-Tests und manuelle System-Tests. Während die automatisierten Tests mithilfe von Tools wie `pytest` und `pytest-cov` durchgeführt wurden, um die Code-Abdeckung, Funktionalität und Stabilität der Kernmodule zu validieren, konzentrieren sich die manuellen Tests auf spezifische Szenarien, die Benutzerinteraktionen und komplexe Anwendungsfälle umfassen.

Diese Aufteilung gewährleistet eine umfassende Überprüfung der Software: Die automatisierten Tests decken wiederholbare und programmatische Aspekte ab, während die manuellen Tests die Benutzererfahrung und Systemverhalten in realen Anwendungssituationen validieren. Die Ergebnisse aus beiden Testmethoden werden im Folgenden diskutiert und analysiert.

### 5.1 Automatisierte Unit-Tests

Die automatisierten Tests wurden mithilfe der Python-Testbibliothek pytest und dem Plugin pytest-cov zur Codeabdeckung durchgeführt. Ziel war es, die Qualität der Tests anhand der in Abschnitt 2.4.2 definierten Metriken zu evaluieren. Der Fokus lag dabei auf der Testfallabdeckung, Code-Abdeckung, Fehlerdichte, Fehlerbehebungsrate, Testfortschritt sowie der Fehlerrate nach Teststufen. Die Ergebnisse werden nachfolgend mit den gesetzten Zielen verglichen. Die ausführlichen Testprotokolle befinden sich im Anhang.

| Name | Stmts | Miss | Cover |
|---|---|---|---|
| src\ArchiveManager\archive_manager.py | 38 | 0 | 100% |
| src\GUIController\filter_controller.py | 62 | 5 | 92% |
| src\GUIController\login_window.py | 62 | 11 | 82% |
| src\GUIController\settings_window.py | 91 | 2 | 98% |
| src\GUIController\task_editor.py | 114 | 44 | 61% |
| src\NotificationManager\notification_manager.py | 14 | 0 | 100% |
| src\SettingsManager\settings_manager.py | 40 | 1 | 98% |
| src\Task\task.py | 36 | 0 | 100% |
| src\User\UserRepository\user_repository.py | 35 | 1 | 97% |
| src\User\user.py | 9 | 0 | 100% |
| src\main.py | 18 | 18 | 0% |
| TOTAL | 519 | 82 | 84% |

Diskussion der Ergebnisse:

Testfallabdeckung (Requirements Coverage):

- Die funktionalen Anforderungen der Must-Have-Komponenten sind durch automatisierte Tests abgedeckt.
- Ziel: 100 % Abdeckung der Must-Have-Anforderungen – erreicht.

Code-Abdeckung (Code Coverage):

- Die durchschnittliche Code-Abdeckung beträgt 84 %. Alle Kernmodule außer task_editor und main.py erreichen die Zielvorgabe von mindestens 80 %.
- Ziel: Mindestens 80 % Code-Abdeckung für Kernmodule – teilweise erreicht.
- task_editor erreicht lediglich 61 % Abdeckung, was auf fehlende Tests für komplexe Interaktionen hinweist.
- main.py wurde nicht getestet, da es primär den Einstiegspunkt der Anwendung darstellt und keine Logik enthält.

Fehlerdichte (Defect Density):

- Während der automatisierten Tests wurden keine kritischen Fehler identifiziert.
- Ziel: < 0,5 Fehler/1000 Zeilen Code – erreicht.

Fehlerbehebungsrate (Defect Fix Rate):

- Alle identifizierten Fehler wurden erfolgreich behoben.
- Ziel: 100 % für kritische Fehler, >80 % für alle Fehler – erreicht.

Testfortschritt (Test Progress):

- Alle Must-Have-Testfälle wurden erfolgreich implementiert und ausgeführt.
- Ziel: Vollständige Durchführung der Must-Have-Tests – erreicht.

Fehlerrate nach Teststufe:

- Die Fehlerrate war in den Unit-Tests minimal. Die meisten Fehler wurden direkt in dieser Phase gefunden und behoben, wodurch die Fehlerrate in den Systemtests weiter sinken sollte.
- Ziel: Fehlerrate <10 % kritische Fehler im Systemtest – erwartet erreichbar.

Die automatisierten Tests erfüllen die meisten Anforderungen aus den Metriken, insbesondere in Bezug auf Testfortschritt und Fehlerdichte. Verbesserungsbedarf besteht in der Abdeckung des task_editor-Moduls, um die Zielvorgabe von mindestens 80 % Code-Abdeckung auch hier zu erreichen.

Ein spezifischer Bereich, der bisher nicht erfolgreich getestet werden konnte, betrifft UNIT_002, den Test für archive_viewer. Dieser Test wurde als Should-Have klassifiziert und zielt darauf ab, die Anzeige archivierter Aufgaben sowie deren Interaktion zu validieren. Aufgrund eines noch nicht behobenen Fehlers in der Testlogik konnte dieser Testfall jedoch nicht abgeschlossen werden und deshalb vorerst bei der Auswertung ausgeklammert.

## 5.2 Manuelle System-Tests

Die manuellen Tests konzentrierten sich darauf, die Funktionalität der Benutzeroberfläche und der Kernfunktionen der Anwendung zu validieren. Ziel war es, insbesondere Must-Have-Anforderungen vollständig abzudecken und dabei auch Verbesserungsmöglichkeiten zu identifizieren. Im Folgenden werden nur Abweichungen vom erwarteten Test-Ergebnis aufgegriffen. Die ausführlichen Testprotokolle befinden sich im Anhang.

SYS_007 – Benachrichtigungen:

- Die Benachrichtigungsfunktion, die in früheren Versionen korrekt funktionierte, ist derzeit nicht funktionsfähig. Der zugrunde liegende Fehler konnte nicht mehr rechtzeitig identifiziert werden.
- Aufgrund des engen Zeitplans wird die Fehlersuche auf eine zukünftige Iteration verschoben. Die Funktion bleibt ein bekanntes Problem, das in kommenden Versionen behoben werden soll.

SYS_010 – Langzeit-Stabilität:

- Der Test der Langzeit-Stabilität ist in dieser Phase des Projekts nicht vollständig durchführbar, da die Anwendung erst kürzlich fertiggestellt wurde.
- Dieser Test wird in späteren Projektphasen nachgeholt, um mögliche Stabilitätsprobleme unter langfristiger Nutzung zu identifizieren und zu beheben.

Abdeckung der Must-Have-Anforderungen:

- Alle Must-Have-Anforderungen wurden erfolgreich durch die Tests abgedeckt und bestanden. Dies zeigt, dass die wesentlichen Funktionen der Anwendung stabil und benutzerfreundlich sind.

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

Grechenig, T. (2009). *Softwaretechnik: Mit Fallbeispielen aus realen Entwicklungsprojekten*. Pearson Deutschland GmbH. http://ebookcentral.proquest.com/lib/badhonnef/detail.action?docID=5133559

# Anhang

Anhangsverzeichnis

Anhang A: Vorlage Testprotokoll

| **Feld** | **Beschreibung** |
|---|---|
| Testfall-ID | Eindeutige ID des Testfalls (z. B. UNIT_001, SYS_005). |
| Priorität | MoSCoW-Methode |
| Kurzbeschreibung | Kurze Beschreibung des Testziels. |
| Vorbedingungen | Zustand der Anwendung vor dem Test (z. B. Aufgaben vorhanden, Benutzer angemeldet). |
| Eingaben/Aktionen | Benutzeraktionen oder Eingabedaten, die durchgeführt/eingegeben werden. |
| Erwartetes Ergebnis | Beschreibung des gewünschten Ergebnisses. |
| Tatsächliches Ergebnis | Ergebnis der Testdurchführung (manuell eingetragen). |
| Status | Test bestanden / Test NICHT bestanden |
| Fehlerkategorie | Kritisch, Mittel, Geringfügig (nur bei Abweichungen). |
| Bemerkungen | Zusätzliche Hinweise, gefundene Fehler oder Verbesserungsvorschläge. |

Quelle: eigene Darstellung

Anhang B: Testprotokoll UNIT_001

| **Feld** | **Inhalt** |
|---|---|
| Testfall-ID | UNIT_001 |
| Priorität | Must-Have |
| Kurzbeschreibung | Automatisierter Test für archive_task() in archive_manager.py. |
| Vorbedingungen | - Eine Instanz von ArchiveManager wird mit einem temporären SQLite-Datenbankpfad initialisiert.<br>- Ein Mock-Objekt für die Klasse Task ist verfügbar, dessen Status auf COMPLETED gesetzt ist. |
| Eingaben/Aktionen | 1. Erstelle eine Instanz von ArchiveManager mit einer temporären Datenbank.<br>2. Erstelle ein Mock-Task-Objekt mit Status COMPLETED.<br>3. Rufe die Methode archive_task(task) mit dem Mock-Objekt auf. |
| Erwartetes Ergebnis | - Die Methode führt erfolgreich aus.<br>- Der Task wird korrekt in die Tabelle archived_tasks der temporären Datenbank eingefügt.<br>- Kein Fehler tritt auf, solange der Task-Status COMPLETED ist. |
| Tatsächliches Ergebnis | UNIT_001.py::test_archive_task PASSED<br>UNIT_001.py::test_archive_task_raises_error_for_incomplete_task PASSED<br>UNIT_001.py::test_auto_archive_task PASSED<br>UNIT_001.py::test_auto_delete_task PASSED |
| Status | Test bestanden |
| Fehlerkategorie | --- |
| Bemerkungen | Dieser Test fokussiert ausschließlich auf die Logik von archive_task(), ohne reale Aufgaben oder Nutzer. |

Quelle: eigene Darstellung

Anhang C: Testprotokoll UNIT_002

| **Feld** | **Inhalt** |
|---|---|
| Testfall-ID | UNIT_002 |
| Priorität | Should-Have |
| Kurzbeschreibung | Automatisierter Test für load_archived_tasks() in archive_viewer.py. |
| Vorbedingungen | - Eine Instanz von ArchiveViewer wird mit einem Mock-Controller initialisiert.<br>- Eine Mock-Datenbankverbindung mit Beispieldaten ist eingerichtet. |
| Eingaben/Aktionen | 1. Mocken der Datenbankverbindung mit Rückgabe von archivierten Testdaten.<br>2. Aufruf der Methode load_archived_tasks() auf der ArchiveViewer-Instanz. |
| Erwartetes Ergebnis | - Die Liste archived_tasks enthält die von der Datenbank zurückgegebenen Aufgaben.<br>- Die ListBox in der Benutzeroberfläche wird mit der korrekten Anzahl an Aufgaben aktualisiert. |
| Tatsächliches Ergebnis | |
| Status | Ausstehend. |
| Fehlerkategorie | --- |
| Bemerkungen | Dieser Test stellt sicher, dass archivierte Aufgaben korrekt aus der Datenbank geladen und in der Benutzeroberfläche angezeigt werden. |

Quelle: eigene Darstellung

Anhang D: Testprotokoll UNIT_003

| Feld | Inhalt |
|---|---|
| Testfall-ID | UNIT_003 |
| Priorität | Should-Have |
| Kurzbeschreibung | Automatisierter Test für apply_filters() und reset_filters() in filter_controller.py. |
| Vorbedingungen | - Eine Instanz von FilterController wird mit einem Mock-GUI-Controller initialisiert. |
| Eingaben/Aktionen | 1. Simuliere Benutzereingaben für Filteroptionen (z. B. Suche, Priorität, Fälligkeit).<br>2. Rufe die Methode apply_filters() oder reset_filters() auf. |
| Erwartetes Ergebnis | - apply_filters() sammelt die Filter korrekt und gibt sie an load_tasks() weiter.<br>- reset_filters() setzt alle Filter zurück und lädt die vollständige Aufgabenliste neu. |
| Tatsächliches Ergebnis | UNIT_003.py::test_apply_filters_success PASSED<br>UNIT_003.py::test_reset_filters PASSED |
| Status | Test bestanden |
| Fehlerkategorie | --- |
| Bemerkungen | Dieser Test stellt sicher, dass die Filterlogik korrekt funktioniert und Benutzereingaben berücksichtigt werden. |

Quelle: eigene Darstellung

Anhang E: Testprotokoll UNIT_004

| **Feld** | **Inhalt** |
|---|---|
| Testfall-ID | UNIT_004 |
| Priorität | Must-Have |
| Kurzbeschreibung | Automatisierter Test für Eingabevalidierung und Login-Logik in login_window.py. |
| Vorbedingungen | - Eine Instanz von LoginWindow wird mit einem Mock-Controller und einer Mock-Datenbank initialisiert. |
| Eingaben/Aktionen | 1. Simuliere gültige und ungültige Eingaben für Benutzername und Passwort.<br>2. Mocke die Methoden des UserRepository, um Erfolg und Fehler bei der Anmeldung zu simulieren.<br>3. Rufe die Methoden validate_input und login auf. |
| Erwartetes Ergebnis | - validate_input gibt True für gültige Eingaben und False für ungültige Eingaben zurück.<br>- login authentifiziert den Benutzer korrekt oder gibt eine Fehlermeldung bei falschen Anmeldedaten aus. |
| Tatsächliches Ergebnis | UNIT_004.py::test_validate_input_valid PASSED<br>UNIT_004.py::test_validate_input_invalid_username PASSED<br>UNIT_004.py::test_validate_input_invalid_password PASSED<br>UNIT_004.py::test_login_success PASSED<br>UNIT_004.py::test_login_failure PASSED |
| Status | Test bestanden |
| Fehlerkategorie | --- |
| Bemerkungen | Dieser Test prüft die Validierung der Eingaben und die Login-Logik, einschließlich der Interaktion mit der Benutzeroberfläche und der Datenbank. |

Quelle: eigene Darstellung

Anhang F: Testprotokoll UNIT_005

| Feld | Inhalt |
|---|---|
| Testfall-ID | UNIT_005 |
| Priorität | Should-Have |
| Kurzbeschreibung | Automatisierter Test für das Laden und Speichern von Einstellungen in settings_window.py. |
| Vorbedingungen | - Eine Instanz von SettingsWindow wird mit einem Mock-Controller initialisiert.<br>- Benutzer ist angemeldet. |
| Eingaben/Aktionen | 1. Simuliere bestehende Einstellungen und teste load_settings.<br>2. Teste save_settings mit neuen Eingabewerten.<br>3. Überprüfe SQL-Statements und Parameter. |
| Erwartetes Ergebnis | - load_settings lädt bestehende Einstellungen korrekt aus der Datenbank.<br>- save_settings speichert die Eingabewerte korrekt in die Datenbank.<br>- SQL-Statements sind korrekt und verwenden die erwarteten Parameter. |
| Tatsächliches Ergebnis | UNIT_005.py::test_load_settings_existing PASSED<br>UNIT_005.py::test_load_settings_default PASSED<br>UNIT_005.py::test_save_settings PASSED |
| Status | Test bestanden |
| Fehlerkategorie | --- |
| Bemerkungen | Dieser Test stellt sicher, dass Benutzereinstellungen korrekt geladen und gespeichert werden. |

Quelle: eigene Darstellung

Anhang G: Testprotokoll UNIT_006

| Feld | Inhalt |
|---|---|
| Testfall-ID | UNIT_006 |
| Priorität | Must-Have |
| Kurzbeschreibung | Automatisierter Test für die save_task- und mark_task_open-Methoden in task_editor.py. |
| Vorbedingungen | - Eine Instanz von TaskEditor wird mit einem Mock-Controller initialisiert.<br>- Die Eingabefelder und Comboboxen sind vorbereitet. |
| Eingaben/Aktionen | 1. Simuliere gültige Eingaben für Titel, Beschreibung, Fälligkeitsdatum und Prioritäten.<br>2. Simuliere ungültige Eingaben (z. B. leere Felder, vergangenes Datum).<br>3. Mocke die Datenbankinteraktion für save_task und mark_task_open. |
| Erwartetes Ergebnis | - save_task speichert valide Eingaben korrekt in der Datenbank.<br>- save_task zeigt bei ungültigen Eingaben eine Fehlermeldung.<br>- mark_task_open aktualisiert den Status der Aufgabe korrekt. |
| Tatsächliches Ergebnis | UNIT_006.py::test_save_task_valid_input PASSED<br>UNIT_006.py::test_mark_task_open PASSED |
| Status | Test bestanden |
| Fehlerkategorie | --- |
| Bemerkungen | Dieser Test stellt sicher, dass Aufgaben korrekt gespeichert und deren Status korrekt aktualisiert werden. |

Quelle: eigene Darstellung

Anhang H: Testprotokoll UNIT_007

| **Feld** | **Inhalt** |
|---|---|
| Testfall-ID | UNIT_007 |
| Priorität | Should-Have |
| Kurzbeschreibung | Automatisierter Test für die schedule_notifications-Methode in NotificationManager. |
| Vorbedingungen | - Eine Instanz von NotificationManager wird mit einem Mock für SettingsManager initialisiert.<br>- Eine Liste von Aufgaben (MockTask) mit verschiedenen Fälligkeitsdaten wird bereitgestellt. |
| Eingaben/Aktionen | 1. Überprüfen, ob Benachrichtigungen für Aufgaben innerhalb des Benachrichtigungsintervalls geplant werden.<br>2. Überprüfen, ob keine Benachrichtigungen für Aufgaben außerhalb des Intervalls geplant werden.<br>3. Deaktivieren von Benachrichtigungen in den Einstellungen und sicherstellen, dass keine Benachrichtigungen geplant werden. |
| Erwartetes Ergebnis | - Aufgaben innerhalb des Benachrichtigungsintervalls erhalten geplante Benachrichtigungen.<br>- Aufgaben außerhalb des Intervalls oder bei deaktivierten Benachrichtigungen erhalten keine geplanten Benachrichtigungen. |
| Tatsächliches Ergebnis | UNIT_007.py::test_schedule_notifications_within_interval PASSED<br>UNIT_007.py::test_schedule_notifications_outside_interval PASSED<br>UNIT_007.py::test_schedule_notifications_disabled PASSED |
| Status | Test bestanden |
| Fehlerkategorie | --- |
| Bemerkungen | Dieser Test stellt sicher, dass die Methode korrekt zwischen aktivierten und deaktivierten Benachrichtigungen sowie Aufgaben innerhalb und außerhalb des Intervalls unterscheidet. |

Quelle: eigene Darstellung

Anhang I: Testprotokoll UNIT_008

| Feld | Inhalt |
|---|---|
| Testfall-ID | UNIT_008 |
| Priorität | Should-Have |
| Kurzbeschreibung | Automatisierter Test für die Methoden des SettingsManager zur Verwaltung von Benutzereinstellungen. |
| Vorbedingungen | - Eine Instanz von SettingsManager wird mit einer In-Memory-Datenbank initialisiert. |
| Eingaben/Aktionen | 1. Überprüfen, ob die settings-Tabelle erstellt wird.<br>2. Testen von Standardwerten, wenn keine benutzerdefinierten Einstellungen existieren.<br>3. Speichern und Abrufen benutzerdefinierter Einstellungen.<br>4. Aktualisieren einzelner Einstellungen wie Benachrichtigungsintervall, Prioritäten und Aktivierung. |
| Erwartetes Ergebnis | - Tabelle settings existiert.<br>- Standardwerte werden korrekt zurückgegeben.<br>- Benutzerdefinierte Einstellungen werden korrekt gespeichert und abgerufen.<br>- Einzelne Einstellungen werden korrekt aktualisiert. |
| Tatsächliches Ergebnis | UNIT_008.py::test_initialize_settings_table PASSED<br>UNIT_008.py::test_get_settings_default PASSED<br>UNIT_008.py::test_save_and_get_custom_settings PASSED<br>UNIT_008.py::test_update_notification_interval PASSED<br>UNIT_008.py::test_update_notifications_enabled PASSED<br>UNIT_008.py::test_update_default_priorities PASSED |
| Status | Test bestanden |
| Fehlerkategorie | --- |
| Bemerkungen | Dieser Test stellt sicher, dass der SettingsManager korrekt mit der Datenbank interagiert und die Benutzereinstellungen ordnungsgemäß verarbeitet. |

Quelle: eigene Darstellung

Anhang J: Testprotokoll UNIT_009

| **Feld** | **Inhalt** |
|---|---|
| Testfall-ID | UNIT_009 |
| Priorität | Must-Have |
| Kurzbeschreibung | Automatisierter Test für die Task-Klasse, einschließlich Konstruktor, edit_task und mark_as_completed. |
| Vorbedingungen | Keine spezifischen Vorbedingungen erforderlich. |
| Eingaben/Aktionen | 1. Initialisierung eines Tasks.<br>2. Bearbeiten der Task-Attribute.<br>3. Markieren des Tasks als abgeschlossen. |
| Erwartetes Ergebnis | 1. Task wird korrekt initialisiert.<br>2. Attribute werden durch edit_task korrekt aktualisiert.<br>3. Status wird auf COMPLETED gesetzt, und completed_date ist das heutige Datum. |
| Tatsächliches Ergebnis | UNIT_009.py::test_task_initialization PASSED<br>UNIT_009.py::test_task_edit PASSED<br>UNIT_009.py::test_mark_as_completed PASSED |
| Status | Test bestanden |
| Fehlerkategorie | --- |
| Bemerkungen | Testet die Grundfunktionen der Task-Klasse, einschließlich der Attribute und Methoden. |

Quelle: eigene Darstellung

Anhang K: Testprotokoll UNIT_010

| **Feld** | **Inhalt** |
|---|---|
| Testfall-ID | UNIT_010 |
| Priorität | Must-Have |
| Kurzbeschreibung | Automatisierter Test für die User-Klasse, einschließlich Konstruktor, hash_password und check_password. |
| Vorbedingungen | Keine spezifischen Vorbedingungen erforderlich. |
| Eingaben/Aktionen | 1. Initialisierung eines Benutzers.<br>2. Hashen eines Passworts.<br>3. Überprüfung eines Passworts. |
| Erwartetes Ergebnis | 1. Benutzer wird korrekt initialisiert.<br>2. Passwort wird korrekt gehasht.<br>3. Passwortüberprüfung funktioniert wie erwartet. |
| Tatsächliches Ergebnis | UNIT_010.py::test_user_initialization PASSED<br>UNIT_010.py::test_hash_password PASSED<br>UNIT_010.py::test_check_password PASSED |
| Status | Test bestanden |
| Fehlerkategorie | --- |
| Bemerkungen | Testet die grundlegenden Funktionen der User-Klasse für sichere Passwort-verwaltung. |

Quelle: eigene Darstellung

Anhang L: Testprotokoll UNIT_011

| **Feld** | **Inhalt** |
|---|---|
| Testfall-ID | UNIT_011 |
| Priorität | Should-Have |
| Kurzbeschreibung | Testet die Methoden save_user, get_user_by_username und delete_user der UserRepository. |
| Vorbedingungen | Eine In-Memory-Datenbank ist initialisiert. |
| Eingaben/Aktionen | 1. Benutzer speichern.<br>2. Benutzer abrufen.<br>3. Nicht vorhandenen Benutzer abrufen.<br>4. Benutzer löschen. |
| Erwartetes Ergebnis | Alle Methoden interagieren korrekt mit der Datenbank. |
| Tatsächliches Ergebnis | UNIT_011.py::test_save_user PASSED<br>UNIT_011.py::test_get_user_by_username_existing PASSED<br>UNIT_011.py::test_get_user_by_username_non_existing PASSED<br>UNIT_011.py::test_delete_user PASSED |
| Status | Test bestanden |
| Fehlerkategorie | --- |
| Bemerkungen | Testet grundlegende CRUD-Operationen für Benutzer. |

Quelle: eigene Darstellung

Anhang M: Testprotokoll SYS_001

| **Feld** | **Inhalt** |
|---|---|
| Testfall-ID | SYS_001 |
| Priorität | Must-Have |
| Kurzbeschreibung | Überprüfung der Funktionalität zur Erstellung von Aufgaben über die Benutzeroberfläche (GUI). |
| Vorbedingungen | Die Anwendung ist gestartet und der Benutzer ist erfolgreich eingeloggt.<br>Keine bestehenden Aufgaben in der Aufgabenliste. |
| Eingaben/Aktionen | Öffnen Sie die Task-Editor-Oberfläche<br>Füllen Sie alle erforderlichen Felder aus:<br>• Titel: "Testaufgabe 1"<br>• Beschreibung: "Dies ist eine Testbeschreibung."<br>• Fälligkeitsdatum: Heute + 3 Tage (z. B. 28.11.2024).<br>• Wichtigkeit: Hoch; Dringlichkeit: Niedrig; Fitness: Hoch<br>Klicken Sie auf "Speichern".<br>Beobachten Sie, ob die neue Aufgabe in der Aufgabenliste angezeigt wird.<br><br>Wiederholen Sie die Schritte 1–4 mit fehlenden oder fehlerhaften Eingaben:<br>• Leerer Titel<br>• Ungültiges Datum (2024-13-32). |
| Erwartetes Ergebnis | Die Aufgabe wird erfolgreich erstellt und korrekt in der Aufgabenliste angezeigt.<br>Fehlermeldungen werden korrekt angezeigt, wenn ungültige Eingaben gemacht werden. |
| Tatsächliches Ergebnis | Entspricht erwartetem Ergebnis. |
| Status | Test bestanden. |
| Fehlerkategorie | --- |
| Bemerkungen | Dieser Test stellt sicher, dass die Kernfunktionalität der Aufgabenerstellung über die GUI einwandfrei funktioniert.<br>Die Validierungslogik für Eingabefelder wird ebenfalls überprüft. |

Quelle: eigene Darstellung

Anhang N: Testprotokoll SYS_001

| **Feld** | **Inhalt** |
|---|---|
| Testfall-ID | SYS_001 |
| Priorität | Must-Have |
| Kurzbeschreibung | Überprüfung der Funktionalität zur Erstellung von Aufgaben über die Benutzeroberfläche (GUI). |
| Vorbedingungen | Die Anwendung ist gestartet und der Benutzer ist erfolgreich eingeloggt.<br>Keine bestehenden Aufgaben in der Aufgabenliste. |
| Eingaben/Aktionen | Öffnen Sie die Task-Editor-Oberfläche<br>Füllen Sie alle erforderlichen Felder aus:<br>• Titel: "Testaufgabe 1"<br>• Beschreibung: "Dies ist eine Testbeschreibung."<br>• Fälligkeitsdatum: Heute + 3 Tage (z. B. 28.11.2024).<br>• Wichtigkeit: Hoch; Dringlichkeit: Niedrig; Fitness: Hoch<br>Klicken Sie auf "Speichern".<br>Beobachten Sie, ob die neue Aufgabe in der Aufgabenliste angezeigt wird.<br><br>Wiederholen Sie die Schritte 1–4 mit fehlenden oder fehlerhaften Eingaben:<br>• Leerer Titel<br>• Ungültiges Datum (2024-13-32). |
| Erwartetes Ergebnis | Die Aufgabe wird erfolgreich erstellt und korrekt in der Aufgabenliste angezeigt.<br>Fehlermeldungen werden korrekt angezeigt, wenn ungültige Eingaben gemacht werden. |
| Tatsächliches Ergebnis | Entspricht erwartetem Ergebnis. |
| Status | Test bestanden. |
| Fehlerkategorie | --- |
| Bemerkungen | Dieser Test stellt sicher, dass die Kernfunktionalität der Aufgabenerstellung über die GUI einwandfrei funktioniert.<br>Die Validierungslogik für Eingabefelder wird ebenfalls überprüft. |

Quelle: eigene Darstellung

Anhang O: Testprotokoll SYS_002

| **Feld** | **Inhalt** |
|---|---|
| Testfall-ID | SYS_002 |
| Priorität | Must-Have |
| Kurzbeschreibung | Überprüfung der Funktionalität zur Bearbeitung von Aufgaben über die Benutzeroberfläche (GUI). |
| Vorbedingungen | Die Anwendung ist gestartet und der Benutzer ist erfolgreich eingeloggt.<br>Mindestens eine bestehende Aufgabe ist in der Aufgabenliste vorhanden. |
| Eingaben/Aktionen | Öffnen Sie die Task-Editor-Oberfläche<br>Füllen Sie alle erforderlichen Felder aus:<br>• Titel: "Geänderter Testtitel"<br>• Beschreibung: "Geänderte Testbeschreibung."<br>• Fälligkeitsdatum: Heute + 5 Tage (z. B. 28.11.2024).<br>• Wichtigkeit: Niedrig; Dringlichkeit: Hoch; Fitness: Niedrig<br>Klicken Sie auf "Speichern".<br>Beobachten Sie, ob die neue Aufgabe in der Aufgabenliste angezeigt wird.<br><br>Wiederholen Sie die Schritte mit fehlenden oder fehlerhaften Eingaben:<br>• Leerer Titel<br>• Ungültiges Datum (2024-13-32). |
| Erwartetes Ergebnis | Die Aufgabe wird erfolgreich bearbeitet und die Änderungen werden in der Aufgabenliste korrekt angezeigt.<br>Fehlermeldungen werden korrekt angezeigt, wenn ungültige Eingaben gemacht werden. |
| Tatsächliches Ergebnis | Entspricht erwartetem Ergebnis. |
| Status | Test bestanden. |
| Fehlerkategorie | --- |
| Bemerkungen | Dieser Test stellt sicher, dass bestehende Aufgaben über die GUI korrekt bearbeitet werden können.<br>Die Validierungslogik für Eingabefelder wird ebenfalls überprüft. |

Quelle: eigene Darstellung

Anhang P: Testprotokoll SYS_003

| Feld | Inhalt |
|---|---|
| Testfall-ID | SYS_003 |
| Priorität | Must-Have |
| Kurzbeschreibung | Überprüfung der Funktionalität zum Löschen von Aufgaben über die Benutzeroberfläche (GUI) und Sicherstellung der Konsistenz in der Anzeige und Datenbank. |
| Vorbedingungen | Die Anwendung ist gestartet und der Benutzer ist erfolgreich eingeloggt.<br>Mindestens eine Aufgabe ist in der Aufgabenliste vorhanden. |
| Eingaben/Aktionen | Wählen Sie eine Aufgabe aus der Aufgabenliste aus.<br>Löschen Sie die Aufgabe. Bestätigen Sie die Löschaktion, falls eine Sicherheitsabfrage erscheint.<br>Prüfen Sie:<br>• Die Aufgabe verschwindet sofort aus der Aufgabenliste.<br>• Es gibt keine verbleibenden Verweise auf die gelöschte Aufgabe (z. B. in Benachrichtigungen, Archiven). |
| Erwartetes Ergebnis | Die gelöschte(n) Aufgabe(n) wird/werden nicht mehr in der Aufgabenliste angezeigt.<br>Die Konsistenz wird gewahrt:<br>• Die Aufgabe ist nicht mehr in der Datenbank gespeichert.<br>• Verknüpfte Daten (z. B. Benachrichtigungen) werden ebenfalls entfernt.<br>• Es treten keine Fehler oder unerwarteten Verhalten in der Benutzeroberfläche auf. |
| Tatsächliches Ergebnis | Entspricht erwartetem Ergebnis. |
| Status | Test bestanden. |
| Fehlerkategorie | --- |
| Bemerkungen | Dieser Test stellt sicher, dass die Löschfunktion vollständig und konsistent arbeitet.<br>Prüfen Sie besonders auf Restdaten in der Datenbank, die nicht ordnungsgemäß gelöscht wurden. |

Quelle: eigene Darstellung

Anhang Q: Testprotokoll SYS_004

| Feld | Inhalt |
|---|---|
| Testfall-ID | SYS_004 |
| Priorität | Must-Have |
| Kurzbeschreibung | Überprüfung der Drag-and-Drop-Funktionalität zum Verschieben von Aufgaben zwischen Kategorien. |
| Vorbedingungen | Die Anwendung ist gestartet und der Benutzer ist erfolgreich eingeloggt.<br>Es ist mindestens eine Aufgabe vorhanden. |
| Eingaben/Aktionen | Ziehen Sie eine Aufgabe aus einer Kategorie in eine andere und lassen Sie sie los:<br>• Zwischen gültigen Kategorien.<br>• Zurück in die ursprüngliche Kategorie. |
| Erwartetes Ergebnis | Aufgaben werden korrekt in die Zielkategorie verschoben:<br>• Die Benutzeroberfläche wird aktualisiert, und die Aufgabe wird nur in der Zielkategorie angezeigt.<br>• Datenbankeinträge für die Aufgabe werden entsprechend angepasst. |
| Tatsächliches Ergebnis | Entspricht erwartetem Ergebnis. |
| Status | Test bestanden. |
| Fehlerkategorie | --- |
| Bemerkungen | Dieser Test sollte auch das Verhalten der Benutzeroberfläche bei Drag-and-Drop-Aktionen prüfen. |

Quelle: eigene Darstellung

Anhang R: Testprotokoll SYS_005

| **Feld** | **Inhalt** |
|---|---|
| Testfall-ID | SYS_005 |
| Priorität | Must-Have |
| Kurzbeschreibung | Test der Funktionalität zum Archivieren und Reaktivieren von Aufgaben über die Benutzeroberfläche. |
| Vorbedingungen | Die Anwendung ist gestartet und der Benutzer ist erfolgreich eingeloggt.<br>Es ist mindestens eine Aufgabe vorhanden. |
| Eingaben/Aktionen | Archivieren<br>• Wählen Sie eine Aufgabe aus.<br>• Klicken Sie auf die Schaltfläche(n) um die Aufgabe zu archivieren.<br>Überprüfung nach der Archivierung:<br>• Öffnen Sie die Archivansicht und überprüfen Sie, ob die archivierten Aufgaben korrekt angezeigt werden.<br>Reaktivieren:<br>• Wählen Sie eine archivierte Aufgabe in der Archivansicht aus.<br>• Reaktivieren Sie die Aufgabe.<br>Überprüfung nach der Reaktivierung:<br>• Überprüfen Sie, ob die reaktivierte Aufgabe korrekt in der Hauptaufgabenliste erscheint und aus der Archivansicht entfernt wird. |
| Erwartetes Ergebnis | Archivieren:<br>• Aufgaben werden erfolgreich in die Archivansicht verschoben.<br>• Archivierte Aufgaben sind nicht mehr in der Hauptaufgabenliste<br>Reaktivieren:<br>• Archivierte Aufgaben werden erfolgreich zurück in die Hauptaufgabenliste verschoben.<br>Bei Fehlern:<br>• Die Benutzeroberfläche zeigt eine verständliche Fehlermeldung an. |
| Tatsächliches Ergebnis | Entspricht erwartetem Ergebnis. |
| Status | Test bestanden. |
| Fehlerkategorie | --- |

| | |
|---|---|
| Bemerkungen | Der Test sollte auch Fehlerfälle prüfen, wie das Archivieren von Aufgaben ohne den Status "Completed" oder das Reaktivieren von Aufgaben ohne Änderungen vorzunehmen. |

Quelle: eigene Darstellung

Anhang S: Testprotokoll SYS_006

| **Feld** | **Inhalt** |
|---|---|
| Testfall-ID | SYS_006 |
| Priorität | Should-Have |
| Kurzbeschreibung | Überprüfung der korrekten Anwendung von Filtern auf der Benutzeroberfläche und Konsistenz der angezeigten Aufgaben. |
| Vorbedingungen | Die Anwendung ist gestartet und der Benutzer ist erfolgreich eingeloggt.<br>Es sind mehrere Aufgaben mit unterschiedlichen Prioritäten, Status und Fälligkeitsdaten vorhanden. |
| Eingaben/Aktionen | Filter nach Priorität:<br>• Wählen Sie in den Filtereinstellungen eine spezifische Priorität (z. B. "High Importance") aus.<br>• Überprüfen Sie die angezeigten Aufgaben, ob nur die Aufgaben mit der gewählten Priorität angezeigt werden.<br>Filter nach Namen:<br>• Wählen Sie einen spezifischen Task-Namen aus.<br>• Überprüfen Sie die angezeigten Aufgaben, ob nur die Aufgaben mit dem gewählten Namen angezeigt werden.<br>Filter nach Fälligkeitsdatum:<br>• Wählen Sie ein Fälligkeitsdatum aus.<br>• Überprüfen Sie, ob die angezeigten Aufgaben in den gewählten Zeitraum fallen.<br>Kombination von Filtern:<br>• Wenden Sie mehrere Filter gleichzeitig an.<br>• Überprüfen Sie, ob die Anzeige korrekt auf die Kombination der Filter reagiert.<br>Filter zurücksetzen:<br>• Setzen Sie alle Filter zurück und überprüfen Sie, ob wieder alle Aufgaben angezeigt werden. |
| Erwartetes Ergebnis | Filter nach Priorität:<br>• Nur Aufgaben mit der gewählten Priorität werden angezeigt.<br>Filter nach Status:<br>• Nur Aufgaben mit dem gewählten Status werden angezeigt.<br>Filter nach Fälligkeitsdatum:<br>• Nur Aufgaben mit einem Fälligkeitsdatum im gewählten Zeitraum werden angezeigt. |

| | |
|---|---|
| | Kombination von Filtern:<br>• Die angezeigten Aufgaben erfüllen alle angewendeten Filterkriterien.<br>Filter zurücksetzen:<br>• Alle Aufgaben werden korrekt ohne Einschränkungen angezeigt. |
| Tatsächliches Ergebnis | Entspricht erwartetem Ergebnis. |
| Status | Test bestanden. |
| Fehlerkategorie | --- |
| Bemerkungen | Überprüfen Sie, ob die Benutzeroberfläche auf Filteränderungen sofort reagiert, ohne dass ein Neustart oder eine Aktualisierung erforderlich ist.<br>Testen Sie auch mögliche Randfälle, wie z. B. das Filtern, wenn keine Aufgaben vorhanden sind oder Filterkriterien ausgewählt werden, die keine Ergebnisse liefern. |

Quelle: eigene Darstellung

Anhang T: Testprotokoll SYS_007

| **Feld** | **Inhalt** |
|---|---|
| Testfall-ID | SYS_007 |
| Priorität | Should-Have |
| Kurzbeschreibung | Überprüfung der Benachrichtigungen in der GUI basierend auf den Fälligkeiten der Aufgaben und den Benutzereinstellungen. |
| Vorbedingungen | Die Anwendung ist gestartet und der Benutzer ist erfolgreich eingeloggt.<br>Es sind Aufgaben mit unterschiedlichen Fälligkeitsdaten und Prioritäten vorhanden.<br>Benachrichtigungen sind in den Einstellungen aktiviert, und der Benachrichtigungsintervall ist konfiguriert. |
| Eingaben/Aktionen | Erstellen Sie eine Aufgabe mit einer Fälligkeit innerhalb des Benachrichtigungsintervalls.<br>Erstellen Sie eine Aufgabe mit einer Fälligkeit außerhalb des Benachrichtigungsintervalls.<br>Überprüfen Sie die Benachrichtigung in der GUI.<br>Deaktivieren Sie die Benachrichtigungen in den Einstellungen.<br>• Überprüfen Sie, ob keine Benachrichtigungen angezeigt werden.<br>Ändern Sie den Benachrichtigungsintervall und überprüfen Sie, ob die Benachrichtigungen entsprechend aktualisiert werden.<br>Testen Sie die Anzeige, wenn keine Aufgaben fällig sind. |
| Erwartetes Ergebnis | Aufgaben mit einer Fälligkeit innerhalb des Intervalls werden korrekt mittels Benachrichtigung angezeigt.<br>Aufgaben mit einer Fälligkeit außerhalb des Intervalls erscheinen nicht in der Benachrichtigung.<br>Nach Deaktivierung der Benachrichtigungen werden keine Benachrichtigungen angezeigt.<br>Änderungen des Benachrichtigungsintervalls aktualisieren die Benachrichtigungen korrekt.<br>Bei keiner fälligen Aufgabe erscheint keine Benachrichtigung. |
| Tatsächliches Ergebnis | Benachrichtigungen werden nicht angezeigt. |
| Status | Test NICHT bestanden. |
| Fehlerkategorie | Mittel |

| | |
|---|---|
| Bemerkungen | Überprüfen Sie die Aktualisierung der Benachrichtigungsliste in Echtzeit, falls die Fälligkeit einer Aufgabe geändert wird.<br>Validieren Sie, ob die Benachrichtigungen korrekt bei Änderungen der Benutzereinstellungen ein- und ausgeschaltet werden. |

Quelle: eigene Darstellung

Anhang U: Testprotokoll SYS_008

| **Feld** | **Inhalt** |
|---|---|
| Testfall-ID | SYS_008 |
| Priorität | Should-Have |
| Kurzbeschreibung | Überprüfung, ob die Anwendung nach dem Start den letzten gespeicherten Zustand korrekt lädt. |
| Vorbedingungen | Es wurden zuvor Aufgaben erstellt, geändert und archiviert.<br>Benutzereinstellungen wurden angepasst.<br>Die Anwendung wurde ordnungsgemäß beendet. |
| Eingaben/Aktionen | Starten Sie die Anwendung.<br>Überprüfen Sie, ob:<br>- Aufgaben korrekt geladen werden.<br>- Benutzerpräferenzen korrekt übernommen wurden. |
| Erwartetes Ergebnis | Die Anwendung zeigt die zuletzt gespeicherte Aufgabenliste an.<br>Archivierte Aufgaben befinden sich weiterhin im Archiv.<br>Benutzereinstellungen, sind korrekt übernommen.<br>Keine Fehlermeldungen werden angezeigt. |
| Tatsächliches Ergebnis | Entspricht erwartetem Ergebnis. |
| Status | Test bestanden. |
| Fehlerkategorie | --- |
| Bemerkungen | Dieser Test prüft die Benutzerfreundlichkeit und Datenkonsistenz nach einem Neustart der Anwendung. |

Quelle: eigene Darstellung

Anhang V: Testprotokoll SYS_009

| **Feld** | **Inhalt** |
|---|---|
| Testfall-ID | SYS_009 |
| Priorität | Should-Have |
| Kurzbeschreibung | Überprüfung, ob der Benutzerwechsel reibungslos funktioniert (nach einem Programm-Neustart). |
| Vorbedingungen | Zwei oder mehr Benutzerkonten sind in der Anwendung erstellt.<br>Aufgaben und Einstellungen für jeden Benutzer wurden angelegt.<br>Die Anwendung wurde ordnungsgemäß beendet. |
| Eingaben/Aktionen | Starten Sie die Anwendung.<br>Melden Sie sich mit einem Benutzerkonto an und überprüfen Sie, ob die entsprechenden Daten korrekt geladen werden.<br>Beenden Sie die Anwendung und starten Sie sie erneut.<br>Melden Sie sich mit einem anderen Benutzerkonto an und überprüfen Sie erneut die geladenen Daten und Einstellungen. |
| Erwartetes Ergebnis | Die Anwendung lädt die korrekten Aufgaben und Einstellungen für das angemeldete Benutzerkonto.<br>Archivierte Aufgaben bleiben benutzerspezifisch korrekt zugeordnet.<br>Es treten keine Fehler oder unerwartete Datenmischungen auf. |
| Tatsächliches Ergebnis | Entspricht erwartetem Ergebnis. |
| Status | Test bestanden. |
| Fehlerkategorie | --- |
| Bemerkungen | Test ist besonders relevant für Szenarien mit mehreren Benutzern und geteilter Nutzung der Anwendung.. |

Quelle: eigene Darstellung

Anhang W: Testprotokoll SYS_010

| Feld | Inhalt |
|---|---|
| Testfall-ID | SYS_010 |
| Priorität | Should-Have |
| Kurzbeschreibung | Test der Stabilität und Performance der Anwendung bei langfristiger Nutzung. |
| Vorbedingungen | Anwendung wird kontinuierlich für mindestens 6 Stunden verwendet.<br>Tasks, Filter und andere Funktionen werden regelmäßig genutzt. |
| Eingaben/Aktionen | Starten Sie die Anwendung.<br>Erstellen, bearbeiten und löschen Sie regelmäßig Aufgaben.<br>Nutzen Sie Drag-and-Drop-Funktionen und Filteroptionen.<br>Lassen Sie die Anwendung für längere Zeit laufen, während Sie zwischendurch weitere Aktionen ausführen.<br>Beobachten Sie die Performance und Stabilität der Anwendung während dieser Zeit. |
| Erwartetes Ergebnis | Die Anwendung bleibt während der gesamten Laufzeit stabil und performant.<br>Es treten keine Speicherlecks, unerwarteten Abstürze oder erhebliche Verzögerungen auf.<br>Alle Benachrichtigungen werden korrekt angezeigt, und die GUI bleibt reaktionsfähig. |
| Tatsächliches Ergebnis | |
| Status | Ausstehend. |
| Fehlerkategorie | |
| Bemerkungen | Langzeit-Stabilität ist besonders wichtig für Szenarien, in denen die Anwendung ganztägig geöffnet bleibt.<br>Test sollte bei realistischen Nutzungsszenarien durchgeführt werden, einschließlich gleichzeitiger Belastung durch mehrere Funktionen. |

Quelle: eigene Darstellung

Anhang X: Testprotokoll SYS_011

| Feld | Inhalt |
|---|---|
| Testfall-ID | SYS_011 |
| Priorität | Should-Have |
| Kurzbeschreibung | Messung der Ladezeiten und Performance bei wichtigen Aktionen wie Task-Erstellung, -Bearbeitung und -Archivierung. |
| Vorbedingungen | Die Anwendung ist gestartet und eine Standard-Datenbank ist geladen.<br>Es existieren mindestens 10 Tasks in der Anwendung. |
| Eingaben/Aktionen | Messen Sie die Ladezeit beim Start der Anwendung.<br>Erstellen Sie eine neue Aufgabe und messen Sie die Zeit bis zur Bestätigung.<br>Bearbeiten Sie eine vorhandene Aufgabe und messen Sie die Reaktionszeit der GUI.<br>Archivieren Sie eine Aufgabe und messen Sie die Zeit bis zur Rückmeldung.<br>Nutzen Sie Drag-and-Drop-Funktionen und prüfen Sie die Verzögerung beim Aktualisieren der GUI. |
| Erwartetes Ergebnis | Ladezeiten für einzelne Aktionen überschreiten nicht die definierten Schwellenwerte (z. B. max. 1 Sekunde für GUI-Aktionen, max. 2 Sekunden für Start).<br>Keine signifikanten Verzögerungen oder Performance-Einbußen bei mehreren aufeinanderfolgenden Aktionen. |
| Tatsächliches Ergebnis | Entsprich erwartetem Ergebnis |
| Status | Test bestanden. |
| Fehlerkategorie | --- |
| Bemerkungen | Messungen sollten mit realistischen Datenmengen und Lasten durchgeführt werden.<br>Testergebnisse könnten als Basis für weitere Optimierungen dienen, insbesondere bei identifizierten Engpässen. |

Quelle: eigene Darstellung

Portfolio – Phase 3

IU Internationale Hochschule Fernstudium

Studiengang: Master of Science Informatik

Kurs: Projekt – Software Engineering (DLMCSPSE01_D)

Abstract

Windows Anwendung:
Priorisierungsmatrix nach dem Sung-Diagramm

Gregor Hubmann

Matrikelnummer: 9196398

Hornstraße 185

5423 St. Koloman

Abgabedatum (Phase 3): 08.12.2024

Änderungen nach Phase 2:

In Phase 3 neu erstellt.

Inhaltsverzeichnis

## 1. Zusammenfassung der Projektdurchführung

Das Projekt zielte darauf ab, eine benutzerfreundliche Priorisierungsmatrix-Anwendung basierend auf dem Sung-Diagramm zu entwickeln. Es folgte einer klar strukturierten Methodik, die in drei Hauptphasen unterteilt war. In der Konzeptionsphase wurde die Grundlage des Projekts gelegt: eine detaillierte Projektdokumentation wurde begonnen, und sowohl das Anforderungs- als auch das Spezifikationsdokument erstellt. Hierbei stand die genaue Definition der funktionalen und nicht-funktionalen Anforderungen im Vordergrund. Diese Phase bildete das Fundament für alle weiteren Aktivitäten und wurde fristgerecht abgeschlossen (Ende Phase 1: 27.10.2024).

In der Umsetzungsphase lag der Fokus auf der technischen Realisierung. Das Architekturdokument wurde erstellt, um die Struktur der Anwendung zu definieren. Hierbei zeigte sich, dass die Balance zwischen Detailliertheit und Übersichtlichkeit nicht immer leicht zu finden war. Trotz dieser Herausforderungen gelang es, die Kernfunktionalitäten, wie Aufgabenverwaltung, Drag-and-Drop und Filtermöglichkeiten, termingerecht umzusetzen. Einige geplante Komfortfunktionen, wie die Bearbeitung von Aufgaben per Doppelklick oder eine intuitive Datumsauswahl über einen Calendar-Picker, wurden aufgrund von Zeitmangel jedoch zurückgestellt (Ende Phase 2: 24.11.2024).

Die abschließende Test- und Optimierungsphase diente der Qualitätssicherung und Verfeinerung der Anwendung. Ein umfangreiches Testdokument wurde erstellt, das sowohl automatisierte Unit-Tests als auch manuelle Systemtests umfasste. Parallel dazu wurden bestehende Funktionen optimiert und offene Punkte bearbeitet. Abschließend wurde das Projekt in einer kritischen Reflexion betrachtet, um „Lessons Learned“ für zukünftige Vorhaben festzuhalten. Die ursprünglich anvisierten Deadlines konnten in allen drei Phasen eingehalten werden, was den Erfolg des Projektmanagements unterstreicht. Hierbei gebührt auch dem ausführlichen und zeitnahen Feedback Dank, das maßgeblich zur Qualität und Zielerreichung beigetragen hat (Ende Phase 3: 08.12.2024).

## 2. Was funktionierte gut?

Der strukturierte Aufbau des Projekts in drei Phasen mit jeweils klar definierten Outputs erwies sich als besonders hilfreich. Dieser Ansatz sorgte für klare Orientierungspunkte und half dabei, den Fortschritt kontinuierlich zu überprüfen und zu bewerten. Ergänzt durch regelmäßige Tests und zeitnahes Feedback konnte eine angemessene Qualität der Arbeit sichergestellt werden.

## 3. Herausforderungen und unerwartete Probleme

Durch die begrenzte praktische Programmiererfahrung und das erstmalige eigenständige Durchführen eines solchen Projekts waren einige Aspekte besonders herausfordernd. Die GUI-Entwicklung stellte sich, insbesondere bei der Implementierung der Drag-and-Drop-Funktion, als komplexer heraus als zunächst angenommen. Hier war es schwierig, sowohl die technische Umsetzung als auch die Nutzerfreundlichkeit in Einklang zu bringen.

Auch das Zeitmanagement erwies sich als problematisch. Dies lag einerseits an der Unerfahrenheit, realistische Zeitbudgets für verschiedene Phasen einzuplanen, andererseits an parallellaufenden Verpflichtungen, die die verfügbare Arbeitszeit stark einschränkten.

Zudem nahmen auftauchende Fehler durch Änderungen oft unerwartet viel Zeit in Anspruch. Diese Debugging-Phasen zeigten deutlich, wie wichtig ein strukturiertes Testen und eine präzise Planung bei Änderungen sind, um Folgeprobleme zu minimieren.

## 4. Was könnte man besser machen?

Ein zentraler Punkt, der verbessert werden könnte, betrifft die verfügbare Zeit für die Projektphasen. Besonders Phase 2, in der die Architektur definiert und die Implementierung begann, und Phase 3, die den finalen Feinschliff und Tests umfasste, hätten mehr Zeit benötigt, um alle Anforderungen in der gewünschten Qualität umzusetzen. Insbesondere Phase 3, mit ihrer Vielzahl an Aufgaben wie Testing, Fehlerbehebung und Dokumentation, war zeitlich zu knapp bemessen.

Die technische Unerfahrenheit war ein weiterer Faktor, der den Ablauf beeinflusst hat. Dies zeigte sich nicht nur bei der GUI-Entwicklung, insbesondere der Drag-and-Drop-Funktion, sondern auch bei der Umsetzung der Schichtenarchitektur. So wurde in manchen Bereichen die Trennung nicht konsequent eingehalten, und Teile der Logik fanden ihren Weg in den GUIController, was ursprünglich vermieden werden sollte. Mit mehr Erfahrung im Vorfeld hätte hier konsequenter strukturiert werden können.

Auch die Dokumentation, speziell das Architekturdokument, hätte besser ausbalanciert werden können. Es war in Teilen (wie den Hauptkomponenten) zu detailliert und in anderen (z. B. beim UML-Klassendiagramm) zu oberflächlich gehalten. Dies zeigt, dass ein besseres Verständnis der erforderlichen Detailebenen und der Zweckmäßigkeit solcher Dokumente eine wichtige Verbesserung darstellt.

## 5. Wichtige Erkenntnisse

Das Projekt hat verdeutlicht, wie entscheidend eine realistische Zeitplanung ist. Die strukturierte Herangehensweise, unterteilt in klar definierte Phasen mit spezifischen Outputs, erwies sich als von großem Vorteil, da sie Orientierung und Fokus schaffte. Dennoch machten unterschätzte Aufgaben und unerwartete Probleme, wie komplexe Fehlerbehebungen, deutliche Pufferzeiten notwendig.

Auch die Bedeutung einer sauberen Schichtenarchitektur wurde klar, da fehlende Trennung von Logik und Präsentation zu erhöhtem Anpassungsaufwand führte. Zudem zeigte sich, dass eine iterative Herangehensweise oft effektiver ist als Perfektionismus von Beginn an. Der pragmatische Umgang mit Problemen war eine wertvolle Erfahrung, die in zukünftigen Projekten weiter vertieft werden kann.