
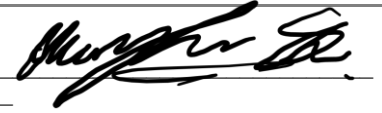



Softwareentwicklungsplan

Erstellt von:	Geprüft von:	Freigegeben von:
Magamet Sulejmanov	*PL Magamet Sulejmanov	*PL Magamet Sulejmanov
Datum	Datum	Datum
29.05.2025	03.06.2025	03.06.2025
Unterschrift	Unterschrift	Unterschrift
		

Gültig ab: „Freigabedatum“

Gültig bis: Auf Widerruf

*PL = Projektleiter

1 Änderungsverzeichnis

Version	Name	Datum	Bemerkung (Betrifft Punkt; Grund der Änderung)
1.0	Ing. Magamet Sulejmanov	29.05.2025	Neuerstellung
1.1	Ing. Magamet Sulejmanov	03.06.2025	User Stories hinzugefügt, Anpassungen im Softwaredesign, Überarbeitung der Implementierungsdetails, Hinzufügen der Schnittstellen zu anderen Prozessen, Aktualisierung des Risikomanagementprozesses
1.2	Ing. Magamet Sulejmanov	03.06.2025	Implementierung und Verifizierung
1.3	Ing. Magamet Sulejmanov	03.06.2025	Software – Systemtests
1.4	Ing. Musa Arslan	03.06.2025	CI – CD Integration

Tabelle 1: Änderungsverzeichnis

2 Zielsetzung

Ziel dieser Arbeitsanweisung ist die Beschreibung des Prozesses Softwareentwicklung unter Berücksichtigung der Norm EN 62304.

3 Geltungsbereich

Dieses Dokument gilt für die Softwareentwicklungsabteilung der Firma „WRAITH – OSINT“.

4 Änderungsgrund

Neuerstellung

User Stories hinzugefügt, Anpassungen im Softwaredesign, Überarbeitung der Implementierungsdetails, Hinzufügen der Schnittstellen zu anderen Prozessen, Aktualisierung des Risikomanagementprozesses, Implementierung und Verifizierung und Software – Systemtests / CI – CD Integration

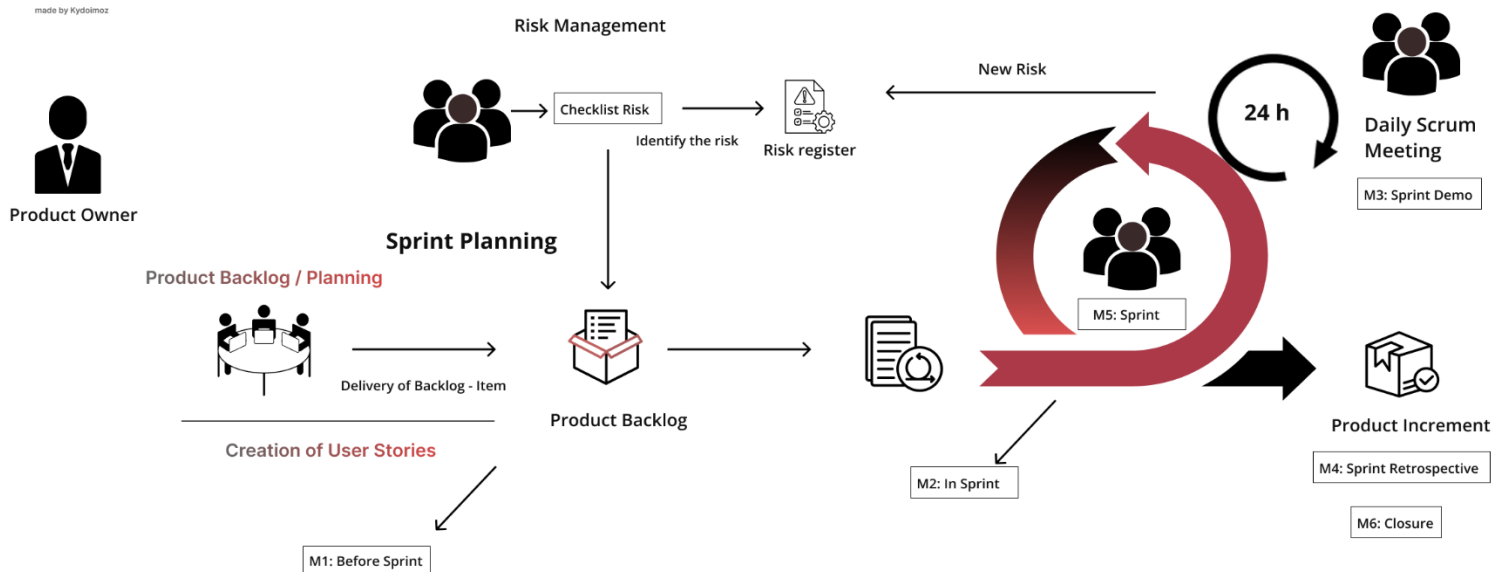
5 Prozesse

Die folgende Grafik zeigt eine Übersicht über den Softwareentwicklungsprozess unter Verwendung eines agilen SCRUM-Frameworks in Kombination mit einem Risikomanagementprozess (WRAITH - OSINT).

Die zentralen Entwicklungsaktivitäten wie Sprint Planning, Sprint selbst und die Erstellung von Produktinkrementen sind in der Grafik klar erkennbar. Unterstützende Prozesse wie das Risikomanagement greifen direkt in die Planung und Umsetzung ein, indem identifizierte Risiken laufend in das Sprint-Geschehen zurückgeführt werden.

WRAITH - OSINT - Risk Framework & SCRUM

made by Kydormaz



Der gesamte Ablauf ist iterativ aufgebaut und basiert auf regelmäßigen Scrum-Zeremonien wie dem Daily Scrum, Sprint Demo, Retrospektive und der kontinuierlichen Pflege des Product Backlogs.

Jede Phase der Planung wird somit durch einen Test verifiziert.

In den folgenden Kapiteln werden die einzelnen Prozesse beschrieben.

Rückverfolgbarkeit

Die Rückverfolgbarkeit zwischen Systemanforderungen, Softwareanforderungen, Softwaresystemprüfung und Risikokontrollmaßnahmen ist über die Kombination aus Anforderungsdokument, Versionsverwaltungssystem und Testautomatisierungstool als Integrationstestsystem gewährleistet. Die Anforderungen müssen eindeutig identifizierbar mit einer ID versehen sein. Die Commits in GIT müssen auf diese IDs verweisen und die Tests werden für jeden Commit durchgeführt.

Beschreibung Prozess Softwareentwicklung:

Prozesseigner:

- Leiter der Entwicklungsabteilung

Eingangsprozess

- Risikomanagementprozess
- Problemlösungsprozess
- Software-Konfigurationsmanagementprozess

Ausgangsprozess

- Wartung und Betrieb

Entstehende Artefakte

Im Rahmen der Entwicklung entstehen strukturierte Commits im GitHub-Repository. Diese durchlaufen automatisierte Tests (CI) und werden nach definierten Kriterien als Release Candidates oder stabile Releases gekennzeichnet. Die GitHub-Plattform wird gleichzeitig als Repository, CI/CD-Tool und Projektverwaltungsumgebung verwendet.

Technologie-Stack

Die Anwendung ist als Ruby-basierte Konsolenanwendung konzipiert. Zur Versionsverwaltung und kontinuierlichen Integration/Deployment wird GitHub Actions eingesetzt. CI/CD-Pipelines prüfen automatisiert Codequalität, Tests und Deploymentfähigkeit.

Prozessbeschreibung

Bei der Softwareentwicklung gibt es folgende Aktivitäten und Tasks:

5.1 Aktivität: Softwareentwicklungsplanung

5.1.1 Task: Softwareentwicklungsplan

Entstehende Artefakte (Deliverables): Softwareentwicklungsplan

Beschreibung: Es wird dieses Dokument als Softwareentwicklungsplan nach ausgewählten Kapiteln der Norm 62304 erstellt.

5.1.2 Task: Planung Softwareintegration und Integrationstest

Entstehende Artefakte (Deliverables):

5.2 Aktivität: Softwareanforderungsanalyse

5.2.1 Task: Inhalt der Softwareanforderungen

Im Rahmen dieses Projekts soll ein modular aufgebautes Recon-Toolset zur automatisierten Informationsgewinnung (OSINT) entwickelt werden, das spezifische Recon-Aufgaben übernehmen und dabei mögliche Sicherheitsrisiken aufdecken kann.

Das Product Increment besteht aus mehreren Einzelmodulen, die jeweils eine klar definierte Funktionalität abdecken. Dabei liegt der Fokus zunächst auf drei zentralen Pflichtmodulen, die den Kern des Tools darstellen und vollständig funktionsfähig ausgeliefert werden müssen.

Pflichtmodule (Kernbestandteile):

1. **google-dorking.rb**
Dieses Modul ermöglicht die gezielte Suche nach sicherheitsrelevanten Informationen über Google. Es nutzt vordefinierte „Google Dorks“, um öffentlich zugängliche, aber sicherheitskritische Inhalte (wie Fehlkonfigurationen oder sensible Dateien) aufzuspüren.
2. **reverse-ip-lookup.rb**
Dieses Modul dient dazu, über eine gegebene IP-Adresse alle darauf gehosteten Domains und Subdomains zu ermitteln.
3. **port-scanner.rb**
Dieses Skript scannt eine Ziel-IP-Adresse auf offene Ports. Es liefert Informationen darüber, welche Dienste erreichbar sind, was erste Rückschlüsse auf potenzielle Angriffspunkte zulässt.

Entstehende Artefakte (Deliverables):

- User Stories-Dokument
- Use Case Diagramm
- Anforderungsdokument

Ablauf:

Bedarfsanalyse:

Zu Beginn des Projekts wird eine strukturierte Bedarfsanalyse durchgeführt, um die funktionalen und nicht-funktionalen Anforderungen an das OSINT-Tool WRAITH zu definieren. Hierbei werden die Perspektiven verschiedener Stakeholder berücksichtigt, darunter IT-Sicherheitsexperten, Entwickler, Analysten sowie potenzielle Endnutzer aus dem Bereich der digitalen Forensik und Netzwerkanalyse.

In einem initialen Workshop werden die konkreten Ziele, Einsatzbereiche und Erwartungen an das Tool gemeinsam erarbeitet. Dabei werden auch typische Anwendungsfälle (Use Cases) identifiziert, um die Priorisierung der Funktionen zu unterstützen. Besonderes Augenmerk liegt auf der Modularität, Benutzerfreundlichkeit sowie der Möglichkeit zur Erweiterung des Tools um weitere Analysefunktionen.

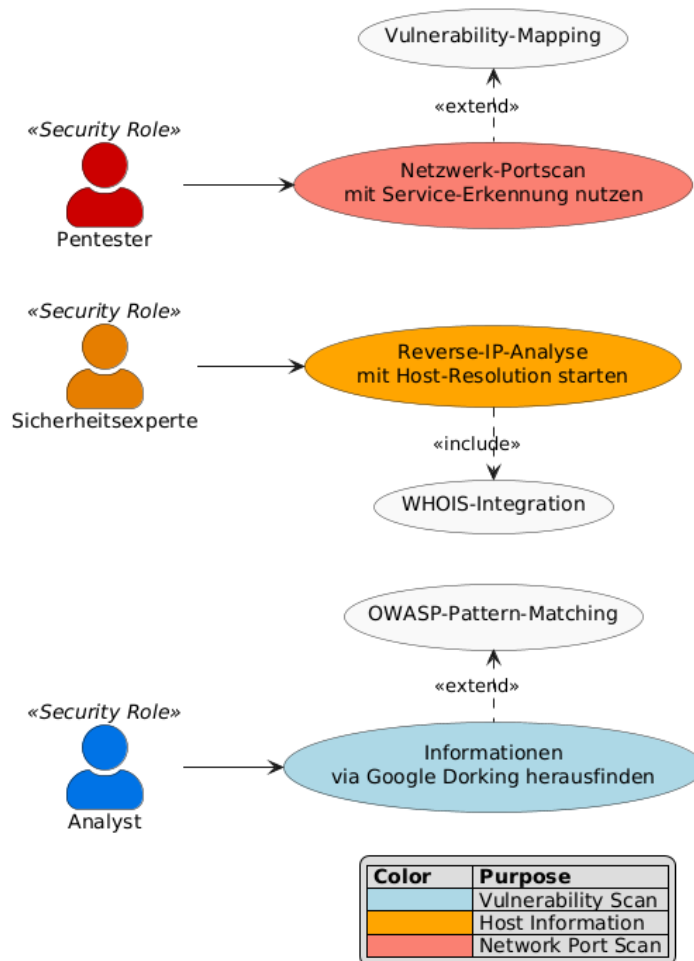
Formulierung der User Stories:

Die gesammelten Informationen werden in User Stories umgewandelt. Folgende User – Stories sind relevant für das Projekt:

- „Als Analyst möchte ich per **google-dorking.rb** gezielt Schwachstellen und sensible Daten in Webanwendungen finden.“
- „Als Sicherheitsexperte möchte ich mit **reverse-ip-lookup.rb** Hostinformationen effizient auflösen.“
- „Als Pentester möchte ich mit **port-scanner.rb** Netzwerke nach offenen Ports untersuchen.“

Ableitung der Anforderungen:

Aus den User Stories werden die spezifischen Anforderungen für das Softwaresystem abgeleitet. Dies geschieht durch die Erstellung eines Use Case Diagramms, das die Interaktionen zwischen den Benutzern und dem System darstellt.



Verantwortlichkeiten:

- **Business Analyst:** Leitung der Anforderungsanalyse und Erstellung der User Stories.
- **Entwicklungsteam:** Technische Beratung zur Umsetzbarkeit der Anforderungen.

- **Stakeholder:** Feedback und Validierung der Anforderungen.

Dokumentation:

- Alle Artefakte werden in einem zentralen Repository (z.B. Confluence oder ein Versionierungssystem wie GitHub) gespeichert, auf das alle Teammitglieder Zugriff haben.
- **User Stories** enthält alle User Stories, einschließlich der Priorität und der Akzeptanzkriterien.
- **Anforderungsdokument** im Softwareentwicklungsplan bietet eine detaillierte Beschreibung der funktionalen Anforderungen.
- **Use Case Diagramm** visualisiert die Benutzerinteraktionen mit dem System.

5.5 Aktivität: Implementierung und Verifizierung

5.5.1 Task: Implementieren der Softwareeinheiten

Entstehende Artefakte (Deliverables):

Die Implementierung der Software „WRAITH – OSINT (Open Source Intelligence Tool)“ erfolgt nach dem festgelegten Softwaredesign. Die Anforderungen werden einzeln von jedem/r Entwickler/in umgesetzt.

- **Auswahl der Anforderungen:** Die Auswahl der Anforderungen erfolgt in enger Absprache mit dem Entwicklungsleiter. Jede Entwicklerin ist dafür verantwortlich, die ihr zugewiesene Anforderung gemäß den festgelegten Standards und dem vorgegebenen Design umzusetzen.
- **Entwicklungsumgebung:** Es muss die in den Tools genannte Entwicklungsumgebung in der vorgegebenen Version verwendet werden. Eine IDE (Integrated Development Environment), wie Visual Studio Code, wird verwendet, da sie eine einheitliche Umgebung für das Schreiben, Testen und Debuggen des Codes bietet.
- **Vorgehensweise:** Die Implementierung folgt den folgenden Schritten:
 1. **Analyse der User Stories:** Jede Entwicklerin beginnt mit der detaillierten Analyse der ihr zugewiesenen User Stories, um die Anforderungen genau zu verstehen.
 2. **Umsetzung der Anforderungen:** Die Anforderungen werden in Code umgesetzt, wobei die Prinzipien der Clean Code-Entwicklung und Best Practices beachtet werden.
 3. **Systemtests:** Jede Entwicklerin führt Systemtests durch, um die Funktionalität der implementierten Komponenten sicherzustellen.
 4. **Regelmäßige Rücksprache:** Es werden wöchentliche Meetings mit dem Entwicklungsleiter abgehalten, um den Fortschritt zu überprüfen, eventuelle Herausforderungen zu besprechen und sicherzustellen, dass die Implementierung auf dem richtigen Weg ist.

5.5.2 Task: Akzeptanzkriterien für Softwareeinheiten

Entstehende Artefakte (Deliverables):

- **Anforderungsverifizierung:** Die Anforderungen werden durch Systemtests verifiziert. Zu jeder Anforderung wird ein entsprechender Testfall erstellt, um sicherzustellen, dass alle Spezifikationen erfüllt sind. Die angestrebte Testabdeckung (Coverage) liegt bei mindestens 80%, um sicherzustellen, dass die kritischen Funktionalitäten ausreichend getestet sind.
- **Mapping von AnforderungsID und TestID:** Ein Mapping-Dokument, das Anforderungs-IDs den jeweiligen Test-IDs zuordnet, wird gepflegt. Dieses Vorgehen stellt sicher, dass jede Anforderung mit einem spezifischen Testfall überprüft wird und gewährleistet die Rückverfolgbarkeit von Anforderungen zu Tests.
- **Code-Analyse durch Linter:** Ein Linter wird in den CI/CD-Prozess integriert, um den Code auf Syntaxfehler und Stilverstöße zu analysieren. Der Linter hilft, die Einhaltung von Codierungsstandards sicherzustellen und verbessert die Wartbarkeit sowie die Lesbarkeit des Codes. Jeder Merge-Request wird automatisch durch den Linter geprüft, bevor er in den Hauptzweig integriert wird.

Langfristige Perspektive für das Diplomprojekt

Um eine solide Grundlage für die Weiterentwicklung im kommenden Jahr zu schaffen, wird besonders auf eine nachhaltige und erweiterbare Codebasis geachtet. Es werden umfassende Dokumentationen zu Tests und Codequalität erstellt, um zukünftige Weiterentwicklungen zu erleichtern. Darüber hinaus werden Mechanismen wie CI/CD und ein strukturierter Review-Prozess etabliert, um die langfristige Stabilität des Projekts sicherzustellen und die Anforderungen für das Diplomprojekt zu unterstützen.

5.6 Softwareintegration und Integrationstests

5.6.1 Integration der Softwareeinheiten

Entstehende Artefakte (Deliverables):

Die Softwareintegration für WRAITH - OSINT erfolgt mit der Codeverwaltungssoftware **GIT** (*UI: GitHub*), um eine effiziente und strukturierte Zusammenarbeit im Entwicklungsteam zu gewährleisten.

- **Verwendung von GIT:** Jedes Teammitglied arbeitet an einem eigenen Branch, um Änderungen isoliert zu entwickeln und zu testen. Diese Branches werden regelmäßig mit dem **Upstream Branch** synchronisiert, um sicherzustellen, dass alle Entwicklerinnen auf dem neuesten Stand sind.
- **Commits:**
 - Nur fertige und lokal getestete Commits werden in den Upstream Branch gepushed. Vor dem Pushen wird jeder Commit auf Vollständigkeit und Funktionalität geprüft.
 - Die Commit-Nachrichten müssen klar und präzise formuliert sein, um den Zweck der Änderungen nachvollziehbar zu dokumentieren. Ein Beispiel für eine Commit-Nachricht könnte sein: *"Implementierung der User Story: Analyst möchte sensible Informationen via Google Dorking herausfinden"*.
- **Pull-Requests:** Vor dem Zusammenführen (Merge) eines Branches in den Upstream Branch wird ein **Pull-Request** erstellt. Dieser Pull-Request wird von mindestens einer weiteren

Entwicklerin überprüft, um die Codequalität und -sicherheit zu gewährleisten. Feedback wird dokumentiert und Änderungen werden gegebenenfalls vorgenommen, bevor der Merge erfolgt.

- **Branch-Strategie:**
 - Die Branch-Strategie folgt dem *Git Flow*-Modell, das die Erstellung von Feature-Banches für neue Funktionen und Bugfixes vorsieht.
 - Es gibt separate Branches für die Entwicklung, Tests und die Produktion, um eine klare Trennung der verschiedenen Entwicklungsphasen zu gewährleisten.
- **Dokumentation:**
 - Alle Änderungen und Integrationsprozesse werden dokumentiert, um Nachvollziehbarkeit und Transparenz im Projektverlauf sicherzustellen.

5.6.2 GIT – Workflow und Continuous Integration

Der verwendete Workflow: GitFlow

Für das Projekt WRAITH - OSINT wird der GitFlow-Workflow verwendet, um eine strukturierte und kontrollierte Softwareentwicklung zu gewährleisten.

Dieser Workflow unterscheidet sich durch die Verwendung mehrerer Branches für verschiedene Entwicklungsphasen:

- **main:** Enthält stets die stabile, freigegebene Version der Software.
- **master:** Enthält den aktuell getesteten Entwicklungsstand, in den alle Feature- und Bugfix-Banches zusammengeführt werden.
- **feature/*:** Für neue Funktionalitäten.

Repository-Konfiguration

- GIT-Anbieter: GitHub
- Zugriffsrechte:
 - Maintainer: Projektleiter/in, CI/CD-Admin
 - Developer: Entwickler/innen mit Push-Rechten auf Feature-Banches
 - Reporter: QA-Team für Review-Zwecke
- Repository-Sichtbarkeit: Privat

Branch-Benennung

- Feature-Branch: feature/REQ-XXX-kurze-beschreibung

Merge-Requests (MRs)

- Ein Merge-Request darf erst gestellt werden, wenn:
 - o Alle Systemtests erfolgreich durchlaufen wurden
 - o Der Code lokal getestet und dokumentiert wurde
 - o Die CI-Pipeline in GitLab erfolgreich ist
- Code-Reviews erfolgen durch mindestens eine andere Entwicklerin sowie ein finales Review durch den Projektleiter.
- Rückfragen und Korrekturen erfolgen über das GitLab-MR-Kommentarsystem.

Tagging (Versionierung)

- Tags enthalten eine Beschreibung
- Tags werden zur Generierung des Release-Notes-Dokuments verwendet.

5.6.3 Test der integrierten Software

Entstehende Artefakte (Deliverables):

Die Integrationsprüfung für das Projekt **WRAITH - OSINT** erfolgt mit dem Tool **RSpec (for Ruby)** für automatisierte Tests. Der Prozess der Integrationsprüfung ist wie folgt definiert:

Automatisierte Testumgebung:

Die Integrationsprüfung wird in einer lokal konfigurierten automatisierten Umgebung durchgeführt. Neue Commits im Repository lösen manuell oder durch einfache Skripte den Testprozess aus.

Testverfahren:

- Bei jedem neuen Commit wird lokal ein automatisierter Build- und Testprozess ausgeführt.
- Die Tests umfassen Systemtests, die mit **RSpec** erstellt wurden. Diese Tests stellen sicher, dass alle Module der Software korrekt zusammenarbeiten und die definierten Anforderungen erfüllen.
- Testfälle werden für jede User Story entwickelt, um spezifische Funktionalitäten zu überprüfen.

Fehlerbericht und Nachverfolgung:

- Wenn ein Test fehlschlägt, wird ein Fehlerbericht erstellt und dem Entwicklungsteam zur Verfügung gestellt.
- Die Entwickler:innen sind verantwortlich dafür, den Fehler zeitnah zu analysieren und zu beheben.

Regelmäßige Integrationsprüfungen:

- Zusätzlich zu den automatisierten Tests führt das Team regelmäßige Integrationsprüfungen durch, um die Funktionalität und Interoperabilität der verschiedenen Module sicherzustellen.
- Diese Prüfungen erfolgen im Rahmen der Sprint-Reviews, in denen alle Teammitglieder aktuelle Fortschritte präsentieren und etwaige Herausforderungen diskutieren.

5.6.4 Continuous Integration

Die CI-Umgebung basiert auf **GitHub Actions** RSpec (for Ruby).

Pipeline-Stages (GitHub Actions):

1. **Prepare**
 - Checkout des Repositories
 - Setup der Ruby-Umgebung mit der definierten Ruby-Version (.ruby-version oder Gemfile)
 - Installation der Abhängigkeiten über bundle install
2. **Build**
 - (Nur falls nötig) Vorbereitung des Builds oder Docker-Container
 - Setup von Testdaten oder Mocks
3. **Test**
 - Ausführen von CLI-Systemtests (RSpec)
4. **Review & QA**
 - Automatische Checks durch GitHub CI
 - Manuelles Code-Review durch die Projektleitung über Pull Requests
 - Optional: QA-Freigabe durch Qualitätsmanagement
5. **Deploy (optional)**
 - Deployment in eine Testumgebung bei Commits auf Release-/Branches
 - Nutzung von GitHub Environments für staging/test

5.7 Aktivität: Software Systemtests

Ziel:

Das Ziel der Software-Systemtests ist die Verifikation, dass sämtliche im Projekt spezifizierten Anforderungen durch geeignete Tests vollständig abgedeckt werden und die Software wie gefordert funktioniert.

5.7.1 Task: Erstellung von Tests für jede Softwareanforderung

1. **Anleitung zur Erstellung und Durchführung der Systemtests:**
 - a. Jede funktionale und nicht-funktionale Anforderung wird als Grundlage für mindestens einen spezifischen Testfall herangezogen.

2. Vorgehen zur Sicherstellung der Abdeckung:

- a. **Anforderungsanalyse:** Alle Anforderungen aus dem Lasten- und Pflichtenheft werden analysiert und in konkrete Testszenarien übersetzt.
- b. **Testfall-Design:** Auf Basis der Anforderungsanalyse werden Testfälle entwickelt, die mit **RSpec** implementiert werden. Diese decken sowohl positive als auch negative Szenarien ab. Randfälle und potenzielle Fehlerquellen werden besonders berücksichtigt.
- c. **Systemintegration:** Die Tests werden gegen die integrierte Ruby-Anwendung ausgeführt, inklusive aller Subsysteme, CLI-Tools und ggf. externen APIs.
- d. **Abnahmekriterien:** Ein Test gilt als erfolgreich, wenn die zugrundeliegende Anforderung vollständig und korrekt erfüllt wird.

3. Verifikation der Anforderungen:

- a. Die Tests werden gemäß agiler Prinzipien (Scrum) entwickelt, um eine kontinuierliche Verifikation und schnelles Feedback zu gewährleisten.
- b. Alle Testergebnisse werden dokumentiert, Fehlerberichte erstellt, und entsprechende Nachtests durchgeführt.

4. Entstehende Artefakte (Deliverables):

- **Testfälle:** Ausführliche Beschreibung aller Testfälle mit Testdaten, erwarteten Ergebnissen und Testprozeduren.
- **Fehlerberichte:** Dokumentation von Abweichungen und Vorschläge für Korrekturmaßnahmen.
- **Abnahmebericht:** Zusammenfassung, ob die Anforderungen erfolgreich erfüllt wurden und das System für die Auslieferung bereit ist.

6 Rollen und Verantwortlichkeiten:

- **Test-Designer:** Erstellen und validieren der Testfälle.
- **Test-Ingenieur:** Durchführung der Systemtests.
- **Projektleiter:** Sicherstellung, dass die Tests alle Anforderungen abdecken.

6.1 Schnittstelle zu anderen Prozessen

Die relevanten Schnittstellen sind wie folgt:

- **Software-Risikomanagementprozess:**

Der Software-Risikomanagementprozess identifiziert, bewertet und minimiert potenzielle Risiken, die während der Softwareentwicklung auftreten können.

Während der Anforderungsanalyse und des Designs werden Risiken systematisch dokumentiert und in ein Risikoregister aufgenommen.

Die Risikobewertung wird in regelmäßigen Abständen während der Entwicklung überprüft, um neue Risiken zu identifizieren und die bereits identifizierten Risiken zu bewerten.

- **Software-Konfigurationsmanagementprozess:**

Der Software-Konfigurationsmanagementprozess stellt sicher, dass alle Softwareartefakte, einschließlich Quellcode, Dokumentation und Testfälle, ordnungsgemäß versioniert und verwaltet werden.

Die Verwendung von Git als Codeverwaltungssoftware ermöglicht eine effiziente Nachverfolgung von Änderungen und die Verwaltung von Versionen.

Regelmäßige Reviews und Audits der Konfiguration helfen, die Integrität und Konsistenz der Software über den gesamten Entwicklungszyklus hinweg sicherzustellen.

- **Problemlösungsprozess für Software:**

Der Problemlösungsprozess für Software befasst sich mit der Identifizierung, Analyse und Behebung von Fehlern und Problemen, die während der Entwicklung oder im Betrieb der Software auftreten können.

Der Problemlösungsprozess umfasst auch die Durchführung von Root-Cause-Analysen, um die zugrunde liegenden Ursachen von Problemen zu identifizieren und sicherzustellen, dass entsprechende Maßnahmen ergriffen werden, um ähnliche Probleme in der Zukunft zu vermeiden.

7 Glossar

Begriffe / Abkürzung	Definition
Aktivität	Satz von einer oder mehreren aufeinander bezogenen oder sich gegenseitig beeinflussenden Aufgaben.
Anomalie	Jeglicher Zustand, der abweicht von dem, was auf Grund von Anforderungsspezifikationen, Entwicklungsdokumenten, Normen usw. oder von Wahrnehmungen oder Erfahrungen zu erwarten ist. Anomalien können sich unter anderem beim Review, bei der Prüfung, der Analyse, der Kompilierung oder bei der Benutzung von Software-Produkten oder zugehöriger Dokumentation finden.
Architektur	Organisationsstruktur eines Systems oder einer Komponente
Änderungsanforderung	Dokumentierte Spezifikation einer Änderung, die an einem Software-Produkt durchgeführt werden soll.
Konfigurationselement	Einheit, die an einem gegebenen Referenzpunkt eindeutig identifiziert werden kann
Zu lieferndes Ergebnis	Gefordertes Resultat oder Ergebnis (einschließlich Dokumentation) einer Aktivität oder Aufgabe
Evaluation	Systematische Bewertung oder Beurteilung, inwieweit eine Einheit ihre festgelegten Merkmale erfüllt
Schaden	Physische Verletzung oder Schädigung der Gesundheit von Menschen oder Schädigung von Gütern oder der Umwelt.
Gefährdung	Potentielle Quelle eines Schadens
Hersteller	natürliche oder juristische Person, die für die Auslegung, Herstellung, Verpackung oder Kennzeichnung eines MEDIZINPRODUKTS, für den Zusammenbau eines SYSTEMS oder für die Anpassung eines MEDIZINPRODUKTS vor dem Inverkehrbringen oder der Inbetriebnahme verantwortlich ist, unabhängig davon, ob diese Tätigkeiten von dieser Person selbst oder stellvertretend für diese von einer dritten Person ausgeführt werden.
MEDIZINPRODUKT	Alle einzeln oder miteinander verbunden verwendeten Instrumente, Apparate, Vorrichtungen, Stoffe oder anderen Gegenstände einschließlich der für ein einwandfreies Funktionieren des MEDIZINPRODUKTS eingesetzten Software, die vom HERSTELLER zur Anwendung für Menschen für folgende Zwecke bestimmt sind: -- Erkennung, Verhütung, Erfassen, Behandlung oder Linderung von Krankheiten, – Erkennung, Erfassung, Behinderungen, – Untersuchung, Ersatz oder Veränderung des anatomischen Aufbaus oder eines physiologischen Vorgangs, – Lebenserhaltung oder Lebensunterstützung, – Empfängnisregelung, – Desinfektion von MEDIZINPRODUKTEN, – Bereitstellung von Informationen für medizinische Zwecke mittels In-vitro-Untersuchung von Probestücken des menschlichen Körpers, Behandlung, Linderung oder Kompensierung von Verletzungen oder und deren bestimmungsgemäße Hauptwirkung im oder am menschlichen Körper weder durch pharmakologische noch durch immunologische oder metabolische Mittel erreicht wird, deren Wirkungsweise aber durch solche Mittel unterstützt werden kann .
MEDIZINPRODUKTE-SOFTWARE	ein Softwaresystem, das entwickelt wurde, um in das in der Entwicklung befindliche MEDIZINPRODUKT integriert zu werden, oder das für die Benutzung als selbständiges MEDIZINPRODUKT vorgesehen ist.
PROBLEMBERICHT	Aufzeichnung über ein tatsächliches oder mögliches Verhalten eines SOFTWARE-PRODUKTS, von dem ein Anwender oder eine andere beteiligte Person glaubt, dass es unsicher, ungeeignet für den bestimmungsgemäßen Gebrauch oder widersprüchlich zur Spezifikation sei.

PROZESS	Satz von in Wechselbeziehung oder Wechselwirkung stehenden AKTIVITÄTEN, der Eingaben in Ergebnisse umwandelt
REGRESSIONSPRÜFUNG	Prüfung, die erforderlich ist um festzustellen, dass eine Änderung einer SYSTEM-Komponente deren Funktionalität, Zuverlässigkeit und Leistungsfähigkeit nicht beeinträchtigt und keine zusätzlichen Defekte verursacht hat.
Risiko	Kombination der Wahrscheinlichkeit des Auftretens eines SCHADENS und des Schweregrades dieses SCHADENS.
RISIKOANALYSE	systematische Auswertung verfügbarer Informationen, um Gefährdungen zu identifizieren und RISIKEN abzuschätzen.
RISIKOBEHERRSCHUNG	PROZESS, in dem Entscheidungen getroffen und RISIKEN auf festgelegte Grenzen reduziert oder in diesen gehalten werden.
RISIKOMANAGEMENT	systematische Anwendung von Managementgrundsätzen, Verfahren und Praktiken auf die Analyse, Bewertung und Kontrolle von RISIKEN.
RISIKOMANAGEMENT-AKTE	Zusammenstellung von Aufzeichnungen und sonstigen Dokumenten, die durch den RISIKOMANAGEMENT-PROZESS erzeugt werden und nicht notwendigerweise an einer Stelle sein müssen.
SICHERHEIT	Freiheit von unvermeidbaren RISIKEN
DATENSICHERHEIT	Schutz von Informationen und Daten derart, dass nicht autorisierte Personen oder SYSTEME sie nicht lesen oder verändern können und dass autorisierten Personen und SYSTEMEN der Zugang zu ihnen nicht verweigert wird.
SCHWERE VERLETZUNG	Verletzung oder Krankheit, die direkt oder indirekt a) lebensbedrohend ist, b) zu einer andauernden Beeinträchtigung einer Körperfunktion oder zu einer andauernden Schädigung des (menschlichen) Körpers führt oder c) ein medizinisches oder chirurgisches Eingreifen erfordert, um eine andauernde Beeinträchtigung einer Körperfunktion oder eine andauernde Schädigung des (menschlichen) Körpers zu verhindern
LEBENSZYKLUS-MODELL DER SOFTWARE-ENTWICKLUNG	konzeptionelle Struktur, welche die Lebenszeit der Software von der Anforderungsdefinition bis zur Freigabe für die Produktion umfasst und die – die PROZESSE, AKTIVITÄTEN und AUFGABEN festlegt, die an der Entwicklung eines SOFTWARE-PRODUKTS beteiligt sind, – die Reihenfolge und die Abhängigkeit von AKTIVITÄTEN und AUFGABEN beschreibt und – die Meilensteine festlegt, bei denen die Vollständigkeit der festgelegten verifiziert wird
Softwarekomponente	jedes identifizierbare Teil eines Computerprogramms
SOFTWARE-PRODUKT	Satz von Computerprogrammen, Prozeduren und möglicherweise zugeordneten Dokumentationen und Daten
Softwaresystem	integrierte Sammlung von Softwarekomponenten, die so organisiert sind, dass eine spezifische Funktion oder ein Satz von Funktionen ausgeführt werden kann.
SOFTWARE-EINHEIT	Softwarekomponente, die nicht in weitere Komponenten unterteilt ist
SOUP (en: SOFTWARE OF UNKNOWN PROVENANCE) SOFTWARE UNBEKANNTER HERKUNFT	Softwarekomponente, die bereits entwickelt und allgemein verfügbar ist und die nicht entwickelt wurde, um in das MEDIZINPRODUKT integriert zu werden (auch bekannt als „Off-The-Shelf Software“), oder bereits fertig entwickelte Software, für die angemessene Aufzeichnungen zum Entwicklungs-PROZESS nicht verfügbar sind.
SYSTEM	integrierter Verbund, bestehend aus einem oder mehreren PROZESSEN, Hardware, Software, Einrichtungen und Personen, der eine Fähigkeit zur Verfügung stellt, um einen bestimmten Zweck oder ein bestimmtes Ziel zu erreichen.
AUFGABE	eine Teilarbeit, die erledigt werden muss
RÜCKVERFOLGBARKEIT	die Möglichkeit, eine Beziehung zwischen zwei oder mehr Produkten des Entwicklungs-PROZESSES herstellen zu können.
VERIFIZIERUNG	Bestätigung durch Bereitstellen eines objektiven Nachweises, dass festgelegte Anforderungen erfüllt worden sind.

VERSION	gekennzeichnete Ausgabe eines KONFIGURATIONSELEMENTES
JIRA	weit verbreitetes Projektmanagement- und Issue-Tracking-Tool, das von Atlassian entwickelt wurde. Es wird häufig in der Softwareentwicklung verwendet, um Aufgaben, Bugs und Projekte zu verwalten.
Root-Cause	bezieht sich auf die zugrunde liegende Ursache eines Problems oder einer Fehlfunktion. Es ist entscheidend, die Root Cause zu identifizieren, um sicherzustellen, dass ähnliche Probleme in der Zukunft vermieden werden.
GitHub	Eine Plattform zur Versionsverwaltung und Zusammenarbeit an Codeprojekten mit Git.
RSpec	Ein Testing-Framework für Ruby, das behavior-driven Tests ermöglicht.
Ruby	Eine objektorientierte Programmiersprache, bekannt für einfache und lesbare Syntax.
Google - Dorking	Eine Technik zur gezielten Informationssuche mit speziellen Google-Suchbefehlen.