HOCHSCHULE FULDA
UNIVERSITY OF APPLIED SCIENCES

Dokumentation und Versionskontrolle in der Softwareentwicklung bei Grenzebach BSH

Luca Michael Schmidt

Hausarbeit am Fachbereich AI der HS Fulda

Matrikelnummer: 1540963
Erstbegutachtung: Prof. Dr.-Ing. J. Konert

Eingereicht am 11.0 6.2025

Sperrvermerk

Die vorliegende Arbeit beinhaltet interne vertrauliche Informationen der Firma ”Grenzebach BSH GmbH”. Die Weitergabe des Inhalts der Arbeit und Daten im Gesamten oder in Teilen ist grundsätzlich untersagt. Es dürfen keinerlei Kopien oder Abschriften - auch in digitaler Form - gefertigt werden. Ausnahmen bedürfen der schriftlichen Genehmigung der Firma Grenzebach BSH GmbH.

Inhaltsverzeichnis

1 Einleitung

1.1 Relevanz von Dokumentation und Versionskontrolle

Die systematische Dokumentation und konsequente Versionskontrolle sind essenzielle Bestandteile moderner Softwareentwicklung. Sie erhöhen die Effizienz von Entwicklungsteams, verbessern die Softwarequalität und ermöglichen eine langfristige Planbarkeit von Wartung und Weiterentwicklung. Eine aktuelle und verständliche Dokumentation reduziert den Zeitaufwand für die Informationssuche und beugt Fehlern vor, die zu unerwartetem Mehraufwand führen können [1]. Dokumentierte Anforderungen dienen als Diskussions- und Entscheidungsgrundlage und fördern die Kommunikation im Team, insbesondere in agilen Kontexten [2]. Die Versionskontrolle sichert den aktuellen Arbeitsstand und ermöglicht durch Branching und Merging eine parallele Feature-Entwicklung, wodurch der Entwicklungsprozess beschleunigt wird [3]. Teams, die Versionskontrollsysteme effektiv nutzen, produzieren tendenziell weniger Fehler und beheben diese schneller. In Kontexten mit schnellen Iterationen und kontinuierlicher Auslieferung sind diese Praktiken unverzichtbar, da sie Stabilität und Nachvollziehbarkeit gewährleisten.

1.2 Unternehmenskontext: Grenzebach BSH im Überblick

Die Grenzebach BSH GmbH mit Sitz in Bad Hersfeld ist Teil der international agierenden Grenzebach-Gruppe. Das 1960 gegründete Unternehmen ist ein globaler Spezialist für Anlagenbau und Automatisierungstechnik. Es beschäftigt weltweit rund 1.600 Mitarbeiter an sieben Standorten und ist in über 55 Ländern aktiv [4]. Grenzebach BSH entwickelt und fertigt schwerpunktmäßig Industrieanlagen für die Baustoffe Gips und Holz. Neben der in diese Anlagen integrierten Steuerungssoftware gewinnt die Entwicklung von Softwareerweiterungen, spezialisierten Industrie-Apps und unterstützenden Softwarelösungen zunehmend an Bedeutung. Die Dokumentations- und Versionsverwaltungsprozesse für diese Softwareentwicklungsprojekte stehen im Zentrum dieser Arbeit.

1.3 Zielsetzung und Aufbau der Arbeit

Bei Grenzebach BSH bestehen Optimierungspotenziale in den Prozessen der Dokumentation und Versionskontrolle. Diese Hausarbeit analysiert die gegenwärtigen Praktiken, identifiziert Schwachstellen und entwickelt konkrete Handlungsempfehlungen.
Die Arbeit ist wie folgt aufgebaut: Kapitel 2 legt die theoretische Basis mit grundlegenden Konzepten, Methoden und aktuellen Standards der Dokumentation und Versionskontrolle. Anschließend erfolgt eine detaillierte Ist-Analyse bei Grenzebach BSH. Kapitel 3 untersucht die Dokumentationspraxis, während Kapitel 4 die Versionskontrollprozesse evaluiert. Beide Kapitel identifizieren Stärken und Schwächen und vergleichen

die Praktiken mit Branchenstandards. Darauf aufbauend präsentiert Kapitel 5 spezifische Optimierungsvorschläge und Handlungsempfehlungen. Abschließend fasst Kapitel 6 die wesentlichen Ergebnisse zusammen, reflektiert diese kritisch und gibt einen Ausblick auf mögliche nächste Schritte.

2 Theoretische Grundlagen

Dieses Kapitel erläutert die zentralen Konzepte, Methoden und Werkzeuge der Softwaredokumentation und Versionskontrolle als Fundament für die spätere Analyse und Entwicklung von Optimierungsvorschlägen.

2.1 Dokumentation in der Softwareentwicklung

Die Softwaredokumentation umfasst alle schriftlichen und grafischen Materialien, die während des Softwarelebenszyklus erstellt werden. Sie beschreibt das Softwaresystem, dessen Entwicklungsprozess und Nutzung. Als integraler Bestandteil der Softwareentwicklung ist sie entscheidend für Erfolg, Qualität und langfristige Wartbarkeit. Zu den primären Zielen zählen: effektive Kommunikation innerhalb des Teams und mit externen Stakeholdern, strukturierte Wissensspeicherung für Teamwechsel und Onboarding sowie Qualitätssicherung durch klare Spezifikationen und nachvollziehbare Entwicklungsschritte [1]. Zusätzlich unterstützt sorgfältige Dokumentation die Wartung und Weiterentwicklung bestehender Systeme und erfüllt vertragliche oder regulatorische Anforderungen.
Grundsätzlich unterscheidet man zwischen Prozess- und Produktdokumentation. Die Prozessdokumentation beschreibt den Entwicklungsprozess selbst (z.B. Projektpläne, Meeting-Protokolle). Die Produktdokumentation fokussiert sich auf das zu erstellende oder erstellte Softwaresystem und umfasst verschiedene Artefakte:

- **Anforderungsdokumentation:** Definiert funktionale Anforderungen durch User Stories oder Use-Case-Diagramme. In agilen Umfeldern ist sie entscheidend zur Vermeidung von Missverständnissen und zur Entwicklung einer gemeinsamen Produktvision [2].
- **Architektur- und Design-Dokumentation:** Beschreibt die Systemstruktur, Komponenten und deren Zusammenspiel sowie Entwurfsentscheidungen als Implementierungsleitfaden.
- **Quellcode-Dokumentation:** Umfasst Code-Kommentare und automatisch generierbare API-Dokumentationen für besseres Verständnis einzelner Module und Funktionen.
- **Testdokumentation:** Beinhaltet Testpläne, Testfallspezifikationen, Testdaten und Testprotokolle zur systematischen Qualitätsüberprüfung.

- **Benutzerdokumentation:** Erklärt Endanwendern die Bedienung und Funktionalität durch Handbücher, Online-Hilfen oder Tutorials.
- **Betriebsdokumentation:** Enthält Informationen für Installation, Konfiguration und laufenden Betrieb der Software.

Typische Herausforderungen sind die kontinuierliche Aktualisierung bei sich entwickelnden Systemen, die Balance zwischen zu viel und zu wenig Dokumentation sowie die Förderung von Akzeptanz und konsequenter Nutzung [1]. Qualitativ hochwertige Dokumentation zeichnet sich durch klare Struktur, zielgruppengerechte Inhalte und Prozesse zur Qualitätssicherung aus. Die ISO/IEC/IEEE 26514 betont dabei die Notwendigkeit von Zielgruppen- und Tätigkeitsanalysen sowie zielgruppenzentrierten Informationskonzepten [5].

2.2 Versionskontrollsysteme und -prozesse

Ein Versionskontrollsystem (VCS) ist ein fundamentales Werkzeug der Softwareentwicklung. Es protokolliert und verwaltet Änderungen an Dateien systematisch über die Zeit. Der VCS-Einsatz ist unerlässlich für effektive Teamkoordination, lückenlose Änderungshistorie und Quellcode-Integrität [6]. Ohne VCS drohen unbeabsichtigtes Überschreiben von Änderungen, Schwierigkeiten beim Zusammenführen von Arbeitsständen oder unwiderruflicher Verlust funktionierender Versionen [3].
Moderne VCS basieren auf Kernkonzepten mit essenziellen Funktionalitäten. Das Repository speichert alle Projektdateien mit vollständiger Historie. Jede gesicherte Änderung heißt Commit und umfasst geänderte Dateien, Metadaten (Autor, Zeitpunkt) sowie eine erläuternde Commit-Nachricht. Branches ermöglichen parallele Arbeit an Features, isolierte Fehlerbehebung oder Experimente ohne direkten Eingriff in den Hauptzweig (`main` oder `master`). Die Integration verschiedener Branches erfolgt durch Merge-Vorgänge. Tags markieren spezifische Stände für Releases. Diffs visualisieren Versionsunterschiede, und Rollbacks ermöglichen die Rückkehr zu früheren Ständen.
Der Git-Arbeitszyklus umfasst typischerweise: initiales Kopieren eines Remote-Repositories (`git clone`), Vorbereiten geänderter Dateien (Staging mit `git add`), lokales Speichern als Commit (`git commit`), Übertragen an Remote-Repository (`git push`) und Abrufen sowie Integrieren von Remote-Änderungen (`git pull` oder `git fetch` mit `git merge`).

2.3 Aktuelle Standards und Best Practices in der Industrie

Die Industrie hat diverse Standards und Best Practices entwickelt, um Dokumentation und Versionskontrolle optimal zu nutzen.
Moderne Dokumentationsansätze zielen auf Flexibilität, Aktualität und bessere Prozessintegration. Agile Dokumentationsprinzipien betonen “gerade genug” Dokumentation mit Fokus auf funktionierende Software und direkte Kommunikation. User Stories

mit präzisen Akzeptanzkriterien spielen oft eine zentrale Rolle [7, 8]. "Documentation as Code" behandelt Dokumentation wie Softwarequellcode: Sie wird in Markup-Sprachen (z.B. Markdown, AsciiDoc) verfasst, im VCS gespeichert, durchläuft Review-Prozesse und kann automatisiert in verschiedene Formate überführt werden [9].
Für Versionskontrollsysteme, insbesondere Git, haben sich etablierte Praktiken durchgesetzt. Dies beginnt bei geeigneten Branching-Modellen wie dem strukturierten Gitflow oder dem agileren Trunk-Based Development [10]. Standardisierte Commit-Nachrichten nach "Conventional Commits" sorgen für klare, maschinell auswertbare Historie [11]. Die Qualitätssicherung erfolgt durch Code Reviews mittels Pull-Requests und transparente Versionierung durch Semantic Versioning (`MAJOR.MINOR.PATCH`) [12, 13]. Die Integration in CI/CD-Pipelines ermöglicht effiziente und zuverlässige Softwareauslieferung.

2.4 Relation zwischen Dokumentation und Versionskontrolle

In der modernen Softwareentwicklung sind Dokumentation und Versionskontrolle eng verzahnt und entfalten erhebliche Synergien. Die Verwaltung von Dokumentationsartefakten mittels VCS ist besonders für den "Documentation as Code"-Ansatz relevant. Textbasierte Dokumente werden gemeinsam mit dem Quellcode versioniert. Durch Branching und Merging können Dokumentationsänderungen parallel zu Code-Features entwickelt und mittels Pull-Requests gemeinsam reviewed werden, was Konsistenz und Nachvollziehbarkeit sicherstellt.
Aussagekräftige Commit-Nachrichten, idealerweise nach Standards wie "Conventional Commits" [11], dienen als granulare Dokumentation der Änderungshistorie. Moderne Entwicklungsplattformen ermöglichen die Verknüpfung von Dokumentation mit spezifischen Features oder Issues, was das Auffinden kontextbezogener Informationen erleichtert.
Die integrierte Nutzung verbessert Wartbarkeit, Qualität und Transparenz von Softwareprojekten und erleichtert das Onboarding. VCS-gestützte Prozesse, bei denen Dokumentationsanpassungen im selben Zyklus wie Code-Änderungen behandelt werden, meistern die Herausforderung der Synchronisation zwischen dynamischem Code und aktueller Dokumentation.

3 Analyse der Dokumentationspraxis bei Grenzebach BSH

Dieses Kapitel untersucht detailliert die aktuelle Praxis der Erstellung, Verwaltung und Nutzung von Softwaredokumentation bei Grenzebach BSH. Ziel ist ein klares Bild der etablierten Prozesse und eingesetzten Werkzeuge zur Identifikation von Stärken, Schwächen und Optimierungspotenzialen.

3.1 Aktuelle Vorgehensweisen und Werkzeuge

Die Dokumentationspraxis für Softwareprojekte bei Grenzebach BSH basiert auf einer Kombination aus Confluence für übergreifende Projektinformationen und GitHub für technische sowie code-nahe Dokumentation. Diese Zweiteilung richtet sich primär an interne Entwicklerteams.
Für übergeordnete und konzeptionelle Dokumentation neuer Projekte werden Projektseiten in Confluence nach standardisierten Vorlagen angelegt. Diese dienen als zentrale Ablage für verschiedene Artefakte: textuelle Beschreibungen, DrawIO-Grafiken und -Diagramme, Screenshots, Click-Dummies, Prototypen, Wireframes, Interface-Spezifikationen und Besprechungsnotizen. Die kollaborative Erstellung und Pflege erfolgt durch mehrere Teammitglieder. Das jeweils verantwortliche Mitglied sichert Richtigkeit und Qualität durch Proofreading. Die Dokumentation ist regelmäßiger Bestandteil wöchentlicher Scrum-Meetings, in denen Verbesserungen und Aktualisierungen diskutiert werden.
Die technische Dokumentation ist eng mit GitHub und den CI/CD-Prozessen via GitHub Actions verknüpft. Jedes Projekt pflegt einen spezifischen "Doc-Branch" im Repository mit HTML-Dateien und eingebundenen JSON-Daten als generierte technische Dokumentation. `README.md`-Dateien mit grundlegenden Projektinformationen (Zweck, Nutzung, Installation, Besonderheiten) werden nach Vorlage erstellt, ins HTML-Format überführt und integriert. GitHub Actions aktualisieren automatisch Badges in `README.md`-Dateien für Versionsstand, Testergebnisse und Sicherheitsstatus. Die generierte Dokumentation, die auch extrahierte Code-Kommentare und Kontext aus dem Quellcode umfasst, wird auf einem internen Server publiziert. Bei komplexen Sachverhalten verlinkt sie direkt zu detaillierteren Confluence-Seiten. Die Code-Kommentierung durch Entwickler bildet stets die grundlegendste Dokumentationsebene.
Grenzebach BSH verfolgt einen zweigeteilten Ansatz: Confluence als Wissensbasis für umfassende, grafisch-konzeptionelle Projektinformationen, die kollaborativ erstellt und regelmäßig besprochen werden. GitHub als Zentrum für technische, versionskontrollierte Dokumentation, die teilweise automatisiert generiert, durch CI/CD aktuell gehalten und bei Bedarf mit Confluence verknüpft wird.

3.2 Stärken und Schwächen der bisherigen Praxis

Die Dokumentationspraxis bei Grenzebach BSH weist sowohl unterstützende Stärken als auch Bereiche mit erkennbaren Herausforderungen und Optimierungspotenzialen auf.
Zu den Stärken zählt die klare Aufgabenteilung zwischen Confluence für übergreifende Informationen und GitHub für code-nahe Dokumentation. Confluence dient als zentrale Ablage für modulübergreifende Artefakte. Die Verwendung von Vorlagen in Confluence ist ein erster Schritt zu mehr Einheitlichkeit. Die automatische HTML-Generierung und Status-Badge-Integration in `README.md`-Dateien bietet Entwicklern eine schnell erfassbare Projektübersicht. Das Team ist bestrebt, diese Indikatoren positiv zu halten. Die Verlinkungsmöglichkeit von Code-Kommentaren zu erläuternden Confluence-Inhalten

wird als hilfreich für das Verständnis komplexer Zusammenhänge empfunden.
Demgegenüber stehen Schwächen und Herausforderungen. Die Aufteilung auf Confluence und GitHub führt trotz Verlinkungen zu potenziellen Problemen ohne automatisierte Synchronisation. Dies kann Inkonsistenzen verursachen: veraltete Code-Kommentare, `README.md`-Dateien oder Confluence-Seiten sowie tote Links [1]. Klare Regeln zur führenden Informationsquelle fehlen, wobei tendenziell Confluence für allgemeine und GitHub für code-spezifische Informationen genutzt wird. Die Qualität der Code-Kommentare variiert stark je nach Teammitglied. Obwohl Confluence-Seiten generell gut gepflegt sind, bleibt die kontinuierliche Aktualisierung eine Herausforderung. Die Dokumentationsthematisierung in Scrum-Meetings fokussiert häufiger auf Tests. Nicht alle Teammitglieder zeigen gleiches Engagement für Dokumentationsaufgaben. Wichtige Aspekte werden teilweise auch auf Rückfrage nicht ergänzt. Eine vollständige Abdeckung aller Dokumentationsarten (siehe 2.1), insbesondere für Endanwender, fehlt, während andere Bereiche gut abgedeckt sind.

3.3 Vergleich mit Branchenstandards

Der Vergleich mit den Branchenstandards aus Kapitel 2 zeigt sowohl fortschrittliche Ansätze als auch Weiterentwicklungspotenziale.
Positiv sind die GitHub-Actions-basierte HTML-Generierung, die Versionierung im "Doc-Branch" und die Code-Kommentar-Extraktion hervorzuheben. Diese Aspekte implementieren Elemente des "Documentation as Code"-Prinzips [9]. Die automatische Generierung aus dem reviewten `develop`-Branch ohne manuelle Überarbeitung fördert Konsistenz. Die Nutzung von `README.md`-Dateien mit automatisierten Status-Badges entspricht modernen Praktiken. Die Dokumentationsbesprechung in Scrum-Meetings unterstützt agile Grundsätze [7, 8]. Die einheitliche DrawIO-Nutzung für Confluence-Diagramme ist ebenfalls positiv.
Optimierungspotenzial zeigt sich in der Durchgängigkeit des "Documentation as Code"-Ansatzes. Werkzeuge, die Markdown oder AsciiDoc direkt aus dem Haupt-Repository verarbeiten (wie Sphinx oder MkDocs), könnten den separaten "Doc-Branch" reduzieren und eine engere Code-Verzahnung ermöglichen. Dies würde Review-Prozesse vereinfachen, da Dokumentationsänderungen im selben Diff wie Code-Änderungen sichtbar wären. Die HTML-Bereitstellung auf einem internen Webserver ist funktional, könnte aber durch integrierte GitHub-Wiki-Funktionen oder spezialisierte Doku-Hosting-Plattformen optimiert werden.
Die Trennung auf Confluence und GitHub-generierte Seiten weicht potenziell vom "Single Source of Truth"-Ideal ab ohne klare Prozesse für führende Quellen und Aktualitätssicherung [1]. Die variierende Dokumentationsqualität je nach Teammitglied deutet auf fehlende verbindliche Richtlinien hin. Eine systematische Qualitätssicherung über Proofreading hinaus, orientiert an ISO/IEC/IEEE 26514 [5], fehlt noch. Eine automatisierte Verlinkungsprüfung zwischen Confluence und GitHub wäre eine Annäherung an Best Practices für wartbare Dokumentationssysteme.

4 Evaluation der Versionskontrolle bei Grenzebach BSH

Nach der Dokumentationsanalyse widmet sich dieses Kapitel der detaillierten Evaluation der Versionskontrollsysteme und zugehörigen Workflows bei Grenzebach BSH. Ziel ist die Beschreibung aktueller Prozesse, Identifikation von Stärken und Schwächen sowie der Vergleich mit Branchenstandards.

4.1 Eingesetzte Versionskontrollsysteme und Workflows

Die Versionskontrolle basiert auf Git mit GitHub als zentraler Hosting-Plattform. Ergänzend kommen GitHub Actions für CI/CD und Jira für das Projektmanagement zum Einsatz. Spezielle Git-Clients über die IDE-integrierten Funktionen (JetBrains, Visual Studio Code) hinaus werden nicht verwendet. Die Git- und GitHub-Nutzung ist im Team etabliert und wird wie vorgesehen umgesetzt.
Die Projekterstellung folgt einem standardisierten Prozess: Ein Git-Template wird heruntergeladen und ein Setup-Skript ausgeführt. Dies generiert die Git-Umgebung, erstellt das GitHub-Repository mit Zugriffsrechten, legt einen "Doc-Branch" an, richtet GitHub Actions ein und passt `README.md` sowie Standarddateien (`.gitignore`, `LICENSE`) an. Dieser Ablauf funktioniert reibungslos und spart Zeit bei der Projekteinrichtung.
Das Branching-Modell ist klar definiert und wird konsequent eingehalten. Der `develop`-Branch ist der Hauptentwicklungszweig. Feature-Branches folgen dem Schema `GBV-xxx` und beziehen sich auf Jira-Tickets. Releases werden durch Tags im Format `vx.x.x` markiert. Diese werden manuell gesetzt, woraufhin die CI/CD-Pipeline deren Validität prüft und einen formalen GitHub-Release erstellt.
Pull Requests (PRs) sind zentraler Workflow-Bestandteil. Änderungen aus Feature-Branches werden nicht direkt gemerged, sondern durchlaufen einen PR-Prozess. Der Feature-Entwickler initiiert den PR, der ein Code-Review durch mindestens einen Kollegen erfordert, typischerweise jemand mit Feature-Kenntnis. Nach Zustimmung erfolgt der Merge in `develop`. Dieser Prozess wird konsequent befolgt.

Abbildung 2 visualisiert den beschriebenen Git-Workflow mit dem etablierten Branching-Modell und dem integrierten Review-Prozess.

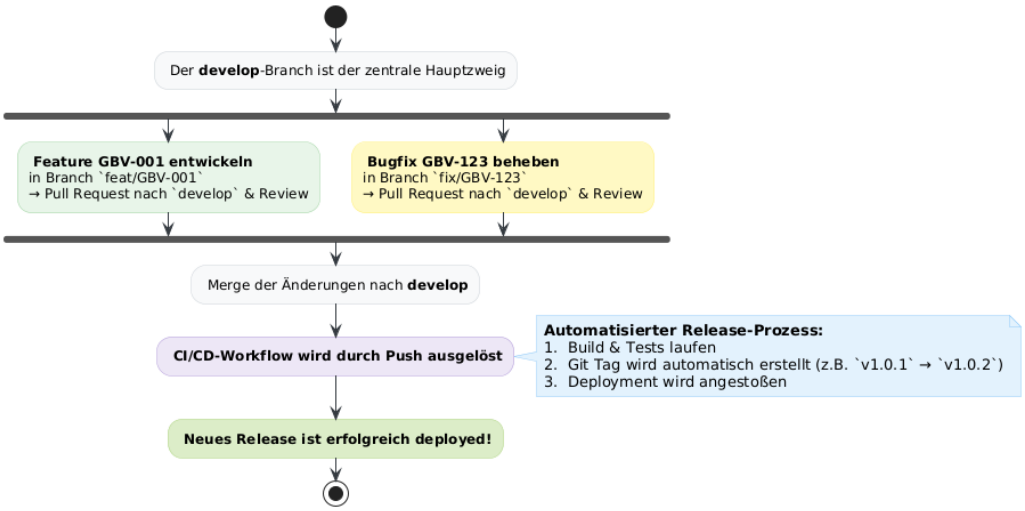

Abbildung 1: Git-Workflow bei Grenzebach BSH: Feature- und Bugfix-Branches (GBV-xxx) durchlaufen einen Pull-Request-Prozess mit Review vor dem Merge in develop. Ein Push auf develop löst automatisch den CI/CD-Workflow und Release-Prozess aus

Die CI via GitHub Actions ist an Commits auf `develop` gekoppelt. Die Pipeline umfasst: Projektinformationsextraktion und `README.md`-Aktualisierung, Code-Linting und -Formatierung, Sicherheitsprüfung, automatisierte Tests und Dokumentationsgenerierung mittels `pdoc`. CI-Rückmeldungen sind meist zielführend; bei komplexen Fehlern ist der Pipeline-Verantwortliche zuständig. Die konkrete Ausgestaltung der CI/CD-Pipeline mit allen automatisierten Schritten und deren Abhängigkeiten zeigt Abbildung 2.

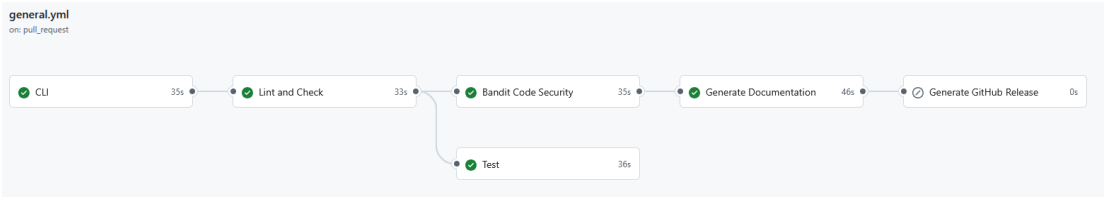

Abbildung 2: GitHub Actions CI/CD-Pipeline (general.yml) bei Grenzebach BSH: Sequenzielle Ausführung von CLI-Checks, Code-Linting, Sicherheitsprüfung mit Bandit, Tests und Dokumentationsgenerierung mit finaler Release-Erstellung

Die CD erfolgt ebenfalls über GitHub Actions nach erfolgreichem `develop`-Push und beinhaltet das Deployment auf einen internen Server. Nach Release-Tag-Erstellung wird die Software auf einem internen PyPi-Server bereitgestellt. Dieser aktualisiert sich selbstständig, was GitHub-unabhängige Bibliotheksinstallationen ermöglicht.
Die Jira-Verknüpfung erfolgt primär über Branch-Benennung nach Ticketnummern (`GBV-xxx`), was die Nachverfolgbarkeit erleichtert. Eine automatisierte Jira-GitHub-Synchronisation

(z.B. automatische Kommentare oder Statusänderungen) ist nicht implementiert; die Information über PRs erfolgt manuell.

4.2 Stärken und Schwächen der aktuellen Prozesse

Die Analyse der Versionskontrollprozesse zeigt Stärken, die zu effizienter und qualitativ hochwertiger Entwicklung beitragen, aber auch Optimierungsbereiche.
Wesentliche Stärken sind die konsequente Git- und GitHub-Nutzung als zentrale Plattform. Der GitHub-Funktionsumfang, insbesondere CI/CD über GitHub Actions mit eigenen Unternehmens-Runnern, wird als vorteilhaft und kosteneffizient empfunden. Die gute Git-Integration in IDEs unterstützt einen reibungslosen Workflow. Der standardisierte Projekterstellungsprozess mittels Templates und Setup-Skripten spart Zeit und sorgt für konsistente Projektstrukturen, primär für Python-Projekte. Das etablierte Branching-Modell mit klarer Trennung von Entwicklungs-, Feature- und Release-Zweigen fördert stabiles Release-Management und organisierte Feature-Entwicklung.
Die weitgehende CI/CD-Automatisierung ist eine weitere Stärke. Automatisierte Tests, Linting und Security Checks erhöhen die Code-Qualität und geben frühes Feedback. Die automatische Dokumentationsgenerierung und `README.md`-Aktualisierung mit Status-Badges erhöht Transparenz und Aktualität wichtiger Projektinformationen. Der obligatorische PR-Prozess mit Code-Review ist ein wichtiger Qualitätssicherungs- und Wissenstransfermechanismus. Entwickler fühlen sich durch VCS-Prozesse und Automatisierung gut unterstützt, während die CI/CD-Wartung bei einem dedizierten Entwickler liegt.
Trotzdem existieren Schwächen und Herausforderungen. Das Debugging von GitHub-Actions-Fehlern kann komplex und zeitaufwendig sein. Die Test-Ausführungsgeschwindigkeit könnte durch stärkere Parallelisierung verbessert werden, wofür jedoch zeitliche Ressourcen fehlen. Strenge Linting-Regeln werden gelegentlich als hinderlich empfunden, obwohl sie grundsätzlich positiv für die Code-Qualität sind.
Bei Code-Reviews findet zwar ein Review statt, formalisierte Richtlinien oder Checklisten fehlen jedoch (außer erfolgreicher CI-Pipeline). Dies kann zu variierender Review-Tiefe führen. Für kritische Änderungen könnte ein einzelner Reviewer unzureichend sein. Die Commit-Nachrichten-Qualität ist mittelmäßig ohne verbindliche Regeln, was zu vagen Nachrichten oder umfangreichen Commits führt.
Die fehlende tiefere Jira-GitHub-Integration ist ein Schwachpunkt. Automatisches Ticket-Schließen bei gemergten PRs oder "tote Link"-Erkennung wären wünschenswerte Verbesserungen.

4.3 Vergleich mit Branchenstandards

Der Abgleich mit den Branchenstandards aus Abschnitt 2.3 ermöglicht weitere Einordnung und zeigt Entwicklungsmöglichkeiten.
Das Branching-Modell kombiniert Elemente von Gitflow und Trunk-Based Develop-

ment. Die Trennung zwischen `develop` und Release-Tags ähnelt Gitflow und ermöglicht stabiles Release-Management. Die zeitnahe Feature-Branch-Integration in `develop` vermeidet langlaufende Zweige, was Trunk-Based Development entspricht [10]. Dieser hybride Ansatz funktioniert gut, könnte aber weniger flexibel für extrem schnelle Release-Zyklen sein.
Die "Conventional Commits"-Prinzipien [11] werden nicht verbindlich umgesetzt. Entwickler bemühen sich um aussagekräftige Nachrichten, eine einheitliche, maschinell auswertbare Struktur fehlt jedoch. Die Einführung dieses Standards würde die Historie verbessern bei geringem Umstellungsaufwand.
Der Ein-Personen-Code-Review entspricht grundlegenden Best Practices [12]. Branchenstandards empfehlen oft detailliertere Review-Richtlinien und für kritische Änderungen ein Vier-Augen-Prinzip zur Effektivitätssteigerung und Betriebsblindheitsreduzierung.
Die Verwendung von `vx.x.x`-Tags und bewusste Versionsnummern-Semantik zeigt klare Orientierung am Semantic Versioning [13], was transparente Änderungskommunikation ermöglicht.
Die CI/CD-Integration wird als gut und Best-Practice-konform eingeschätzt. Verbesserungspotenzial wie Test-Workflow-Parallelisierung wurde intern erkannt und entspricht dem Optimierungsgedanken der Branchenstandards.
Grenzebach BSH nutzt GitHub-Kernfunktionen gut aus. Ungenutztes Potenzial besteht bei erweiterten Features: GitHub Projects für integriertere Projektplanung, detaillierte Issue-Templates oder Code-Owner-Features könnten den Workflow optimieren.

5 Optimierung

Aufbauend auf der Analyse der Dokumentations- und Versionskontrollpraxis (Kapitel 3 und 4) sowie dem Branchenstandardvergleich werden konkrete Optimierungsvorschläge und Handlungsempfehlungen vorgestellt. Ziel ist die Adressierung identifizierter Schwachstellen und Steigerung von Effizienz, Qualität und Konsistenz.

5.1 Handlungsempfehlungen zur Dokumentation

Die Untersuchung zeigt trotz vorhandener Stärken diverse Optimierungspotenziale. Die nachfolgenden Empfehlungen zielen auf deren Nutzung:

5.1.1 Verbesserung der Konsistenz und Verknüpfung zwischen Confluence und GitHub

Die Konsistenzgewährleistung zwischen Confluence und GitHub ist durch fehlende automatisierte Verknüpfung herausfordernd. Folgende Maßnahmen werden empfohlen:

- **Link-Management-Tool einführen:** Ein "Broken-Link-Checker" könnte fehlerhafte Verweise frühzeitig identifizieren und die Verlinkungszuverlässigkeit erhöhen.

- **Plugin-Integration verbessern:** Confluence-Plugins wie "GitHub links for Confluence" [14] sollten für direktere, dynamischere Verknüpfungen evaluiert werden.
- **"Single Source of Truth" definieren:** Für spezifische Informationstypen sollte die führende Quelle verbindlich festgelegt werden. API-Dokumentationen könnten ausschließlich aus Code generiert werden, während Prozessbeschreibungen in Confluence verbleiben.

5.1.2 Stärkung des "Documentation as Code"-Ansatzes

Der etablierte Ansatz der automatisierten technischen Dokumentationsgenerierung ist ein Kernbestandteil der "Documentation as Code"-Strategie [9]. Zur weiteren Optimierung:

- **Generierungs-Tooling optimieren:** Periodische Evaluation, ob `pdoc` alle Anforderungen erfüllt oder alternative Werkzeuge Vorteile bieten.
- **Workflows modularisieren:** GitHub-Actions-Workflows sollten flexibler gestaltet werden für vereinfachte Projektanpassung und erleichtertes Debugging.

Da die Dokumentationspflege primär in Code-Kommentaren erfolgt, würde konsequente Kommentarqualitätsverbesserung (siehe 5.1.3) Akzeptanz und Nutzen erhöhen.

5.1.3 Steigerung der Motivation und Qualität bei der Dokumentationserstellung

Die variierende Kommentarqualität und unterschiedliches Engagement deuten auf Motivations- und Verantwortlichkeitsherausforderungen hin:

- **Klare Verantwortlichkeiten:** Definition von Zuständigkeiten für Confluence-Seiten oder Moduldokumentationen schafft Verbindlichkeit.
- **Sprint-Integration:** Dokumentation sollte feste Aufgabe in Jira-Tickets werden und Teil der "Definition of Done" sein.
- **Code-Kommentar-Richtlinien:** Praxisnahe Richtlinien können Konsistenz steigern, da Grundkompetenz vorhanden ist.
- **Effektivere Scrum-Thematisierung:** Ein fester Agenda-Punkt (z.B. "Doku-Mittwoch") analog zum "Test-Montag" könnte etabliert werden.

5.1.4 Verbesserung der Auffindbarkeit und Struktur von Dokumentation

Gegen erschwerte Informationssuche bei wachsendem Dokumentationsumfang:

- **Einheitliche Nomenklatur:** Vereinheitlichung von Fachbegriffen über alle Plattformen zur Missverständnisvermeidung.

- **Strukturierungsprinzipien implementieren:** Einheitliche Prinzipien für Confluence und HTML-Dokumentation mit konsequenter Indexierung.

5.1.5 Systematische Erstellung von Architektur- und Testdokumentation

Zur systematischen Erstellung und Pflege wird die Einführung fester Vorlagen analog zu Code-Templates empfohlen. Diese sollten wesentliche Aspekte vorgeben und als Standard etabliert werden.

5.2 Optimierungsansätze für die Versionskontrolle

Trotz solider Grundlage existieren konkrete Ansatzpunkte zur Steigerung von Effizienz, Stabilität und Nutzerfreundlichkeit.

5.2.1 Verbesserung der CI/CD-Pipeline-Effizienz und -Wartbarkeit

Die CI/CD-Prozesse bieten Optimierungspotenzial. Für erleichtertes Debugging empfiehlt sich detaillierteres Logging und Pre-Commit-Hooks für lokale Vorabprüfungen. Modularere GitHub-Actions-Workflows würden Projektanpassbarkeit erhöhen und Wartungsabhängigkeiten reduzieren. Zur Durchlaufzeitenverkürzung sollte zusätzliche Code-Runner für stärkere Parallelisierung evaluiert werden. Caching-Mechanismen oder selektive Testausführung könnten zusätzlich zur Geschwindigkeitssteigerung beitragen.

5.2.2 Förderung der Akzeptanz und Effektivität von Code-Qualitätsstandards

Bei als streng empfundenen Regeln sollte der Fokus auf transparenter Vorteilskommunikation liegen, verbunden mit periodischen Team-Überprüfungen. Die verbindliche Einführung von “Conventional Commits” [11] wird empfohlen. Der Schulungsaufwand wäre gering bei IDE-Plugin-Unterstützung, der Nutzen läge in verbesserter Nachvollziehbarkeit und professionellerer Projekthistorie.

5.2.3 Stärkung des Code-Review-Prozesses

Der aktuelle Review-Prozess kann zur Qualitätssteigerung ausgebaut werden. Review-Checklisten könnten Konsistenz und Gründlichkeit erhöhen. Diese sollten Kernaspekte wie Fehlerfreiheit, Standardkonformität, Lesbarkeit und Testabdeckung umfassen. Für kritische Änderungen sollte das situative Vier-Augen-Prinzip erwogen werden.

5.2.4 Optimierung der Werkzeugnutzung und Integrationen

Es sollte geprüft werden, inwieweit GitHub Apps oder Actions eine stärkere Jira-GitHub-Automatisierung realisieren können: automatische Verlinkung von Commits/PRs in

Tickets und Status-Updates. Die Nutzung des GitHub Wikis pro Repository als technische Dokumentationsplattform sollte evaluiert werden, um Dokumentation direkter am Quellcode zu bündeln.

5.3 Integrierte Implementierungsstrategie

Die erfolgreiche Umsetzung erfordert eine durchdachte Strategie mit Priorisierung, Vorgehensmodell, Verantwortlichkeiten und Erfolgsmessung.
Für die Priorisierung werden "Quick Wins" mit hohem Nutzen bei geringem Aufwand empfohlen: verbindliche "Conventional Commits", einheitliche Nomenklatur und stärkere Dokumentationsthematisierung in Scrum-Meetings. Mittelfristig sollten aufwändigere Verbesserungen wie GitHub-Actions-Optimierung und verbesserte Confluence-GitHub-Konsistenz angegangen werden. Langfristige Maßnahmen wie weitere CodeRunner oder tiefere Atlassian-Integration könnten nach erfolgreichen ersten Schritten evaluiert werden.
Ein schrittweises Vorgehen mit Pilotprojekten für 1-2 Module minimiert Risiken und ermöglicht Erfahrungssammlung vor vollständigem Rollout. Kontinuierliche Team-Einbindung durch regelmäßige Updates und Feedback-Möglichkeiten ist essenziell für Akzeptanz.
Die Planungsverantwortung sollte beim gesamten Team liegen für breites Commitment.
Die Umsetzung einzelner Maßnahmen kann an kompetente oder interessierte Teammitglieder delegiert werden. Interne Workshops und klare Anleitungen sind ausreichend; externe Schulungen scheinen nicht erforderlich.
Zur Erfolgsmessung könnten regelmäßige Team-Umfragen zur Zufriedenheit durchgeführt werden. Quantitativ ließe sich der Erfolg durch Analyse von Workflow-Fehlern bewerten, die auf mangelhafte Dokumentation oder unklare Commits zurückzuführen sind.
Die Umsetzungsplanung kann durch Jira unterstützt werden. Quick Wins sollten innerhalb weniger Wochen umgesetzt werden, mittelfristige Ziele in ein bis zwei Monaten, langfristige Ziele in drei bis sechs Monaten.

6 Fazit und Ausblick

Diese Hausarbeit analysierte die Dokumentations- und Versionskontrollprozesse bei Grenzebach BSH und entwickelte konkrete Optimierungspotenziale sowie Handlungsempfehlungen. Dieses Kapitel fasst die Kernerkenntnisse zusammen, gibt einen Zukunftsausblick und beinhaltet ein persönliches Resümee.

6.1 Zusammenfassung der Kernerkenntnisse

Die Dokumentationspraxis-Untersuchung (Kapitel 3) zeigt eine solide Basis mit erheblichem Optimierungspotenzial. Positiv ist der Ansatz der teilweise automatisierten technischen Dokumentationsgenerierung aus Quellcode mit Integration von Testergebnis-

sen und Sicherheitschecks. Als zentrale Schwachstelle wurde mangelnde Standardisierung bei Erstellung und Pflege identifiziert. Dies zeigt sich in variierender Kommentarqualität und der Notwendigkeit einheitlicherer Vorlagen für Confluence.
Die Versionskontrollevaluation (Kapitel 4) ergab ein sehr solides Bild. Besonders positiv sind etablierte, automatisierte CI/CD-Workflows und die Git-Template-Nutzung für standardisierten Projektstart. Als wesentliche Schwachstelle wurde die fehlende Commit-Nachrichten-Standardisierung identifiziert, was im Kontrast zur erfolgten Branch-Namen-Standardisierung steht.
Die übergeordnete Erkenntnis: Bei Grenzebach BSH existieren gute informationstechnische und prozessuale Grundlagen, die Potenziale könnten jedoch durch stärkere Standardisierung und Maßnahmen zur Motivationssteigerung für Dokumentationsaufgaben besser ausgeschöpft werden.
Zur Schwachstellenadressierung wurden in Kapitel 5 konkrete Empfehlungen entwickelt. Für die Dokumentation: einheitliche Confluence-Vorlagen zur Konsistenzerhöhung und Code-Kommentar-Standardisierung zur Verbesserung von Lesbarkeit und Nachvollziehbarkeit. Für die Versionskontrolle: Commit-Nachrichten-Standardisierung zur Nachvollziehbarkeitssteigerung und CI/CD-Workflow-Weiterentwicklung zur Qualitätssicherungsverbesserung. Diese Maßnahmen zielen auf Effizienzsteigerung, Qualitätssicherung und Prozessoptimierung gemäß der Einleitungszielsetzung.

6.2 Zukunftsausblick und persönliches Resümee

Über die vorgeschlagenen Optimierungsmaßnahmen hinaus ergeben sich weitere Perspektiven für Grenzebach BSH und persönliche Weiterentwicklung.
Nach Umsetzung priorisierter Empfehlungen könnte die Anschaffung zusätzlicher CodeRunner für CI/CD-Pipelines sinnvoll sein. Dies würde stärkere Test-Parallelisierung ermöglichen und die Gesamteffizienz steigern. Eine tiefere Atlassian-Toolchain-Integration bedarf sorgfältiger Team-Evaluation. Technologische Trends wie KI-gestützte Dokumentationswerkzeuge sollten beobachtet werden. Sie könnten Effizienz und Konsistenz erheblich verbessern, bergen aber Risiken bei unsachgemäßer Anwendung. Die bestehenden DevOps-Praktiken könnten durch kontinuierlichen Wissensaufbau und neue Werkzeuge stetig optimiert werden. Persönlich war die tiefgehende Auseinandersetzung mit Dokumentationspraktiken besonders erkenntnisreich. Die unterschiedliche Konsistenz zwischen Entwicklern war mir bewusst, aber erst die Analyse verdeutlichte das volle Ausmaß und deren Einfluss auf die Gesamtqualität. Daraus resultierte die Erkenntnis über die Notwendigkeit, Standards nicht nur zu definieren, sondern deren Umsetzung zu fördern. Bei der Versionskontrolle bestätigte die Analyse die Notwendigkeit der CI/CD-Workflow-Überarbeitung. Die größte Herausforderung bestand darin, komplexe, teils informelle Praktiken vollständig zu erfassen und strukturiert darzustellen.
Die gewonnenen Erkenntnisse schärften meine Sichtweise auf die Bedeutung von Dokumentation und Versionskontrolle. Für meine zukünftige Entwicklertätigkeit werde

ich verstärkt auf Standardeinhaltung, Konsistenzförderung und die Betrachtung von Dokumentation als integralen, wertvollen Prozessbestandteil achten, statt sie als nachgelagerte Pflichtübung zu verstehen.

7 Verzeichnis der verwendeten Werkzeuge

[Claude] **Claude** (3.7 Sonnet, Anthropic)
Verwendung:

- Beihilfe für die Gliederung und Strukturierung der Hausarbeit
- Überarbeitung von Formulierungen in: 1, 2, 4

[Claude] **Claude** (4 Opus, Anthropic)
Verwendung:

- Rechtschreibprüfung Gesamtdokument
- Konsistenzprüfung Gesamtdokument
- Quellenprüfung Gesamtdokument
- Korrekturempfehlungen Gesamtdokument

[Gemini] **Gemini** (Gemini 2.5 Pro, Google)
Verwendung:

- Beihilfe der Strukturierung von 1, 5
- Rechtschreibprüfung in: 1, 2, 3, 4, 5, 6
- Literaturempfehlungen

8 Literatur

[1] WebMakers. The importance of documentation in software projects. how to avoid chaos? https://webmakers.expert/en/blog/the-importance-of-documentation-in-software-projects, November 2024. Zugriff am 19.05.2025.

[2] Eddy Groen, Matthias Koch, and Phil Stüpfert. Anforderungsdokumentation (3/5): Typische herausforderungen bei der dokumentation agiler anforderungen. https://www.iese.fraunhofer.de/blog/anforderungsdokumentation-3-herausforderungen-bei-dokumentation-agiler-anforderungen/, Januar 2020. Zugriff am 19.05.2025. Veröffentlicht im Blog des Fraunhofer IESE.

[3] MoldStud Research Team. The importance of version control systems in software development. https://moldstud.com/articles/p-the-importance-of-version-control-systems-in-software-development, Februar 2024. Zugriff am 19.05.2025.

[4] Grenzebach BSH GmbH. Unternehmen | grenzebach. https://www.grenzebach.com/de/unternehmen/, April 2025. Zugriff am 21.05.2025.

[5] Styrz - Technische Redaktion. Wo die normen en 82079 und 26514 hand in hand gehen. https://www.styrz.de/wo-die-normen-en-82079-und-26514-hand-in-hand-gehen/, September 2022. Zugriff am 21.05.2025.

[6] Scott Chacon and Ben Straub. *Pro Git*. Apress, second edition edition, 2014. https://git-scm.com/book/de/v2.

[7] Beck, Kent and Beedle, Mike and van Bennekum, Arie et al. Manifest für agile softwareentwicklung. https://agilemanifesto.org/iso/de/manifesto.html, 2001. Zugriff am 21.05.2025.

[8] Eddy Groen, Matthias Koch, and Phil Stüpfert. Anforderungsdokumentation (2/5): Der mythos „keine dokumentation“ für agile anforderungen. https://www.iese.fraunhofer.de/blog/anforderungsmanagement-2-der-mythos-keine-dokumentation-fuer-agile-anforderungen/, Januar 2020. Zugriff am 21.05.2025. Veröffentlicht im Blog des Fraunhofer IESE.

[9] Write the Docs. What is docs as code? https://www.writethedocs.org/guide/docs-as-code/, 2024. Zugriff am 21.05.2025.

[10] Atlassian. Git workflows im vergleich. https://www.atlassian.com/de/git/tutorials/comparing-workflows, 2024. Zugriff am 21.05.2025.

[11] Conventional Commits Contributors. Conventional commits 1.0.0. https://www.conventionalcommits.org/de/v1.0.0/, 2023. Zugriff am 21.05.2025.

[12] Atlassian. Was ist ein pull-request? https://www.atlassian.com/de/git/tutorials/making-a-pull-request, 2024. Zugriff am 24.05.2025.

[13] Tom Preston-Werner and contributors. Semantic versioning 2.0.0. https://semver.org/lang/de/, 2013. Zugriff am 24.05.2025.

[14] Atlassian Marketplace. Github links for confluence. https://marketplace.atlassian.com/apps/1216106/github-links-for-confluence, 2025. Zugriff am 11.06.2025.

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die Zitate deutlich kenntlich gemacht zu haben.
Ich erkläre weiterhin, dass die vorliegende Arbeit in gleicher oder ähnlicher Form noch nicht im Rahmen eines anderen Prüfungsverfahrens eingereicht wurde.

Bad Hersfeld, den 11. Juni 2025 Luca Michael Schmidt