Hochschule für Angewandte Wissenschaften Hamburg

Fakultät Life Science

Bachelorarbeit zur Erlangung des akademischen Grades B.Sc.

Studiengang Medizintechnik/Medical Engineering

CI/CD-gestützte Erstellung und Freigabe LaTeX-basierter QMS-Dokumente mit GitLab

vorgelegt von

Melissa Wilde

[redacted]

Hamburg, 26. März 2026

Gutachter: Prof. Dr. Markus, Riemenschneider (HAW Hamburg)

Gutachter: Dipl. Ing. Jürgen, Sauerzapf (Firma CardioSecur)

Die Abschlussarbeit wurde betreut und erstellt in Zusammenarbeit mit der Firma Personal MedSystems GmbH CardioSecur.

Vorwort

Der Bereich Qualitätsmanagement weckt mein Interesse, da ich es für sinnvoll und notwendig erachte, so effizient wie möglich zu arbeiten und stetig nach Verbesserung der Qualität zu streben. Einen neuen Workflow und damit einen effizienteren Weg bei der Überarbeitung von Dateien und deren Versionsverwaltung zu erarbeiten, ist für mich ein reizvolles Themengebiet, welches CardioSecur mir angeboten hat.

Aufgrund des vorgegebenen Umfangs dieser Arbeit konnten nicht alle Funktionen, die noch sinnvoll wären zu integrieren, betrachtet werden. Darunter fällt z.B. die voreingestellte Festlegung von Reviewern für bestimmte Dateien oder Dateigruppen, die Approver-Daten von GitLab in den Inhalt eines genehmigten LaTeX-Dokuments einzufügen, eine automatische Synchronisation der generierten PDF-Dokumente mit dem gewählten Ablageort und die Übersetzung von den generierten PDF-Dokumenten in andere Sprachen. Diese Automatisierungen würden den Mitarbeitern die Arbeit erleichtern und manuelle Arbeiten einsparen.

Ich danke meinen Betreuern, Herrn Markus Riemenschneider und Herrn Jürgen Sauerzapf, für die Themenbereitstellung und fachliche Unterstützung.

Ich danke CardioSecur für die Möglichkeit, meine Bachelorarbeit in dem Unternehmen schreiben zu dürfen und für die zur Verfügung gestellten Ressourcen.

Aus Gründen der besseren Lesbarkeit wird im Text das generische Maskulinum verwendet. Gemeint sind damit alle Geschlechter, einschließlich weiblicher und diverser Identitäten. Die Formulierung hat keine Wertung und dient der sprachlichen Inklusion.

Oststeinbek, 19.03.2026

Melissa Wilde

Inhaltsverzeichnis

Anhangsverzeichnis

1 Abkürzungsverzeichnis

Abkürzung	Bedeutung
QMS	Qualitätsmanagementsystem
QM	Qualitätsmanagement
MR	Merge request
Repo	Repository
VM	Virtual Machine
z.T.	zum Teil
bzw.	beziehungsweise
z.B.	zum Beispiel
i.d.R	in der Regel
u.a.	unter anderem
evtl.	eventuell
ff.	folgende
M	Mitarbeiter
R	Reviewer

Tabelle 1: Abkürzungsverzeichnis

2 Einleitung

Das Unternehmen CardioSecur möchte seinen Workflow hinsichtlich der Versionsverwaltung von gelenkten Dokumenten verbessern, wofür GitLab integriert werden soll. Es handelt sich hierbei um eine webbasierte Versionsverwaltungs-Plattform. Dabei ist wichtig, dass die Anforderungen von der ISO 13485 (QMS für Medizinprodukte) bezogen auf gelenkte Dokumente und Computersystemvalidierung erfüllt werden. Die Rückverfolgbarkeit der einzelnen Versionen soll mit Hilfe von GitLab einfacher werden und das Vorliegen aller Versionen eines Dokuments auf eine einzige Version des gewählten Ablageorts (hier: Netzlaufwerk) reduzieren und dadurch die Übersichtlichkeit und Aktualität der Dokumente verbessern sowie Fehler reduzieren (z.B. veraltete Versionen zu verschicken). Außerdem soll ausgeschlossen werden, dass verschiedene Mitarbeiter auf ihrem lokalen Endgerät gleichnamige Versionen mit unterschiedlichem Inhalt desselben Dokuments vorliegen haben, welche aufgrund mangelnder Übersichtlichkeit nicht zusammengeführt werden.

Um GitLab effektiv nutzen zu können, ist allerdings auch die Verwendung von textbasierten Dokumenten (hier: LaTeX-Dokumente) notwendig, da GitLab bzw. ein Git-Client bei diesem Format Änderungen an demselben Dokument erkennen kann.

Durch CI/CD-Jobs und zusätzliche Funktionen sollen Automatisierungen integriert werden. Darunter z.B. die Erstellung eines PDF-Dokuments basierend auf einem LaTeX-Dokument.

Ein Teil dieser Arbeit besteht aus der Fragestellung, ob die Verwendung von GitLab einen ISO-konformen Workflow der Dokumentenlenkung erfüllt, sodass die Verwendung von GitLab im Bereich der Medizinprodukte auditfähig ist. Die Kernthematiken sind der Workflow, die Automatisierungen und die notwendigen Einstellungen in GitLab.

Es wird vorausgesetzt, dass ein GitLab-Server vorhanden ist.

3 Anforderungen

3.1 Allgemeine Anforderungen

Aus der Idee heraus, GitLab zu verwenden, sind folgende Anforderungen entstanden:

- Anforderungen der ISO 13485 müssen erfüllt werden
- Verwendung von Git
- Verwendung von GitLab
- Verwendung eines LaTeX-Editors: LaTeX-Dokumente zum Testen und Validieren
- Automatisierung zur Generierung eines PDF-Dokuments aus einem LaTeX-Dokument
- Bereitstellung des PDF-Dokuments an einem für jeden zugänglichen Ablageort
- Rollenmodell für alle Mitarbeiter
- Je nach zugewiesener Rolle und Aufgabe sollen automatisch E-Mail-Benachrichtigungen verschickt werden
- Benutzerfreundlicher Workflow
- Einstellungen in GitLab festlegen

3.2 Anforderungen der ISO 13485

Nachfolgend sind die zentralen Anforderungen an gelenkte Dokumente laut ISO 13485 (insbesondere Kapitel 4.2) aufgelistet [1, 2]**:**

- Dokumentenlenkung
  - Dokumente müssen genehmigt werden, bevor sie verwendet werden
  - Änderungen müssen geprüft, genehmigt und nachvollziehbar sein
- Identifikation
  - Jedes Dokument muss eindeutig identifizierbar sein Dateiname
- Verfügbarkeit
  - Gültige Dokumente müssen dort verfügbar sein, wo sie benötigt werden (z.B. Arbeitsplätze, IT-Systeme)
  - Ungültige oder veraltete Dokumente dürfen nicht unbeabsichtigt verwendet werden
- Änderungskontrolle
  - Änderungen müssen dokumentiert sein (was, wann, warum, von wem)
- Lesbarkeit und Verständlichkeit
  - Dokumente müssen lesbar sein
- Archivierung und Aufbewahrung
  - Alte Versionen müssen kontrolliert aufbewahrt oder eindeutig als ungültig gekennzeichnet werden
  - Aufbewahrungsfristen müssen eingehalten werden

4 Umsetzung

In diesem Kapitel wird beschrieben und erklärt, was für die Umsetzung der Anforderungen benötigt wird. Dabei wird z.T. auf Anleitungen im Anhang für die praktische Umsetzung verwiesen.

4.1 Verwendete Software

4.1.1 Git

Git sollte auf jedem Gerät lokal installiert sein, damit zur Not von der Kommandozeile aus auf den Git-Server zugegriffen werden kann und Aktionen wie Commit, Push oder das Hinzufügen von Dateien auch ohne direkten Zugang zu GitLab ausgeführt werden können. Der offiziellen *GitLab Homepage* kann die Installation von Git auf den unterschiedlichen Betriebssystemen macOS [3], Windows [4] und Linux [5] entnommen werden.

Da sich der Git-Server in diesem Fall im Unternehmensnetzwerk, im Intranet, befindet und der Zugang nur mit einer VPN-Verbindung möglich ist, kann kein Außenstehender den Server erreichen [6]. Es kann lediglich der Router bzw. die Firewall von außen erreicht werden, es sei denn es wurden aktiv andere Einstellungen vorgenommen. Der Zugriff auf den Git-Server und die dort enthaltenen Daten durch unbefugte Personen ist damit ausgeschlossen.

4.1.2 LaTeX-Editor (hier: Texmaker Version 6.0.1)

Texmaker ist ein kostenloser LaTeX-Editor, mit dem LaTeX-Dokumente geöffnet, erstellt und bearbeitet werden können. Zudem beinhaltet er eine integrierte PDF-Vorschau des LaTeX-Dokuments.

LaTeX-Dokumente sind textbasiert, was notwendig ist, damit GitLab verschiedene Versionen vergleichen bzw. Differenzen an demselben Dokument erkennen und anzeigen kann. Es wird dabei der gesamte Inhalt eines Dokuments Zeile für Zeile mit dem Inhalt der neuen Version dieses Dokuments verglichen. Dies ist mit z.B. Word-Dokumenten nicht möglich.

In diesem Fall wird ein LaTeX-Editor verwendet, um den Workflow mit Hilfe von Test-Dokumenten validieren zu können.

4.1.3 Git-Client (hier: Sourcetree Version 3.4.30)

Sourcetree ist ein Git-Client mit einer grafischen Benutzeroberfläche, auf der per Klick verschiedene Funktionen ausgeführt werden können, die sonst entweder in GitLab selbst oder in der Konsole (Windows) bzw. Terminal (macOS) per Kommandos ausgeführt werden. Die Nutzung eines Clients ist für Mitarbeiter, die keine Erfahrung mit der Verwendung von Kommandozeilen haben, ein benutzerfreundlicherer und effizienterer Weg zu arbeiten.

Hier wird mit Sourcetree gearbeitet, da dieser Client kostenlos nutzbar ist und alle grundsätzlichen Funktionen beinhaltet, die für die Umsetzung dieser Arbeit nötig sind. Es gibt aber noch diverse andere Clients, die mehr oder weniger Funktionen beinhalten und entweder ebenso kostenlos nutzbar oder kostenpflichtig sind. Es sollte bei der Auswahl des Clients jedes Unternehmen für sich herausfinden und entscheiden, welche Funktionen benötigt werden und welcher Client basierend auf dieser Grundlage die Anforderungen des Unternehmens am besten erfüllen kann. Eine Auswahl an Git-Clients ist in den *GitLab Docs* aufgeführt [7].

Verwendet werden hier folgende Funktionen:

- Repository klonen
- Branch erstellen
- Commit erstellen
- Push zu GitLab
- „Explorer“ öffnet den Pfad des lokalen Repos
- „Remote“ öffnet das Repo in GitLab (Browser)

4.2 GitLab

GitLab basiert auf Git und ist eine webbasierte Versionsverwaltungs-Plattform, welches im Gegensatz zu Git eine Benutzeroberfläche bietet. Es dient der Verwaltung und Zusammenarbeit von Dateien sowie dem allgemeinen Projektmanagement. Außerdem bietet es zahlreiche Funktionen wie die Strukturierung von Projekten, Änderungen an Dateien, deren Freigabeprozess und die Integration von automatischen Abläufen. Zudem werden automatisch Bearbeitungs-Historien jeder Datei in Git angelegt, welche zu jeder Zeit in GitLab einsehbar, vergleichbar und wiederherstellbar sind [8]. So wird die Rückverfolgbarkeit jeder Datei und aller Versionen von ihr gewährleistet. Bei textbasierten Dokumenten kann GitLab die Änderungen vergleichen und Differenzen zur vorigen Version erkennen. Dafür geeignet sind unter anderem Quellcode und LaTeX-Dokumente. [9]

4.2.1 Repository (Repo)

Die Repos sind Teil einzelner Projekte und bilden die Grundlage für die Strukturierung in GitLab. Sie können mit einer Ordnerstruktur auf einem Computer verglichen werden. Repos beinhalten die LaTeX-Dokumente und andere in den Dokumenten verwendete Dateien. [10]

Es wäre sinnvoll, wenn die Administratoren in GitLab die Repos erstellen, da diese alle Berechtigungen auf dem Server besitzen und eher die Erfahrung mitbringen eventuell auftretende Probleme zu lösen. Theoretisch kann aber jeder ein Repo erstellen, der eine dafür berechtigte Rolle in GitLab besitzt. Derjenige, der das Repo erstellt, ist automatisch der Eigentümer von diesem und hat andere Berechtigungen. Er kann z.B. Personen oder Personengruppen Rollen mit den gewünschten Berechtigungen zuweisen. Der Zugang kann zeitlich begrenzt werden.

4.2.2 Branches

Main branch (i.d.R. der *target branch*)

Der *Main branch* wird automatisch beim Erstellen eines Repos angelegt und enthält die ursprünglichen Dokumente mit den darin verwendeten Dateien, welche zum Bearbeiten verwendet werden. Hier wird die jeweils gültige Version einer Datei zur Verfügung gestellt, wenn eine Person eine Änderung vorgenommen hat und diese freigegeben (*gemerged*) wurde. Die daraus entstehende aktuelle Datei bildet die neue Grundlage für das weitere Bearbeiten dieser Datei. Bezogen auf die in dem Rahmen dieser Arbeit vorgenommenen Einstellungen der Historien-Erstellung im *Main branch* bilden die Historie die *Merge commits* (s. Kapitel 4.3.3). Der *Main branch* besteht folglich aus einer Reihe von *Merge commits*.

Normalerweise ist der *Main branch* ein geschützter Branch (*protected branch*), in dem keine direkten Änderungen vorgenommen werden sollen und können [11]. Damit wird sichergestellt, dass in dem *Main branch* nur überprüfte und freigegebene Versionen der Dateien vorhanden sind. Mit den richtigen Einstellungen des *protected branches* können Benutzer mit der Rolle *Developer* und *Maintainer* (s. Kapitel 4.3.1) eine MR (*Merge request;* s. Kapitel 4.2.3) *mergen*, aber ein direkter *Push* in diesen *Branch* ist von niemandem möglich. Diese Einstellung ist wichtig für die Einhaltung der Anforderung aus der ISO 13485, welche besagt, dass Änderungen immer geprüft und genehmigt werden müssen, bevor sie freigegeben werden. Die Anleitung für die Einstellungen eines *protected branches* eines Projekts befinden sich in Anhang I. Administratoren können im Admin-Modus für alle Projekte die *Branch-Regeln* für *protected branches* voreinstellen [12]. Diese können dann von den Eigentümern der einzelnen Projekte bei Bedarf angepasst werden.

Branch (*source branch*)

Ein *Branch*, auch als *feature branch* bezeichnet, ist in erster Linie ein Abbild eines *Main branches*, der von einer Person erstellt wird und worin die ursprünglichen Dateien bearbeitet werden können, ohne dass die Dateien direkt im *Main branch* verändert werden. Dies ist zwingend notwendig, da in dem *Main branch* keine direkten Änderungen vorgenommen werden sollen und können. In einem *Branch* werden ein oder mehrere *Commits* erstellt, die nach einem *Merge* (Freigabe; s. Kapitel 4.2.3) ein *Merge commit* im *Main branch* bilden. Es ist möglich, dass mehrere Personen jeweils einen neuen *Branch* desselben *Main branches* erstellen, ihre individuellen Änderungen vornehmen und diese dann nach und nach in den *Main branch* per *Merge* übernommen werden. Zudem können Änderungen, die Probleme bereiten, rückgängig gemacht werden, ohne die Arbeit anderer Personen zu beeinträchtigen. Die einzelnen *Branches* werden i.d.R. nach erfolgreichem *Merge* gelöscht, da diese dann nicht mehr benötigt werden und dadurch die Übersichtlichkeit der aktuellen *Branches* bewahrt wird. [13, 14] In der folgenden Abb. 1 ist der übliche Weg mit *Branches* zu arbeiten dargestellt.

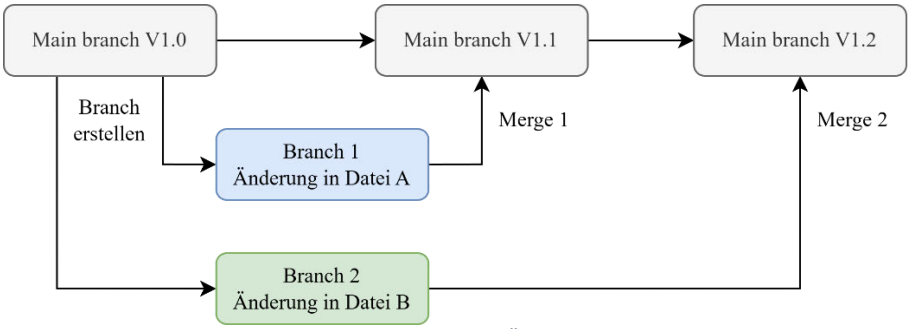

Abb. 1: Arbeit mit verschiedenen Branches mit Überarbeitung verschiedener Dateien

4.2.3 Commit, Push, Merge request (MR) und Merge

Mit einem *Commit* in einem Git-Client werden die lokal vorgenommenen Änderungen von Dateien eines Repos gespeichert und eine neue lokale Version des Repos angelegt. Dabei können Dateien innerhalb des Repos geändert, hinzugefügt oder entfernt werden. Sie bilden die Bearbeitungs-Historie (*Commit-Historie*) einer Datei mit dem Inhalt, Autor und Zeitpunkt der Änderung sowie eine hinzuzufügende *Commit-Beschreibung* der vorgenommenen Änderungen [15]. Von GitLab wird eine eindeutige Kennung für jeden einzelnen *Commit* (*commit SHA*) generiert [15]. *Commits* sind somit die Grundlage für neue Versionen eines Repos bzw. dessen Dateien. Dabei kann zu jedem Zeitpunkt jeder beliebige *Commit* angesehen werden, was den Grundbaustein der Dokumentation für die Rückverfolgbarkeit der Versionsverwaltung legt. *Commits* sollten regelmäßig erfolgen, wenn die Bearbeitung eines Dokuments mehr Zeit in Anspruch nimmt (oder auch über mehrere Tage hinweg bearbeitet wird) und/oder viele Änderungen beinhaltet.

Mit einem *Push* werden die lokal vorgenommenen *Commits* von Dateien eines Repos an GitLab gesendet und sind dort in dem entsprechenden *Branch* vorhanden. Dort können sie dann eingesehen und heruntergeladen werden, auch wenn sie noch nicht freigegeben sind.

Wenn die Änderungen aller Dateien in einem Repo abgeschlossen sind und diese in GitLab übernommen werden sollen, sodass eine neue gültige Version in dem *Main branch* entsteht, wird eine MR gestellt. Dafür muss zuerst ein *Push* der *Commits* zu GitLab erfolgen. Diese MR wird i.d.R. von mindestens zwei Reviewern überprüft. Nachdem alle Reviewer die Änderungen in der MR überprüft und genehmigt (*approved*) haben, erfolgt die Freigabe der Änderungen durch einen *Merge*. Dem *Main branch* wird der *Merge commit* in der Historie hinzugefügt und beinhaltet u.a. die Informationen, wer zu welchem Zeitpunkt die MR *approved* und *gemerged* hat. Der *Merge commit* ist die neue gültige Version der Datei im *Main branch* des Repos in GitLab und bildet die neue Grundlage für weitere Bearbeitungen. Der *Branch* wird gelöscht. Es ist sinnvoll die Regeln für die *Approvals* so einzustellen, dass die Ersteller der MR und Personen, die *Commits* in dem *Branch* vorgenommen haben, nicht die Berechtigung besitzen die MR zu *approven*. Somit wird sichergestellt, dass keiner seine eigenen Änderungen genehmigen kann. Eine Anleitung für die *MR approvals* kann den *GitLab Docs* entnommen werden [16].

Wenn eine MR abgelehnt wird, müssen die entsprechenden Dateien erneut bearbeitet werden und nach der Änderung eine neue MR gestellt werden. Der oben beschriebene Prozess beginnt dann von vorne.

In folgender Abb. 2 ist der Ablauf mit *Commits* eines *Branches* bis hin zum *Merge* der Änderungen dargestellt. Der „Main branch V1.0“ bildet in dem Beispiel die Grundlage des *Branches* und „Main branch V1.1“ ist die neue Version des *Main branches* nach dem *Merge*.

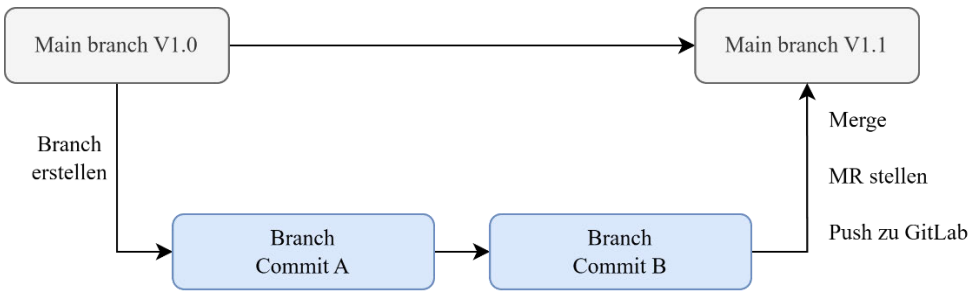

Abb. 2: Ablauf von Commits eines Branches bis zum Merge der Änderungen

4.2.4 Tag

Ein *Tag* wird manuell in GitLab erstellt oder automatisch, wenn ein Release erstellt wird. Generell handelt es sich bei *Tags* um eine endgültige neue Version eines Repos (i.d.R. aus dessen *Main branch*), die eine neue Bezeichnung mit z.B. „V2.0“ erhält. Die Bezeichnung ist nicht vorgegeben und somit frei wählbar. *Tags* sind übersichtlicher als *Merge-Commits* zu verwenden, wenn endgültige Versionen benötigt werden, die z.B. auch an Dritte gesendet werden. Es ist sinnvoll einen *Tag* zu erstellen, wenn die Änderungen einer oder mehrerer Dateien in einem Repo so bedeutend sind, dass die bisherige Version nicht mehr aktuell ist bzw. verwendet werden darf (z.B. bezogen auf aktualisierte Normen). *Tags* werden in einer separaten Historie des Projekts gespeichert und können jederzeit eingesehen und heruntergeladen werden. Dadurch sind sie leicht zu finden und kennzeichnen formale Stände mit bedeutenden Änderungen. Hier wird nach einer *Tag-Erstellung* ein PDF-Dokument aus dem ursprünglichen textbasierten Dokument erstellt, welches anschließend an einem Ort (z.B. Netzlaufwerk) abgelegt wird, zu dem jeder Zugang hat.

Es gibt die Optionen eines *lightweight tags* und eines *annotated tags*. *Lightweight tags* beziehen sich auf spezifische *Commits* und können wieder entfernt werden. *Annotated tags* hingegen enthalten Metadaten, können signiert und nicht mehr verändert oder entfernt werden. Bezogen auf die Rückverfolgbarkeit aller Dateien und deren Versionen, zu denen auch *Tags* gehören, müssen hier *annotated tags* verwendet werden. Dazu muss eine *Tag-Beschreibung* hinzugefügt werden. [17]

In Abb. 3 ist der Ablauf einer *Tag-Erstellung* grafisch dargestellt.

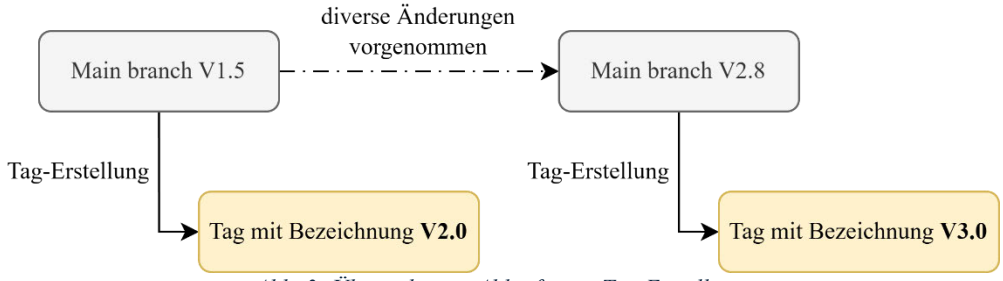

Abb. 3: Übersicht zum Ablauf einer Tag-Erstellung

4.2.5 Projekte archivieren

Projekte, die nicht mehr aktiv benötigt werden, können in GitLab von einem Administrator oder dem Eigentümer (*Owner*) des Projekts archiviert werden. Das Projekt erscheint dann nicht mehr in der Liste der Projekte (auf dem Dashboard), wird per Suchfunktion nicht mehr angezeigt und es können keine Änderungen an den Inhalten vorgenommen werden. Das gesamte Projekt wird auf *read-only* gestellt und ist mit *archived* gekennzeichnet. Die Archivierung kann jederzeit von einem Admin oder dem Eigentümer des Projekts rückgängig gemacht werden. Die Anleitungen zur Archivierung eines Projekts und zur Entfernung eines Projekts aus dem Archiv können den *GitLab Docs* entnommen werden [18].

Eine Archivierung kann z.B. verwendet werden, wenn ein Projekt Dokumente eines Produkts beinhaltet, welches vom Markt genommen wird. An diesen Dokumenten wird es keine Änderungen mehr geben. Da Projekte weiterhin laut Anforderung der ISO 13485 im Bezug zu der Aufbewahrungsfrist zur Einsicht zur Verfügung stehen müssen (s. Kapitel 3.2), ist die Archivierung ein wichtiger Bestandteil für die Einhaltung der Anforderung.

4.2.6 CI/CD-Pipeline

CI/CD steht für *Continuous Integration/Continuous Delivery* und bedeutet, dass Prozesse kontinuierlich und automatisch unter bestimmten Voraussetzungen ausgeführt werden. Die *CI/CD-Pipelines* haben die Aufgabe der Automatisierung von Prozessen, damit Arbeit und Zeit eingespart sowie Fehler reduziert werden. Dies wird durch *Runner* bzw. *Jobs* umgesetzt, die entsprechende Skripte ausführen. Es ist außerdem eine Sicherheitsvorkehrung, wenn automatisierte Prozesse nach einer gewissen Zeit Dateien überprüfen. Z.B. wird bei der Generierung eines PDF-Dokuments auf Grundlage eines LaTeX-Dokuments der entsprechende *Job* fehlerhaft bzw. gar nicht zu Ende ausgeführt, wenn in dem LaTeX-Dokument Fehler vorhanden sind, z.B. undefinierte Formatierungen oder ein falsch angegebener Pfad einer integrierten Bild-Datei, die verwendet werden. Dies stellt sicher, dass die generierten PDF-Dokumente den vollständigen richtigen Inhalt der verwendeten LaTeX-Dokumente beinhalten. [19]

Build-Jobs

Ein *Build-Job* ist ein Skript, das sich in Form einer `.gitlab-ci.yml`-Datei in einem Repo eines Projekts befindet. Das Skript wird immer ausgeführt. In dem Skript können sich mehrere einzelne auszuführende *Jobs* befinden, die jeweils unter festgelegten Bedingungen ausgeführt werden. Es kann folglich bei dem Durchlaufen des Skripts auch nur ein Teil der integrierten Jobs ausgeführt werden. Sie sind Teil der Automatisierung und greifen hier auf *Docker Container* (s. Kapitel 4.2.7) zu, damit die dort installierten Pakete für die Ausführung des Jobs verwendet werden. In diesem Fall sollen aus den LaTeX-Dokumenten PDF-Dokumente generiert werden.

Artifacts (Artefakte)

Artefakte sind aus den ausgeführten *Build-Jobs* generierte Dokumente, die für eine bestimmte Zeit (Standardeinstellung: 30 Tage) in GitLab heruntergeladen werden können. Dabei werden die in diesem Fall generierten PDF-Dokumente in einem ZIP-Archiv zum Download zur Verfügung gestellt.

4.2.7 Docker Container und Docker Registry

Container

Im Gegensatz zu einer virtuellen Maschine, die ein komplettes Betriebssystem beinhaltet und von einem Hypervisor (z.B. VirtualBox) gestartet wird, teilt sich ein *Container* den Kernel (das Herzstück des Betriebssystems) mit dem Host, arbeitet aber isoliert [20]. In einem *Container* lassen sich Tools installieren und ausführen, ohne den Host direkt zu verändern und ohne Zugriff außerhalb der freigegebenen Grenzen. Der *Container* kann zudem ein beliebiges Betriebssystem verwenden und muss nicht mit dem lokalen Betriebssystem übereinstimmen. Dadurch kann ein Betriebssystem gewählt werden, welches am besten zu den eigenen Anforderungen und Bedürfnissen passt. Als Betriebssystem für den *Container* wird Linux (ubuntu) empfohlen.

GitLab unterstützt *Docker-Container*, indem *CI/CD-Jobs* in frei wählbaren *Container-Images* [21] ausgeführt werden können, und stellt mit der *Container Registry* einen integrierten Speicher für diese *Images* bereit. Sowohl für Administratoren als auch für Benutzer bieten *Container* entscheidende Vorteile: Benutzer können innerhalb eines *Containers* genau die Tools und Abhängigkeiten installieren, die sie für *Build-Jobs* benötigen, während gleichzeitig sichergestellt

ist, dass diese in einer kontrollierten, isolierten Umgebung laufen. Ohne *Container* müssten Benutzer den Administrator um die Installation der benötigten Komponenten bitten. Dieser müsste sich zusätzlich zur Wartung auch um Themen wie Sicherheit, Versionskonflikte und mögliche Seiteneffekte mit anderen Komponenten kümmern. Zudem kann ein *Container* global angelegt werden, sodass er für alle Projekte nutzbar ist.

GitLab unterstützt auch andere *Container-Formate*, aber hier werden ausschließlich *Docker-Container* betrachtet. Im Gegensatz zu der Verwendung von *Shell* sind *Container* sicher und stellen kein Risiko im Bezug zum Datenschutz dar, weil sie isoliert arbeiten und somit nicht von außen auf sie zugegriffen werden kann. *Shell* arbeitet hingegen direkt auf dem lokalen Betriebssystem und ist dadurch deutlich risikoreicher als die Verwendung von *Containern*, da hier nicht isoliert gearbeitet wird [22].

Container Registry

Der *Docker Container* muss registriert werden, damit der Zugang zu ihm gewährleistet ist und die dort installierten Tools von *Build-Jobs* verwendet werden können. Für die Installation und Registrierung eines *Docker Containers* wird die Konsole (Microsoft) bzw. das Terminal (macOS) benötigt. Daher sollte dieser Vorgang nur durch Fachpersonal erfolgen. Nachdem dies erfolgreich abgeschlossen wurde, ist der *Docker Container* registriert und kann verwendet werden. Die Anleitungen zur Installation und Registrierung von *Docker Containern* sowie das Erstellen und Hochladen von *Docker-Images* befinden sich in Anhang II. [23]

4.3 Einstellungen in GitLab

4.3.1 Rollenmodell und Gruppen

Es gibt ein festgelegtes Rollenmodell in GitLab, welches besagt, mit welcher Rolle eine Person welche Berechtigungen hat. Die Rollen sollten so zugewiesen werden, dass die Person ihre geforderte Tätigkeit ausführen kann, auch wenn sie dafür nebenbei andere Berechtigungen erhält, die nicht genutzt werden. Die Person, die z.B. ein Review durchführen soll, bekommt folglich die Rolle des *Maintainers* für das entsprechende Repo. Ein Administrator hat die Berechtigung einzelnen Personen eine generelle Rolle zuzuweisen. Der jeweilige Eigentümer eines Projekts kann für dieses ebenfalls Rollen zuweisen. Dementsprechend kann eine Person in GitLab generell, als Mitglied in verschiedenen Projekten und auch als Mitglied in verschiedenen Gruppen jeweils unterschiedliche Rollen und Berechtigungen haben. Somit kann er in einem Projekt z.B. ein *Developer* und in einem anderen ein *Maintainer* sein. [24]

Für den Inhalt dieser Arbeit wird mindestens die Rolle des Developers benötigt, weshalb auf die anderen Rollen nicht weiter eingegangen wird. Die Rollen werden wie folgt vergeben (aufsteigende Reihenfolge der Berechtigungen):

- **Developer** sind die normalen Mitarbeiter, welche die Berechtigung besitzen Repos zu klonen und MRs zu stellen
- **Maintainer** sind die Reviewer, welche die Berechtigung besitzen MRs zu *approven* und *mergen*
- **Owner** sind die Ersteller und Eigentümer von Projekten und besitzen darin nahezu alle Berechtigungen
- **Admin** sind die Administratoren in GitLab und eventuell die Geschäftsführung, welche (je nach Einstellungen in GitLab) nahezu alle Berechtigungen besitzen

Zu den weiteren Rollen gehören (aufsteigende Reihenfolge der Berechtigungen):

- Minimal Access
- Guest
- Planner
- Reporter

Ein gängiges Vorgehen ist das Erstellen von Gruppen und diese dann zu einzelnen Projekten einzuladen. Somit wird die Arbeit gespart jede einzelne Person zu suchen und hinzuzufügen. Dies ist vor allem sinnvoll, wenn das Unternehmen sehr viele Mitarbeiter hat. Bei kleineren Unternehmen mit geringerer Mitarbeiterzahl ist dies nicht unbedingt nötig, da die Personenliste recht übersichtlich ist, sich die Mitarbeiter alle kennen und viele Mitarbeiter an den meisten Projekten mitwirken. Somit würde so gut wie jeder Mitarbeiter Zugang zu jedem Projekt bekommen. In dem Fall könnte z.B. eine Gruppe für alle Developer und eine Gruppe für alle Reviewer erstellt werden. Die Anleitungen für das Hinzufügen einzelner Personen zu einem Projekt [25], eine Gruppenerstellung [26], das Hinzufügen von Gruppenmitgliedern zu einer bestehenden Gruppe [27], und die Einladung einer Gruppe zu einem Projekt [28] können den *GitLab Docs* entnommen werden.

4.3.2 E-Mail-Benachrichtigungen

In folgenden Fällen soll eine E-Mail-Benachrichtigung automatisch von GitLab an die zuständige/n Person/en geschickt werden. Die folgende Auswahl an Optionen ist lediglich ein Vorschlag für die einzelnen Rollen und deren Funktionen. Sie können jederzeit beliebig nach Bedarf angepasst werden. Dabei müssen die einzelnen Personen in ihrem eigenen Profil die entsprechenden Einstellungen vornehmen. Die Anleitungen der einzelnen Einstellungen (Mitarbeiter, Reviewer, Geschäftsführung/Administratoren, Tag-Erstellung) befinden sich in Anhang III.

Mitarbeiter

Der Mitarbeiter, der Dateien bearbeitet und eine MR stellt, muss in erster Linie darüber informiert werden, was mit der MR passiert ist, z.B. ob die vorgenommenen Änderungen genehmigt wurden oder nicht. Gerade wenn anschließend weiterer Handlungsbedarf für eine Überarbeitung besteht, muss der zuständige Mitarbeiter darüber informiert werden, um eine Übersicht über die zu erledigenden Aufgaben zu erhalten. Auch über einen Wechsel der Reviewer einer MR und eine erneute Öffnung einer MR sollte der Mitarbeiter informiert werden. Dies sind die relevantesten Benachrichtigungen, um für die zu tätigende Arbeit auf dem aktuellen Stand zu bleiben.

Reviewer

Der zuständige Reviewer eines Repos muss darüber informiert werden, wenn eine neue MR gestellt wurde und wenn ein *Push* zu einer bestehenden MR hinzugefügt wurde. Dies stellt sicher, dass die Arbeit ohne Verzögerung erledigt werden kann. Es kann zusätzlich die Einstellung vorgenommen werden darüber informiert zu werden, wenn eine MR erneut geöffnet wurde.

Geschäftsführung/Administratoren

Die Geschäftsführung bzw. Administratoren müssen sich Gedanken darüber machen über wie viele kleine Schritte sie informiert werden möchten. Wenn die Geschäftsführung weder als Reviewer noch als Bearbeiter von Dateien tätig ist, werden die Benachrichtigungen für diese Aufgaben nicht zwingend benötigt. Es ist dann unter anderem wichtig, über zu bearbeitende Aufgaben bzw. Tickets (*Issues*), durchgeführte Merges und Releases informiert zu werden.

Tag-Erstellung

Bei einer *Tag-Erstellung* sollen alle für dieses Repo relevanten Mitarbeiter informiert werden, dass eine neue Version vorliegt und welche Änderung/en sie beinhaltet. Dafür wird die Integration „*Emails on push*“ von GitLab verwendet, bei der jede Person, die informiert werden soll, eingetragen wird [29].

4.3.3 Historie

Grundsätzlich bilden die *Commits* von einer Datei dessen Historie. Bei einem *Merge* können mehrere *Commits* zusammengefasst (*squashing*) werden, wodurch die Historie mit weniger *Commits* erstellt wird [30]. Sie können aber auch alle einzeln gelassen und aufgeführt werden, was eher den Anforderungen der ISO 13485 für eine detailliertere Rückverfolgbarkeit entspricht.

Wie bzw. wann die Historie in einem Projekt erstellt wird, kann in jedem Projekt einzeln eingestellt werden. Es empfiehlt sich, in jedem Projekt die gleichen Einstellungen vorzunehmen, damit die Historien in allen Projekten einheitlich und leichter nachzuvollziehen sind. Damit wird Verwirrung entgegengewirkt, wenn eine Person sich die Historien verschiedener Projekte ansieht.

Es gibt drei Möglichkeiten, wie eine Historie erstellt werden kann [31]. Jedes Unternehmen muss für sich entscheiden, welche Methode am besten zu dessen Ansprüchen und Arbeitsweise passt. Die Unterschiede liegen hierbei in der Linearität der Historie und der Durchführung eines Merges.

1. Merge commit

Mit jedem *Merge* eines *Branches* in den *Main branch* wird ein *Merge commit* erstellt. Dabei werden in den *Main branch* die einzelnen *Commits* als Versionen des *Main branches* geschrieben. Zusätzlich wird die letzte Version als *Merge commit* zusammengefasst und als aktuelle Version des *Main branches* gespeichert. Die einzelnen *Commits* eines *Branches* befinden sich also in dem *Main branch*. Dies kann dazu führen, dass die Historie des *Main branches* sehr lang und unübersichtlich werden kann. Außerdem ist die Historie im *Main branch* nicht linear, da *Commits* aus unterschiedlichen *Branches* zu unterschiedlichen Zeitpunkten erstellt und *gemerged* werden.

In Abb. 4 ist dies einmal grafisch dargestellt. Dabei werden die *Commits* A, B und C in dieser zeitlichen Abfolge getätigt. Dennoch wird „Commit B“ erst nach dem „Commit C“ per *Merge commit* in den *Main branch* übernommen, wodurch er zeitlich nicht linear ist.

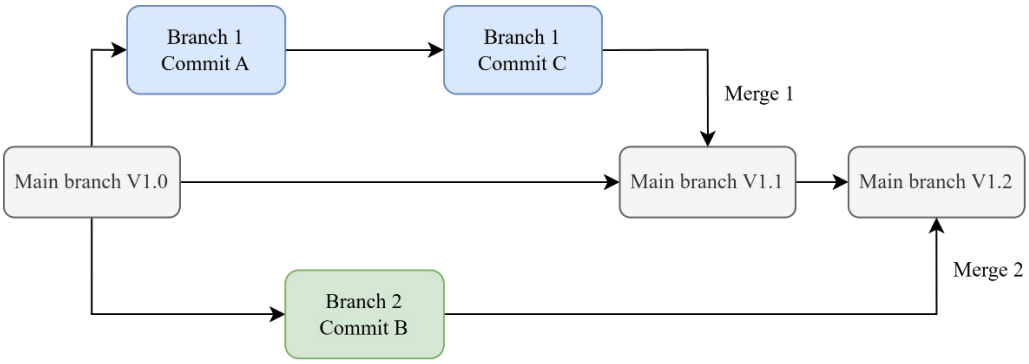

Abb. 4: Zeitliche Linearität der Methode „Merge commit“

2. Merge commit with semi-linear history

Auch hier wird mit jedem *Merge* eines *Branches* in den *Main branch* ein *Merge commit* erstellt. Dies hat allerdings die Voraussetzung, dass ein *fast-forward merge* möglich ist. Das bedeutet, dass der *Branch* sich auf die aktuelle Version des *target branches* (meistens der *Main branch*) bezieht. Sollte dies nicht der Fall sein, wird dem Benutzer die Option gegeben ein *Rebase* durchzuführen (s. Kapitel 5.4.1). In dem *Main branch* wird nur der *Merge commit* als neue Version gespeichert. Dadurch wirkt der *Main branch* in der Historie übersichtlich. Trotzdem ist noch nachvollziehbar wann welcher *Branch* erstellt und *gemerged* wurde. Die Historie ist in diesem Fall zeitlich semi-linear.

In Abb. 5 wird nach erfolgreichem *Merge* von „Branch 1“ eine neue Version des *Main branches* als *Merge commit* angelegt. „Branch 2“ kann ohne *Rebase* nicht *gemerged* werden, da er sich sonst auf den Stand einer veralteten Version des *Main branches* beziehen würde (hier: V1.0). Nach dem *Rebase* kann „Branch 2“ *gemerged* werden. Die zeitliche Abfolge der *Commits* wird dementsprechend verschoben, da „Commit B“ vor „Commit C“ erstellt wurde, aber erst als *Merge commit* nach „Commit C“ in den *Main branch* übernommen wird.

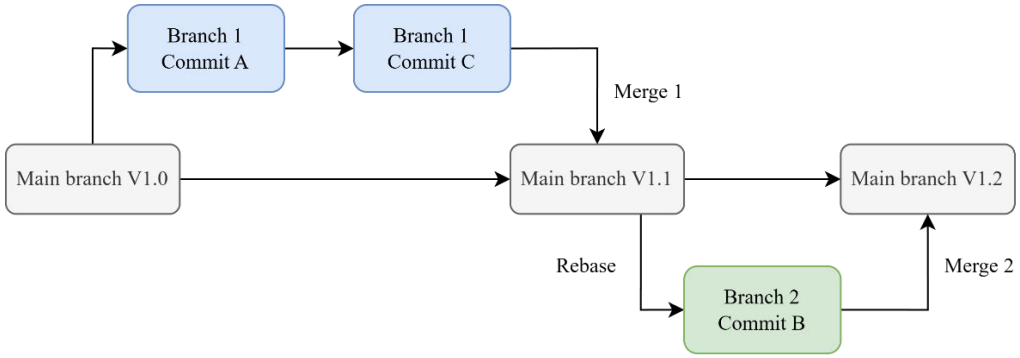

Abb. 5: Zeitliche Linearität der Methode „Merge commit with semi-linear history“

3. Fast-forward merge

Bei dieser Methode wird eine lineare Historie erstellt, bei der jeder einzelne *Commit* in den *Main branch* übernommen wird. Dafür wird bei einem *Merge* kein *Merge commit* erstellt, da der letzte *Commit* zu der neuen Version des *Main branches* wird. Ein *Merge* kann nur erfolgen, wenn ein *fast-forward merge* möglich ist, d.h. wenn sich der *Branch* auf die aktuelle Version des *Main branches* bezieht. Sollte dies nicht der Fall sein, muss ein *Rebase* durchgeführt werden (s. Kapitel

5.4.1). Der *Main branch* wird dadurch sehr voll und kann schnell überladen wirken. Die Erstellung der *Branches* sowie deren *Merge* in den *Main branch* sind danach nicht mehr sichtbar. Die Übersichtlichkeit in der Hinsicht geht damit verloren. Allerdings ist jeder einzelne Commit in dem *Main branch* gespeichert, wodurch alle *Commits* mit linearer Historie an einem einzigen Ort zu finden sind.

In Abb. 6 ist zu sehen, dass nach einem *Merge* eines *Branches* die einzelnen *Commits* die Versionen des *Main branches* bilden. Dabei wird kein separater *Merge commit* erstellt.

Abb. 6: Zeitliche Linearität der Methode „Fast-forward merge“

In Bezug auf die ISO 13485 ist die Historie einer Datei sehr wichtig für die Rückverfolgbarkeit und Dokumentation der einzelnen Änderungen und Versionen. Dabei geht es darum wer, wann, welche Änderungen aus welchem Grund (per *Commit*- und MR-Beschreibung) vorgenommen hat. Der Zugang zu diesen Informationen und die Aufbewahrungsfrist der Dateien und deren Historien müssen jederzeit einsehbar sein, um die Anforderungen eines Audits zu erfüllen. Genau das ist mit der Historie, die automatisch in GitLab zu den einzelnen Dateien erstellt wird, möglich. Dabei ist bei einem Zusammenfügen mehrerer *Commits* nicht mehr gewährleistet, dass jeder *Commit* plus dessen *Commit-Beschreibung* einsehbar ist. Diese Option ist damit nicht zu empfehlen. Als *Merge*-Methode ist daher der *Merge commit with semi-linear history* ohne *squashing* der *Commits* zu empfehlen.

Die Anleitung für die Einstellungen der Historie in Projekten befindet sich in Anhang IV. Wo die Historie einer Datei zu finden ist kann den *GitLab Docs* entnommen werden [32]. Sie beinhaltet lediglich einen Vorschlag für die einzustellenden Optionen auf Grundlage der beschriebenen Möglichkeiten der Historien-Erzeugung im *Main branch* und den *Merge-Methoden*.

4.4 Automatisierung – Generierung eines PDF-Dokuments

Aus einem vorhandenen LaTeX-Dokument soll ein gleichnamiges PDF-Dokument mit demselben Inhalt generiert werden. Dies geschieht in diesem Fall nach jeder *Tag-Erstellung* eines Repos. Voraussetzung dafür ist, dass sich in dem Repo eine entsprechende `.gitlab-ci.yml`-Datei befindet. Es wird dann automatisch der *Build-Job* gestartet. Dieser *Build-Job* greift dann auf den angegebenen *Docker Container* (s. Kapitel 4.2.7) zu, sodass die dort enthaltenen Tools für die PDF-Generierung verwendet werden. Das generierte PDF-Dokument wird nicht zu dem Repo in GitLab hinzugefügt, sondern steht als Artefakt in dem Projekt in GitLab zur Verfügung. Die Anleitung zur Generierung eines PDF-Dokuments befindet sich in Anhang V.

4.5 Sonstiges

An dem gewählten Ablageort der PDF-Dokumente, liegen ausschließlich PDF-Dokumente vor. Diese sind auf *read only* eingestellt. Dies ist unter anderem eine Sicherheitsmaßnahme, damit nicht aus Versehen etwas an dem Inhalt der Dokumente direkt an dem Ablageort geändert werden kann (z.B. eine Signatur oder ein Kommentar einfügen). Sie können aber kopiert und verschickt werden.

5 Workflow in GitLab und Sourcetree

Zuerst muss ein Projekt mit einem Repo und einer Readme-Datei angelegt werden. Um eine Änderung in dem Repo vorzunehmen, muss das Repo in Sourcetree geklont werden. Anschließend muss ein *Branch* in Sourcetree erstellt werden. In diesem *Branch* können nun Änderungen in dem lokalen Repo vorgenommen werden. Dabei können Dateien hinzugefügt, entfernt oder bearbeitet werden. Änderungen werden mit einem *Commit* und der dazugehörigen *Commit-Beschreibung* vorerst nur lokal gespeichert. Wurden alle gewünschten Änderungen vorgenommen, wird ein *Push* zu GitLab durchgeführt, damit die *Commits* in GitLab vorhanden sind. Anschließend wird in GitLab eine MR gestellt und die Reviewer ausgewählt. Die Reviewer *approven* und *mergen* dann die Änderungen in GitLab, wenn diese in Ordnung sind. Abschließend sind die Änderungen in GitLab vorhanden und die *Commit*-Historie wurde in GitLab übernommen und gespeichert.

Der allgemeine Workflow ist in Abb. 7 dargestellt.

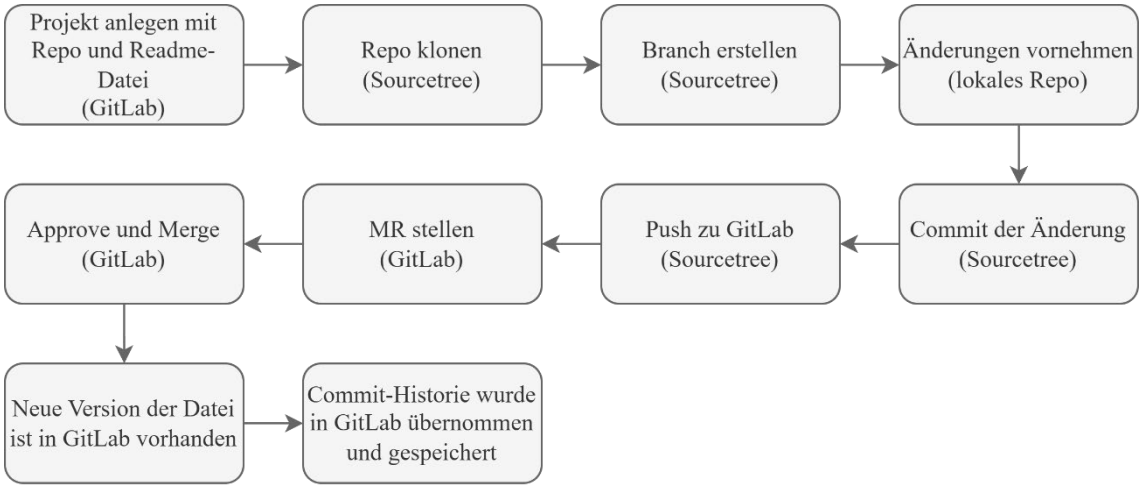

Abb. 7: Allgemeiner Workflow

Die genauen Schritte des Workflows sind in den folgenden Abschnitten genauer erklärt und dargestellt. Bei den folgenden Workflows wird vorausgesetzt, dass die Installationen und Einstellungen aus den vorigen Kapiteln sowie aus den Anhängen I, II, III, IV und V vorgenommen wurden.

5.1 Erstellen eines Repos

Derjenige, der Eigentümer des neuen Projekts sein soll, erstellt ein neues Repo in GitLab. Der Eigentümer der Projekte sollte einer der Administratoren sein, da diese in GitLab selbst mehr Berechtigungen besitzen als andere Personen und das Einbinden benötigter Dateien von ihnen vorgenommen werden kann. Zudem sollten sie Zugang zu allen Projekten haben. Das Repo ist entweder leer, basiert auf einer Vorlage oder importiert ein komplettes anderes Projekt. Bei einem leeren Projekt sollte immer ein Repo plus eine Readme-Datei angelegt werden, damit automatisch ein Repo erstellt wird und direkt Dateien hinzugefügt werden können. Im Nachhinein ein Repo in dem erstellten Projekt anzulegen und Dateien hinzuzufügen wird ansonsten sehr kompliziert. Danach wird von dem Eigentümer die Zugangs- und Handlungsberechtigung anderer Personen für das Repo festgelegt. Dies kann sowohl über die Auswahl einzelner Personen als auch Personengruppen erfolgen. Die Erstellung eines Repos wird in der Historie als *Commit* automatisch von GitLab erstellt. Die Anleitung für die Erstellung eines Repos befindet sich in Anhang VI.

Bei QMS-Dokumenten müssen inhaltlich die Anforderungen der ISO 13485 eingehalten werden, wodurch das Layout und gewisse Inhalte in jedem Dokument identisch sein sollten. Dabei bietet es sich an ein Repo mit den Vorlage-Dateien zu erstellen und dieses dann entweder als Vorlage für ein neu erstelltes Repo zu verwenden oder als Submodule (s. Anhang VII) darauf zuzugreifen. Die Verwendung von Submodules ist lediglich eine Option und muss nicht genutzt werden. In Abb. 8 ist der Ablauf noch einmal grafisch dargestellt:

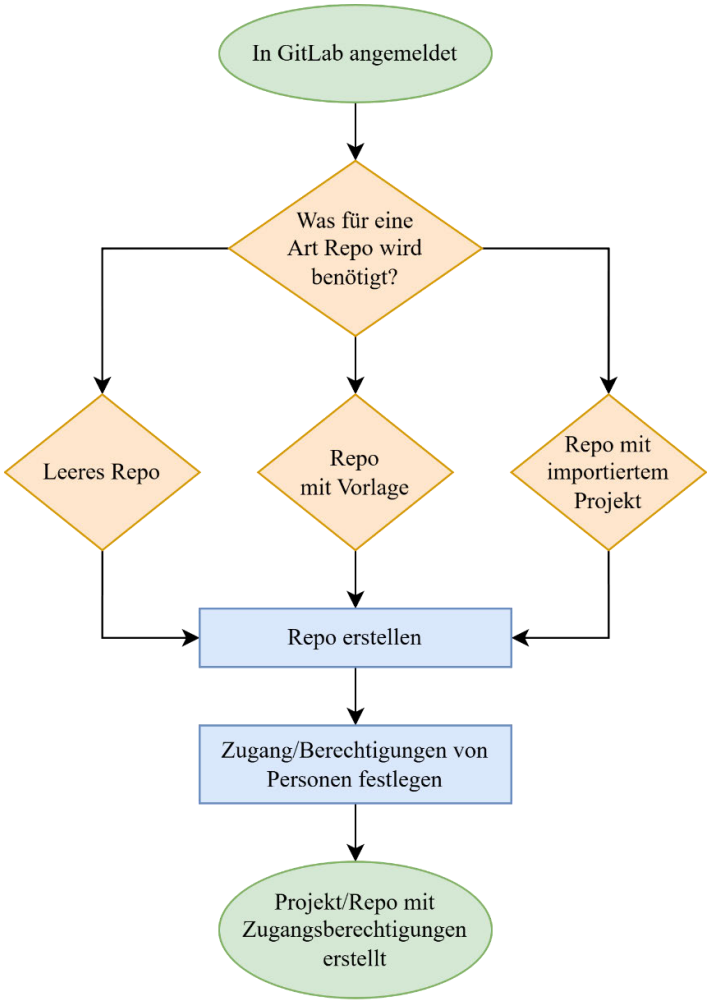

Abb. 8: Neues Projekt/Repo in GitLab erstellen

5.2 Hinzufügen/Entfernen einer Datei zu/aus einem Repo

Nachdem ein Repo erstellt wurde, sollen nun Dateien hinzugefügt werden. Dies funktioniert mit Hilfe eines Git-Clients, hier Sourcetree.

Zuerst erfolgt die Anmeldung eines Mitarbeiters, der eine Datei in das Repo hinzufügen oder entfernen möchte, in GitLab. Anschließend wird dort das gewünschte Projekt ausgewählt. Wenn kein SSH-Key verwendet wird, muss die URL des Repos mit HTTPS kopiert werden. In Sourcetree wird dann ein neuer Tab angelegt und die URL von GitLab eingefügt. Anschließend werden lokaler Pfad und Name automatisch vorgeschlagen. Dies kann aber manuell geändert werden. Nachdem die besagten Felder ausgefüllt sind, wird das Repo von GitLab mit dem gesamten Inhalt als lokales Repo gespeichert. Danach muss von dem Mitarbeiter ein *Branch* angelegt werden, der einen Namen erhält, welcher die vorgesehene Änderung einer oder mehrerer Dateien kurz und präzise beschreibt. Die weitere Bearbeitung erfolgt nun ausschließlich mit den Dateien in dem lokalen Repo. Um sicherzustellen, dass an dem richtigen Ort gearbeitet wird, kann über Sourcetree der lokale Pfad des Repos geöffnet werden. In das lokale Repo können nun Dateien hinzugefügt oder entfernt werden. Alle Dateien, die sich in dem Repo befinden, werden automatisch von Sourcetree erkannt und als neue noch nicht eingebundene Datei angezeigt. Die Dateien werden einzeln in Sourcetree zu dem Repo hinzugefügt und per *Commit* und *Commit-Beschreibung* integriert. Per *Push* werden alle *Commits*, die sich noch nicht auf dem Git-Server befinden, an GitLab gesendet. Dort sind sie dann in dem erstellten *Branch* vorhanden und können heruntergeladen werden. Sie sind aber noch nicht freigegeben. Anschließend startet der Freigabeprozess per MR. Die Anleitung für das Hinzufügen bzw. Entfernen einer Datei zu/aus einem Repo befindet sich in Anhang VIII. In Abb. 9 ist der Ablauf noch einmal grafisch dargestellt:

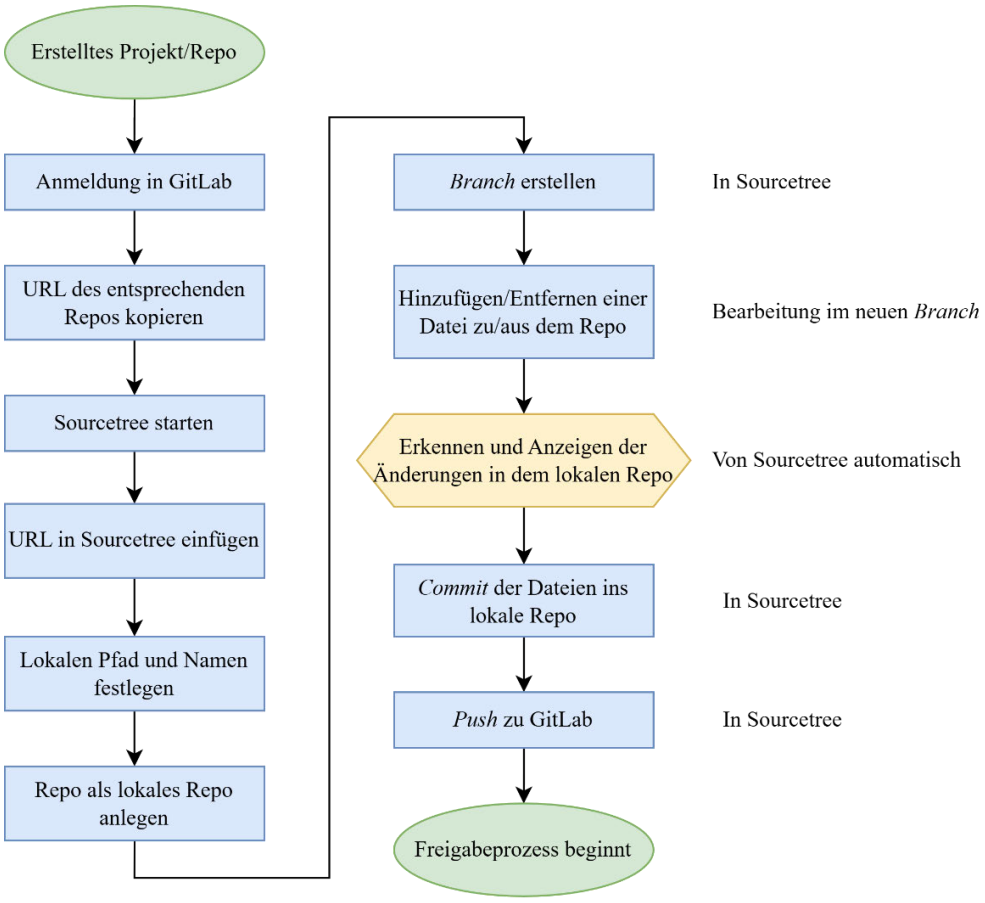

Abb. 9: Hinzufügen/Entfernen einer Datei zu/aus einem Repo

5.3 Ändern einer vorhandenen Datei in einem Repo

Die ersten Schritte zur Änderung einer Datei in einem vorhandenen Repo entsprechen dem Ablauf von Kapitel 5.2 bis zu dem Punkt, an dem ein *Branch* von dem Mitarbeiter angelegt wurde.

Es wird auch bei der Änderung einer vorhandenen Datei, welche ein Mitarbeiter vornimmt, ausschließlich mit den Dateien im lokalen Repo gearbeitet. Um sicher zu gehen, dass auf die Dateien in dem richtigen lokalen Repo zugegriffen wird, kann in Sourcetree der lokale Pfad des Repos geöffnet werden. Dort wird die zu ändernde Datei manuell geöffnet und inhaltlich geändert. Diese Änderung wird automatisch von Sourcetree erkannt und als Differenz zu der vorigen Version des Dateiinhalts derselben Datei farblich angezeigt. Die abweichenden Dateien zur vorigen Version müssen manuell zu dem nächsten *Commit* hinzugefügt werden. Die jeweils neue Version der Dateien wird dann per *Commit* und *Commit-Beschreibung* lokal gespeichert. *Commits* sollten regelmäßig erfolgen, um die Änderungen zu sichern, falls unerwartete Probleme auftreten (z.B. Absturz des Endgeräts). Sind alle vorgesehenen Änderungen vorgenommen worden, erfolgt ein *Push* aller getätigten *Commits* zu GitLab. Auch ein *Push* sollte regelmäßig erfolgen, um den Fortschritt in GitLab zu sichern, falls lokal gespeicherter Fortschritt verloren gehen sollte. In GitLab sind die *Commits* dann in dem erstellten *Branch* vorhanden und können heruntergeladen werden. Sie sind aber noch nicht freigegeben. Anschließend startet der Freigabeprozess per MR. Die Anleitung für das Ändern einer vorhandenen Datei in einem Repo befindet sich in Anhang IX. In Abb. 10 ist der Ablauf noch einmal grafisch dargestellt:

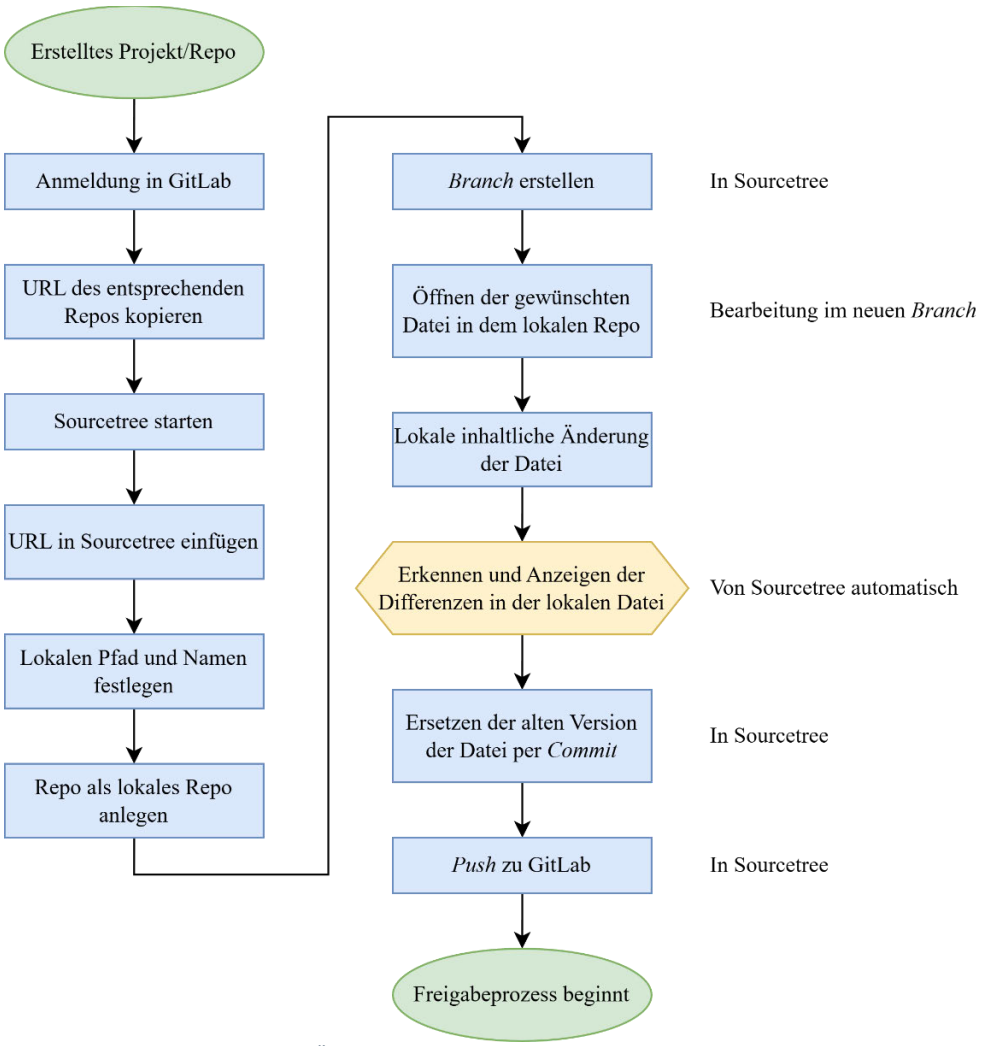

Abb. 10: Ändern einer vorhandenen Datei in einem Repo

5.4 Freigabeprozess eines Repos

Der Freigabeprozess beginnt, wenn eine MR gestellt wird. Eine MR wird immer in GitLab direkt gestellt. Die Person, die die *Commits* vorgenommen und zu GitLab *gepusht* hat, loggt sich in GitLab ein und stellt eine MR in dem entsprechenden Projekt. Diese ist an auswählbare Reviewer gerichtet (i.d.R. mindestens zwei). Die Reviewer erhalten eine Benachrichtigung per E-Mail, dass eine MR für ein Repo gestellt wurde, bei der sie als Reviewer ausgewählt wurden. Diese loggen sich daraufhin in GitLab ein und navigieren zu den offenen MRs.

Mit einem Klick auf den entsprechenden Auftrag öffnet sich dieser und das Prüfen des geänderten Inhalts durch die Reviewer kann beginnen. Unter „*Changes*“ werden nun die Änderungen einer Datei zur vorigen oder einer anderen auswählbaren Version der gleichen Datei als „*diff*“ (Differenz) angezeigt. In Grün werden die hinzugefügten und in Rot die geänderten bzw. entfernten Zeilen angezeigt (Standardeinstellung). Dies ist entscheidend für den Reviewer, da hier die Unterschiede visuell deutlich erkennbar sind und überprüft werden kann, ob die vorgenommenen Änderungen in Ordnung sind oder sie nochmals überarbeitet werden müssen.

Fall 1: Die Änderungen werden direkt genehmigt

Bei direkter Genehmigung der Änderungen werden diese von den Reviewern *approved*. Dabei kann optional noch ein Kommentar hinzugefügt werden. Erst wenn alle Reviewer *approved* haben, kann die MR *gemerged* werden. Der *Merge* wird von dem letzten Reviewer durchgeführt. Die Person, die die MR gestellt hat, wird über den erfolgreichen *Merge* per E-Mail benachrichtigt. Diese Person ist dann mit ihrer Arbeit an dem Repo fertig.

Fall 2: Änderungen werden nicht genehmigt

Einer der Reviewer genehmigt die Änderungen nicht, sodass eine erneute Überarbeitung des Inhalts der Datei nötig ist. Zu den nicht genehmigten Änderungen werden jeweils Kommentare von dem Reviewer verfasst, warum sie nicht genehmigt wurden bzw. was daran überarbeitet werden muss. Dabei wird kein *Approve* durchgeführt. Das Feedback mit den Kommentaren des Reviewers wird per E-Mail an die Person geschickt, die die MR gestellt hat. Diese überarbeitet die nicht genehmigten Änderungen und *pusht* diese zu GitLab. Die Reviewer werden per E-Mail benachrichtigt, dass ein neuer *Push* zu einer vorhandenen MR getätigt wurde und überprüfen den neuen Inhalt. Werden diese Änderungen ebenfalls nicht genehmigt, beginnt der Prozess von vorn. Wenn die Änderungen genehmigt werden, wird von allen Reviewern ein *Approve* durchgeführt und letztendlich ein *Merge*. Die Person, die die MR gestellt hat, wird über den erfolgreichen *Merge* per E-Mail benachrichtigt und ist damit mit der Arbeit an dem Repo fertig.

Weiteres Verfahren in beiden Fällen

Der erstellte *Branch* für die vorgenommene Änderung wird nicht länger benötigt und kann nun zeitgleich mit dem erfolgreichen *Merge* gelöscht werden. Es bietet sich an bei dem *Merge* eine *Merge commit message* zu schreiben, da dies Teil der Dokumentation ist. Die Änderungen werden in den *Main branch* übernommen. Die jeweils neue aktuelle Version der Dateien befindet sich nun in GitLab und kann von dort heruntergeladen werden. In der Historie kann jederzeit nachvollzogen werden, wer wann welche Änderungen vorgenommen und wer diese *approved* und *gemerged* hat. Die Anleitung für den Freigabeprozess eines Repos befindet sich in Anhang X.

In Abb. 11 ist der Ablauf noch einmal grafisch dargestellt:

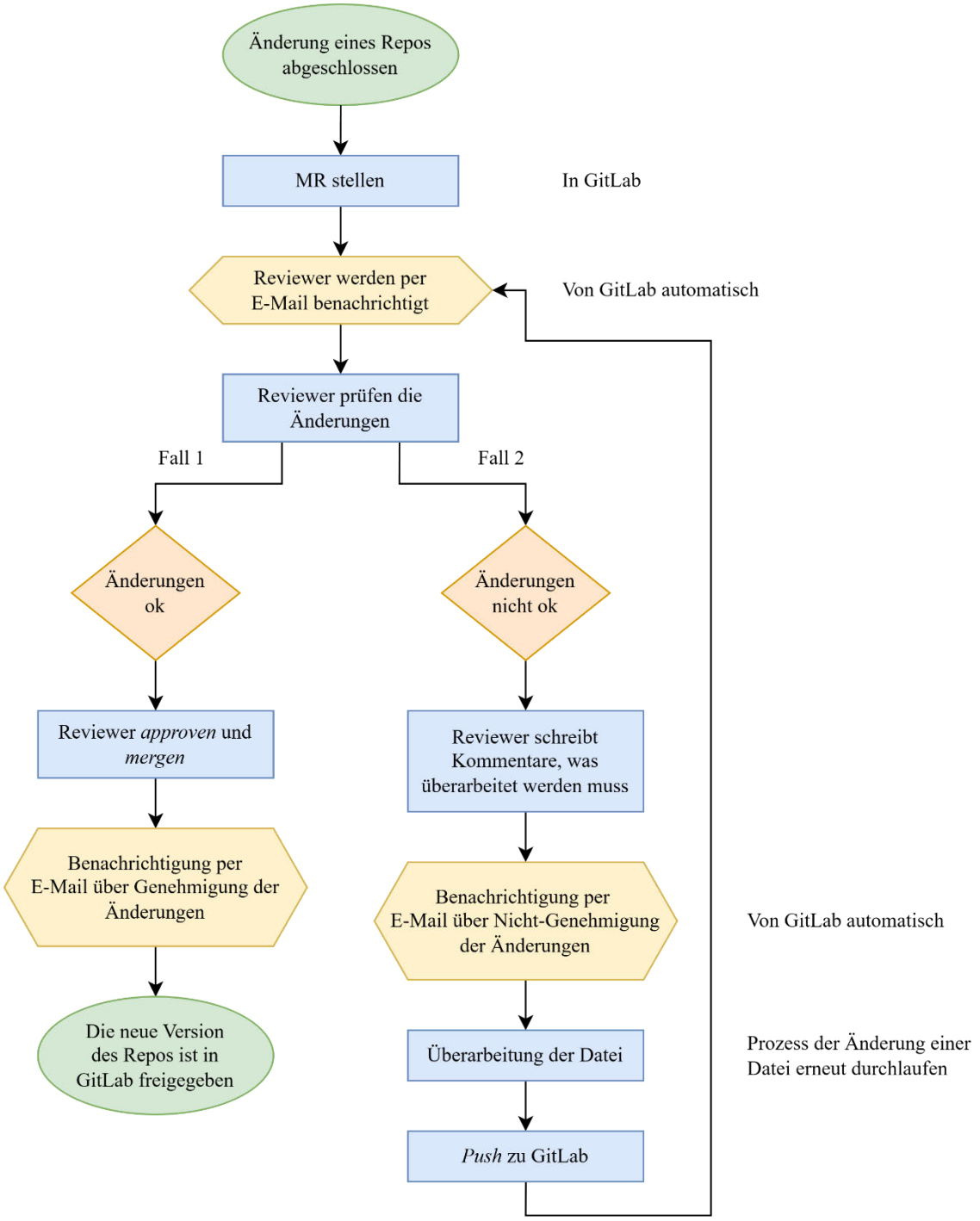

Abb. 11: Freigabeprozess von Änderungen eines Repos in GitLab

5.4.1 Konfliktlösung

Problem: Es wurde in zwei unterschiedlichen *Branches* desselben *target branches* (meistens der *Main branch*) gleichzeitig gearbeitet und es wurden sich überschneidende Änderungen an derselben Datei vorgenommen. Die Problematik ist in folgender Abb. 12 grafisch dargestellt.

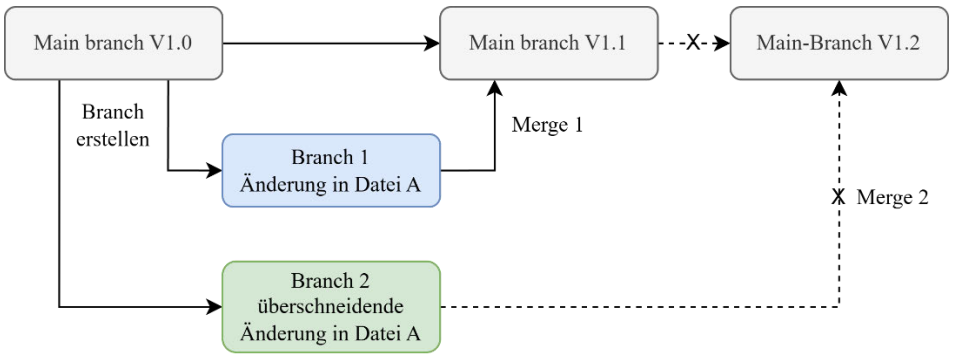

Abb. 12: Konflikt bei sich überschneidenden Änderungen derselben Datei in zwei Branches ausgehend von demselben source branch

Beschreibung: Die Änderung aus *Branch 1* wird ohne Probleme in den *Main branch gemerged*. Durch den neuen *Commit* ist allerdings eine neue Version des *target branches* (hier: V1.1) entstanden. Somit hat die zweite Änderung (hier: *Branch 2*), die zum *Mergen* geschickt wird, eine alte Version des *target branches* (hier: V1.0) als Grundlage verwendet. Die Änderung kann deshalb nicht direkt *gemerged* werden.

Lösung: Die Person, die Änderungen in *Branch 2* vorgenommen hat, muss ein *Rebase* des *source branches* (hier: *Branch 2*) in GitLab durchführen, damit sie mit dem aktuellen Stand des letzten freigegebenen *Commits* arbeiten kann. Die Konflikte müssen dann manuell in der Datei gelöst werden. Danach kann die Version aus *Branch 2* per MR in den *target branch* als neue Version *gemerged* werden. Wie ein *Rebase* in GitLab vorgenommen wird, kann den *GitLab Docs* entnommen werden [33]. Bei der hier gewählten *Merge-Methode* wird ein Konflikt erkannt und die Option eines *Rebase* gegeben [34]. Die Lösung ist in folgender Abb. 13 grafisch dargestellt.

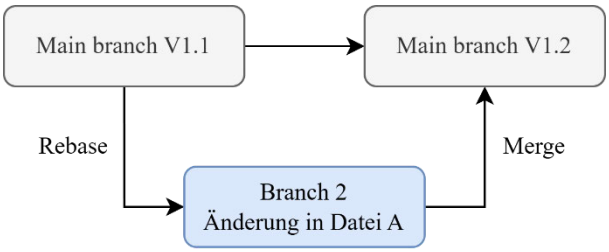

Abb. 13: Lösung zu dem Konflikt bei sich überschneidenden Änderungen derselben Datei in zwei Branches

Um Konflikte bei der Bearbeitung derselben Datei zu vermeiden, wird sehr empfohlen sich so zu organisieren, dass nur eine Person zur gleichen Zeit an einer Datei arbeitet. Ein Rebase muss ansonsten für jeden in der Zwischenzeit von anderen Personen getätigten Commit der Datei durchgeführt werden. Dies kann sehr mühsam und zeitaufwendig werden. Sollte der Fall trotzdem eintreten, kann die eigene Änderung aus dem lokalen Repo an einem anderen Ort lokal abgelegt werden und ein Pull per Sourcetree erfolgen, sodass der Stand des letzten Commits im lokalen Repo ist. Dann wird ein neuer Branch erstellt und die vorigen Änderungen dort übertragen.

5.5 PDF-Dokument zur Verfügung stellen

Bei einer *Tag-Erstellung* (hier: *annotated tag*) eines Repos wird automatisch der *Build-Job* für die PDF-Generierung ausgeführt (s. Kapitel 4.4). Nachdem ein PDF-Dokument generiert wurde, soll es nun an dem gewünschten Ort abgelegt werden. Hierbei wird das PDF-Dokument in GitLab für eine gewisse Zeit zum Herunterladen als ein Artefakt in dem entsprechenden Projekt zur Verfügung gestellt. Dort muss es als ZIP-Archiv heruntergeladen werden. Anschließend wird es an dem gewünschten Ablageort manuell abgelegt und ersetzt die alte Version desselben Dokuments. Da *Tags* einen formalen Stand einer Datei kennzeichnen, sollte die Geschäftsführung die *Tag-Erstellung* durchführen. Die Geschäftsführung sollte immer den Überblick über die Dateien haben und wissen, wann die vorgenommenen Änderungen sinnvoll sind mit einem *Tag* zu kennzeichnen. Es ist sinnvoll, dass derjenige, der die *Tag-Erstellung* durchgeführt hat, ebenfalls das PDF-Dokument zur Verfügung stellt. Der Ablageort muss für jeden Mitarbeiter zugänglich sein. Die Anleitung für das zur Verfügung stellen eines PDF-Dokuments befindet sich in Anhang XI.

In Abb. 14 ist der Ablauf noch einmal vereinfacht grafisch dargestellt:

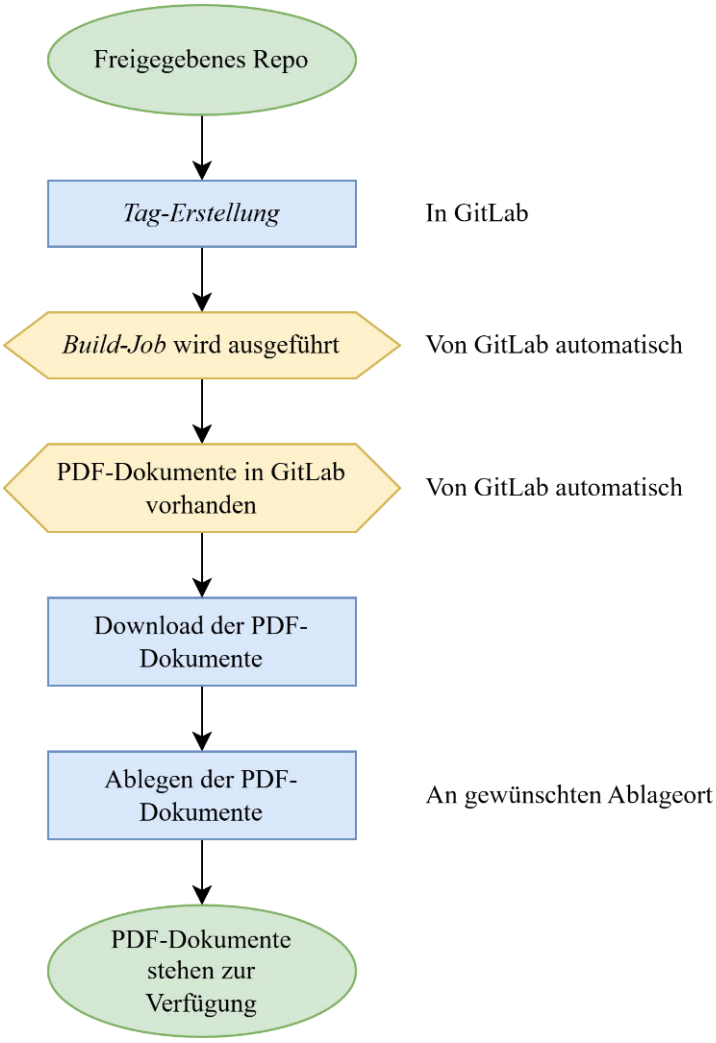

Abb. 14: PDF-Dokument zur Verfügung stellen

5.6 Bewertung der ISO-Konformität bezüglich des Workflows

Folgend ist die Umsetzung der einzelnen Anforderungen der ISO 13485 aus Kapitel 3.2 im Bezug zu dem hier dargestellten Workflow zusammengefasst.

Die Dokumentenlenkung wird mit den MRs und dessen *MR-Beschreibungen* sowie den *Approvals*, dessen *Approval-Regeln* und den *Merges* durch ausgewählte Reviewer umgesetzt. Dadurch muss jedes Dokument vor der Verwendung geprüft und genehmigt werden. Die *Commit-Historie* mit den dort enthaltenen *Commit-Beschreibungen* setzt die Dokumentation und Nachvollziehbarkeit der Änderungen von Dateien in GitLab um. Wichtige Versionen werden durch *Tag-Erstellungen* und Releases gekennzeichnet.

Mit einer klaren und eindeutigen Repo-Struktur und den darin enthaltenen Dateien, die sinnvolle Dateinamen enthalten, wird die eindeutige Identifikation von Dateien gewährleistet.

Dadurch dass die gültige Version eines Dokuments als PDF-Dokument an dem gewählten Ablageort (z.B. Netzlaufwerk, Cloud) für alle Mitarbeiter zur Verfügung steht, wird der Zugang dort, wo das Dokument benötigt wird, sichergestellt. Damit ist die unbeabsichtigte Verwendung von ungültigen oder veralteten Dokumenten ausgeschlossen. In GitLab wird dies durch die *Branch-Regeln* des *protected branches* umgesetzt, sodass alte Versionen nur über die Historie abrufbar sind. Auf die freigegebenen Dateien in den Repos können die Mitarbeiter jederzeit zugreifen, wenn sie die entsprechende Zugriffsberechtigung dafür besitzen.

Durch die Verwendung von digitalen Dokumenten in Form von LaTeX- und PDF-Dokumenten ist die Lesbarkeit immer gewährleistet.

Die Archivierung und Aufbewahrungsfrist der Dokumente werden durch archivierte Projekte, die *Commit-Historie*, *Tags* und Releases umgesetzt. Diese bleiben unverändert bestehen und sind jederzeit einsehbar. Dies ist besonders für Audits wichtig.

Fazit

Bei korrekter Anwendung des Workflows in GitLab, den richtig vorgenommenen Einstellungen in GitLab für den Freigabeprozess, die Historien-Erstellung, den entsprechenden Rollenzuweisungen bzw. Berechtigungen einzelner Personen, die Einhaltung geforderter Informationen in den *Commit-* und *Merge-Beschreibungen*, eine geeignete Repo-Struktur und die Einhaltung der Aufbewahrungsfrist ist die ISO 13485 konforme Verwendung im Bezug zu dem Workflow gewährleistet.

6 Computersystemvalidierung

6.1 Risikoanalyse

In dieser Risikoanalyse geht es u.a. um den Zugang zu den verwendeten Daten durch unbefugte externe Personen. Wenn sensible Daten ungewollt an Dritte gelangen, kann dies schwerwiegende Konsequenzen für das Unternehmen haben. Dieses Risiko gilt es so gering wie möglich zu halten. Hierbei wird die Verwendung von GitLab, *Docker Containern* und Sourcetree betrachtet. [35]

Der Zugang zu den Daten von GitLab ist auch ein wichtiger Bestandteil der Risikoanalyse, da ohne den Zugriff auf die Daten eventuell nicht weitergearbeitet werden kann. Es können dann z.B. keine Dokumente an Kunden gesendet werden, die dringend benötigt werden, oder intern Dokumente überarbeitet werden. Auch für Audits muss der Zugang zu den Daten gewährleistet sein. Die Sicherstellung des Zugangs zu den Daten in GitLab ist daher zwingend erforderlich.

6.1.1 Zugang zu Daten durch unbefugte externe Personen

GitLab

Da sich der Git-Server im Intranet befindet, ist der Zugriff auf die Daten in GitLab durch unbefugte Personen ausgeschlossen und stellt im Bezug zum Datenschutz kein Risiko dar (s. Kapitel 4.1.1).

Docker Container

Dadurch dass *Container* in einer isolierten Umgebung arbeiten, kann von außerhalb nicht auf sie zugegriffen werden (s. Kapitel 4.2.7). Es können auch keine Daten von Unbefugten abgefangen werden. Mit dem Löschen bzw. Entfernen eines *Docker Containers* kann nichts auf dem Git-Server selbst zerstört werden. Der Zugriff durch unbefugte Personen, auf die vom *Docker Container* verwendeten und generierten Daten, ist ausgeschlossen und stellt im Bezug zum Datenschutz kein Risiko dar.

Sourcetree (hier: Version 3.4.30)

Generell besitzt Sourcetree eine gute Sicherheitsstufe, solange immer die aktuelle Version verwendet wird, da gelegentlich auftretende Sicherheitslücken und Fehler meist schnell erkannt und in einem neuen Update durch Atlassian behoben werden [36]. Eine sichere Methode zur Authentifizierung (SSH-Key, Personal Access Token) ist ebenfalls ratsam zu verwenden, um die Sicherheit zu erhöhen. Da Sourcetree mit Git interagiert, sollten ebenfalls die neuesten Updates für Git und OpenSSH installiert sein. [37]

Da Sourcetree in diesem Fall nur auf den Git-Server des Unternehmens zugreift und nur von dem Unternehmen selbst erstellte Repos verwendet, sind an dieser Stelle einige der allgemeinen Risikoquellen irrelevant (manipulierte Git-Repos, Ausführung von Git-Hooks, Abhängigkeit von Drittbibliotheken). Das größte Risiko stellen gespeicherte Zugangsdaten dar, auf die ein Zugriff bei kompromittiertem Rechner möglich ist. [38]

Mögliche Maßnahmen zur Reduzierung der Risiken zusammengefasst:

- Aktuelle Version von Sourcetree verwenden
- Sichere Authentifizierungsmethode verwenden
- Updates für Git und OpenSSH installieren
- Alternativ einen anderen Client verwenden, dessen Sicherheitsrisiko geringer ist

6.1.2 Zugang zu den Daten von GitLab

Um die Erreichbarkeit des Git-Servers zu gewährleisten, muss das Unternehmensnetzwerk (per VPN) erreichbar und ein gültiges Zertifikat für den Server vorhanden sein. Damit sollte der Zugang zu den Daten gewährleistet sein. Das Risiko der Nicht-Erreichbarkeit des Git-Servers ist sehr gering, kann aber dennoch nicht ausgeschlossen werden. Als Maßnahme können Sicherungen (*Backups*) der Daten vorgenommen werden, um den Zugang zu ihnen außerhalb von GitLab zu gewährleisten. Welche Daten eines Projekts dabei exportiert werden und welche nicht sowie das Exportieren und Importieren eines Projekts und dessen Daten kann den *GitLab Docs* entnommen werden [39, 40]. Diese können an einem ausgewählten Ort (z.B. in einer Cloud) gesichert werden. Eine Anleitung zur Sicherung und Wiederherstellung der Daten von GitLab kann ebenso den *GitLab Docs* entnommen werden [41].

Der gewählte Ablageort, zu dem alle Mitarbeiter Zugang haben, gewährleistet außerhalb von GitLab den Zugang zu den PDF-Dokumenten. Jedoch kann mit diesen keine Änderung an einem Dokument vorgenommen werden. Das Risiko keinen Zugang zu den Daten von GitLab zu haben ist in Hinsicht auf eine Sicherung der Daten sehr gering.

Mögliche Maßnahmen zur Reduzierung der Risiken zusammengefasst:

- Erreichbarkeit des Unternehmensnetzwerks sicherstellen
- Auf ein gültiges Zertifikat für den Git-Server achten
- Sicherung in z.B. einer Cloud vornehmen
- Erreichbarkeit des Ablageorts der PDF-Dokumente sicherstellen

6.1.3 Risikomatrix

In der folgenden Tabelle 2 sind die Risiken auf Grundlage der durchgeführten Risikoanalyse unter Einbeziehung der möglichen Maßnahmen (s. Kapitel 6.1.1, 6.1.2) in einer Risikomatrix dargestellt. Hierbei wird die Wahrscheinlichkeit des Auftretens eines Risikos in Bezug zu dessen Auswirkung gestellt und dementsprechend eingeordnet.

Risiko	Keine Auswirkung	Leichte Auswirkung	Mittelschwere Auswirkung	Schwerwiegende Auswirkung
Keine Wahrschein-lichkeit	• Verwendung von GitLab (Intranet) • Verwendung von Containern			
Sehr geringe Wahrschein-lichkeit		• Kein Zugang zu Daten der Sicherung • Kein Zugang zu Daten des Ablageorts	• Kein Zugang zu Daten in GitLab	• Kein Zugang zu Daten der Sicherung + in GitLab + an dem Ablageort
Geringe Wahrschein-lichkeit			• Verwendung von Sourcetree	
Mittlere Wahrschein-lichkeit				
Hohe Wahrschein-lichkeit				

Tabelle 2: Risikomatrix basierend auf der Risikoanalyse

6.1.4 Bewertung der Risikoanalyse

Es wird an dieser Stelle Bezug zu der Risikoanalyse aus Kapitel 6.1 ff. und der dazugehörigen Risikomatrix genommen.

Die Verwendung von GitLab und den dort verwendeten *Docker Containern* stellt dementsprechend kein Sicherheitsrisiko bezüglich des Zugangs durch unbefugte Personen dar.

Die Verwendung von Sourcetree kann mit den in Kapitel 6.1.1 aufgeführten Maßnahmen und aufgrund der Verwendung weniger Funktionen ein geringes Maß an Risiko erreichen. Die Folgen einer kritischen Sicherheitslücke können allerdings einen höheren Schweregrad annehmen, da hier ungewollt Daten an Unbefugte übermittelt werden können [42]. Es kann u.a. deshalb ratsam sein einen anderen Git-Client zu verwenden, der ein geringeres Sicherheitsrisiko aufweist.

Der Zugang zu den Daten an dem Ablageort und der Sicherung stellen ein sehr geringes Risiko dar, welches maximal leichte Auswirkungen auf die laufende Arbeit der Mitarbeiter hat. Der Zugang zu den Daten in GitLab stellt ebenfalls ein geringes Risiko dar, allerdings wird dadurch

der Workflow deutlich stärker beeinträchtigt, da hier die aktuellen Daten aller Dateien vorliegen und mit ihnen gearbeitet wird.Die Auswirkungen sind dementsprechend als mittelschwer einzustufen. Schwerwiegende Auswirkungen auf die gesamte Arbeit treten in dem sehr unwahrscheinlichen Fall auf, dass der Zugang zu den Daten in GitLab plus am Ablageort plus der Sicherung nicht gegeben ist.

Insgesamt stellt die Verwendung von GitLab und Sourcetree im Bezug zu dem verwendeten Workflow ein geringes Sicherheitsrisiko dar, was den Zugang zu den Daten durch unbefugte externe Personen und generelle Sicherheitslücken betrifft. Bei fast jedem Risikofall können Maßnahmen eingeleitet werden, um die Sicherheit zu erhöhen.

6.2 Durchführung von Tests

Nachfolgend sind Testfälle und deren Testschritte bezogen auf einzelne Anforderungen beschrieben. Anschließend wird das zu erwartende Ergebnis und das tatsächliche Ergebnis nach der Durchführung der Testschritte beschrieben. Abschließend wird entschieden, ob die jeweilige Validierung erfolgreich war.

6.2.1 Einstellungen in GitLab

Da die Einstellungen, die in GitLab vorgenommen werden, von GitLab selbst stammen, wird davon ausgegangen, dass diese das gewünschte Ergebnis liefern. Dementsprechend wird an dieser Stelle nicht näher darauf eingegangen und es werden keine einzelnen Tests durchgeführt.

6.2.2 Automatisierung

Testfall 1: PDF-Generierung aus einem LaTeX-Dokument per Build-Job nach Tag-Erstellung eines Repos

Anforderung: Nach einer *Tag-Erstellung* eines Repos in GitLab sollen mit dem *Build-Job* automatisch PDF-Dokumente auf Grundlage aller in dem Repo befindlichen LaTeX-Dokumente generiert werden und anschließend in GitLab als Artefakte zum Download zur Verfügung stehen.

Testschritte: Voraussetzung der Umsetzung von Anhang II und V. Durchführung der Anleitung aus Anhang XI Schritte 1 bis 9.

Zu erwartendes Ergebnis: Der *Build-Job* wurde erfolgreich ausgeführt. Die PDF-Dokumente stehen in GitLab als Artefakte zum Download in einem ZIP-Archiv zur Verfügung und besitzen denselben Inhalt der verwendeten LaTeX-Dokumente.

Tatsächliches Ergebnis: Der *Build-Job* wurde erfolgreich ausgeführt. Die PDF-Dokumente stehen in GitLab als Artefakte zum Download in einem ZIP-Archiv zur Verfügung und besitzen denselben Inhalt der verwendeten LaTeX-Dokumente.

Validierung erfolgreich: Ja

6.2.3 Workflow in GitLab und Sourcetree

Testfall 2: Erstellung eines Projekts mit Repo in GitLab

Anforderung: Ein leeres Projekt soll mit Repo und Readme-Datei in GitLab erstellt werden. Es soll anschließend in GitLab vorhanden sein.

Testschritte: Durchführung der Anleitung aus Anhang VI.

Zu erwartendes Ergebnis: Das Projekt mit dem eingegebenen Namen wurde mit einem Repo, in dem sich eine Readme-Datei befindet, erstellt und wird in GitLab als vorhandenes Projekt angezeigt.

Tatsächliches Ergebnis: Das Projekt mit dem eingegebenen Namen wurde mit einem Repo, in dem sich eine Readme-Datei befindet, erstellt und wird in GitLab als vorhandenes Projekt angezeigt.

Validierung erfolgreich: Ja

Testfall 3: Hinzufügen einer Datei zu einem lokalen Repo in Sourcetree

Anforderung: In ein lokales Repo soll eine Datei unter Verwendung von Sourcetree und einem dort erstellten lokalen *Branch* hinzugefügt werden. Ein neuer lokaler *Branch* soll in Sourcetree vorhanden sein**.** Die Datei soll nach einem *Commit* in Sourcetree in dem neuen lokalen *Branch* des lokalen Repos vorhanden sein

Testschritte: Durchführung der Anleitung aus Anhang VIII.

Zu erwartendes Ergebnis: Der neue lokale *Branch* ist mit der hinzugefügten Datei in Sourcetree im lokalen Repo vorhanden. Es wird keine neue Datei in dem lokalen Repo von Sourcetree erkannt und angezeigt.

Tatsächliches Ergebnis: Der neue lokale *Branch* ist mit der hinzugefügten Datei in Sourcetree im lokalen Repo vorhanden. Es wird keine neue Datei in dem lokalen Repo von Sourcetree erkannt und angezeigt.

Validierung erfolgreich: Ja

Testfall 4: Ändern einer vorhandenen Datei in einem lokalen Repo in Sourcetree

Anforderung: Ein vorhandenes LaTeX-Dokument eines lokalen Repos soll in einem neu erstellten lokalen *Branch* inhaltlich geändert werden. Diese Differenz zur Vorgängerversion soll in Sourcetree dargestellt werden. Mit einem *Commit* soll die vorige Version durch die neue Version ersetzt werden. Danach darf keine inhaltliche Differenz der Datei in dem lokalen Repo von Sourcetree erkannt und angezeigt werden.

Testschritte: Durchführung der Anleitung aus Anhang IX Schritte 1 bis 14.

Zu erwartendes Ergebnis: Die inhaltliche Änderung der Datei wird von Sourcetree erkannt und als Differenz angezeigt. Nach dem *Commit* wird die Änderung in dem lokalen *Branch* übernommen, sodass keine Differenz mehr von Sourcetree erkannt und angezeigt wird.

Tatsächliches Ergebnis: Die inhaltliche Änderung der Datei wird von Sourcetree erkannt und als Differenz angezeigt. Nach dem *Commit* wird die Änderung in dem lokalen *Branch* übernommen, sodass keine Differenz mehr von Sourcetree erkannt und angezeigt wird.

Validierung erfolgreich: Ja

Testfall 5: Push zu GitLab in Sourcetree und MR in GitLab stellen

Anforderung: Der in Sourcetree getätigte *Commit* soll per *Push* in Sourcetree zu GitLab geschickt werden. Dort soll eine MR für die Genehmigung des *Commits* gestellt werden. Reviewer müssen ausgewählt werden.

Testschritte: Durchführung der Anleitungen aus Anhang IX Schritte 15 bis 16 und Anhang X Schritte 1 bis 12.

Zu erwartendes Ergebnis: Die *gepushten Commits* sind in dem entsprechenden *Branch* vorhanden. Die ausgewählten Reviewer bekommen eine E-Mail-Benachrichtigung über die neu gestellte MR. Die MR ist für diese Reviewer in GitLab sichtbar und vorhanden.

Tatsächliches Ergebnis: Die *gepushten Commits* sind in dem entsprechenden *Branch* vorhanden. Die ausgewählten Reviewer bekommen eine E-Mail-Benachrichtigung über die neu gestellte MR. Die MR ist für diese Reviewer in GitLab sichtbar und vorhanden.

Validierung erfolgreich: Ja

Testfall 6: Erfolgreicher Merge eines Repos in GitLab

Anforderung: Nach einem erfolgreichen *Merge* in GitLab sollen die vorgenommenen Änderungen freigegeben sein und die neue Version der Datei soll in GitLab zur Verfügung stehen. Ein neuer Eintrag in der Historie der Datei soll vorhanden sein.

Testschritte: Durchführung der Anleitung zur direkten Genehmigung der Änderungen aus Anhang X Schritte 13 bis 19.

Zu erwartendes Ergebnis: Eine neue Version der Datei steht in GitLab zur Verfügung und es ist ein neuer Eintrag in der Historie vorhanden.

Tatsächliches Ergebnis: Eine neue Version der Datei steht in GitLab zur Verfügung und es ist ein neuer Eintrag in der Historie vorhanden.

Validierung erfolgreich: Ja

Testfall 7: PDF-Dokument zur Verfügung stellen

Anforderung: Ein aus einem LaTex-Dokument generiertes PDF-Dokument soll an einem ausgewählten Ablageort allen Mitarbeitern zur Verfügung stehen.

Testschritte: Durchführung der Anleitung aus Anhang XI Schritte 8 bis 11.

Zu erwartendes Ergebnis: Das PDF-Dokument ist für alle Mitarbeiter an dem ausgewählten Ablageort zugänglich.

Tatsächliches Ergebnis: Das PDF-Dokument ist für alle Mitarbeiter an dem ausgewählten Ablageort zugänglich.

Validierung erfolgreich: Ja

6.2.4 Bewertung der Durchführung der Testfälle

Die Validierung der Einstellungen in GitLab, der Automatisierung und des Workflows in GitLab und Sourcetree wurden erfolgreich abgeschlossen, sodass die zu erwartenden Ergebnisse der jeweiligen Testfälle ohne Einschränkungen eingetreten sind.

7 Abschließendes Fazit

Die Verwendung von GitLab und Sourcetree für einen Workflow bezogen auf die Versionsverwaltung von QMS-Dokumenten kann bei korrekter Anwendung die Anforderungen der ISO 13485 erfüllen. Generell ist der hier dargestellte Workflow sehr benutzerfreundlich. Lediglich das Installieren und Registrieren von *Containern*, das Erstellen deren *Images* und der *Build-Jobs*, die auf die *Container* zugreifen, müssen von Fachpersonal durchgeführt werden. Der Git-Client Sourcetree kann für diesen Workflow durch einen anderen ersetzt werden, da nur wenige essenzielle Funktionen verwendet werden. Dadurch kann die Sicherheit erhöht werden. Das Sicherheitsrisiko ist allgemein sehr gering und insbesondere die Verwendung von GitLab ist sicher im Bezug zum Datenschutz vor unbefugten Personen, solange sich der Git-Server im Intranet befindet.

Hinweis zu verwendeten Abbildungen und Logos

Alle Abbildungen wurden selbst erstellt mit der Software draw.io.

Das Logo der HAW Hamburg [43] sowie das Firmenlogo von CardioSecur [44] wurden von den offiziellen Homepages heruntergeladen und stehen zur freien Verwendung zur Verfügung.

Abbildungsverzeichnis

Literaturverzeichnis

[1] ChatGPT. Zentralen Anforderungen an gelenkte Dokumente laut ISO 13485. https://chatgpt.com/. [Zugriff 04.02.2026].

[2] Lösungsfabrik. Normkapitel 4 - Qualitätsmanagementsystem. https://mpl.loesungsfabrik.de/faq/normkapitel-iso-13485/normkapitel-4-qualitaetsmanagementsystem. [Zugriff 04.02.2026].

[3] git. Install. https://git-scm.com/install/mac. [Zugriff 14.03.2026].

[4] git. Install. https://git-scm.com/install/windows. [Zugriff 15.03.2026].

[5] git. Install. https://git-scm.com/install/linux. [Zugriff 15.03.2026].

[6] IONOS. Intranet: Bedeutung und Einsatzmöglichkeiten. https://www.ionos.de/digitalguide/startup/produktivitaet/intranet-sicherer-interner-datenaustausch/#c552427. [Zugriff 17.03.2026].

[7] git. GUI Clients. https://git-scm.com/tools/guis. [Zugriff 12.03.2026].

[8] GitLab Docs. What is Git?. https://docs.gitlab.com/tutorials/make_first_git_commit/#what-is-git. [Zugriff 23.01.2026].

[9] ChatGPT. Vorteil der Arbeit mit textbasierten Dokumenten in GitLab. https://chatgpt.com/. [Zugriff 23.01.2026].

[10] GitLab Docs. Repository. https://docs.gitlab.com/user/project/repository/. [Zugriff 16.03.2026].

[11] GitLab Docs. Protected branches. https://docs.gitlab.com/18.3/user/project/repository/branches/protected/. [Zugriff 15.03.2026].

[12] GitLab Docs. For all projects in an instance. https://docs.gitlab.com/18.3/user/project/repository/branches/default/#for-all-projects-in-an-instance. [Zugriff 15.03.2026].

[13] GitLab Docs. Branches. https://docs.gitlab.com/user/project/repository/branches/. [Zugriff 16.03.2026].

[14] GitLab Docs. Create a Git branch for your changes. https://docs.gitlab.com/topics/git/branch/. [Zugriff 18.03.2026].

[15] GitLab Docs. Git file history. https://docs.gitlab.com/user/project/repository/files/git_history/. [Zugriff 18.03.2026].

[16] GitLab Docs. Merge request approvals. https://docs.gitlab.com/administration/merge_requests_approvals/. [Zugriff 18.03.2026].

[17] GitLab Docs. Tags. https://docs.gitlab.com/user/project/repository/tags/. [Zugriff 18.03.2026].

[18] GitLab Docs. Archive a project. https://docs.gitlab.com/user/project/working_with_projects/#archive-a-project. [Zugriff 05.03.2026].

[19] GitLab Docs. Get started with GitLab CI/CD. https://docs.gitlab.com/ci/. [Zugriff 18.03.2026].

[20] GitLab Docs. Docker executor. https://docs.gitlab.com/runner/executors/#docker-executor. [Zugriff 18.03.2026].

[21] GitLab Docs. Docker executor. https://docs.gitlab.com/runner/executors/docker/. [Zugriff 15.03.2026].

[22] GitLab Docs. The Shell executor. https://docs.gitlab.com/runner/executors/shell/. [Zugriff 18.03.2026].

[23] GitLab Docs. GitLab container registry. https://docs.gitlab.com/user/packages/container_registry/. [Zugriff 18.03.2026].

[24] GitLab Docs. Roles and permissions. https://docs.gitlab.com/user/permissions/. [Zugriff 18.03.2026].

[25] GitLab Docs. Add users to a project. https://docs.gitlab.com/user/project/members/#add-users-to-a-project. [Zugriff 18.03.2026].

[26] GitLab Docs. Create a group. https://docs.gitlab.com/user/group/#create-a-group. [Zugriff 18.03.2026].

[27] GitLab Docs. Add users to a group. https://docs.gitlab.com/user/group/#add-users-to-a-group. [Zugriff 18.03.2026].

[28] GitLab Docs. Invite a group to a project. https://docs.gitlab.com/user/project/members/sharing_projects_groups/#invite-a-group-to-a-project. [Zugriff 18.03.2026].

[29] GitLab Docs. Set up the integration. https://docs.gitlab.com/user/project/integrations/emails_on_push/#set-up-the-integration. [Zugriff 18.03.2026].

[30] GitLab Docs. Squash and merge. https://docs.gitlab.com/18.3/user/project/merge_requests/squash_and_merge/. [Zugriff 08.03.2026].

[31] GitLab Docs. Merge methods. https://docs.gitlab.com/18.3/user/project/merge_requests/methods/. [Zugriff 08.03.2026].

[32] GitLab Docs. View a file´s Git history. https://docs.gitlab.com/user/project/repository/files/git_history/#view-a-files-git-history. [Zugriff 18.03.2026].

[33] GitLab Docs. Rebase. https://docs.gitlab.com/user/project/merge_requests/conflicts/#rebase. [Zugriff 16.03.2026].

[34] GitLab Docs. Rebasing in (semi-)linear merge methods. https://docs.gitlab.com/user/project/merge_requests/methods/#rebasing-in-semi-linear-merge-methods. [Zugriff 17.03.2026].

[35] qmBase. So meistern Sie die Software Validierung nach ISO 13485. https://www.qmbase.com/software-validierung/. [Zugriff 04.02.2026]

[36] ATLASSIAN. Atlassian Sourcetree. https://stack.watch/product/atlassian/sourcetree/. [Zugriff 11.03.2026].

[37] ChatGPT. Umsetzbare Sicherheitsmaßnahmen bezüglich Sourcetree. https://chatgpt.com/. [Zugriff 11.03.2026].

[38] ChatGPT. Risikoquellen bei der Verwendung von Sourcetree. https://chatgpt.com/. [Zugriff 11.03.2026].

[39] GitLab Docs. Export a project and its data. https://docs.gitlab.com/user/project/settings/import_export/#export-a-project-and-its-data. [Zugriff 18.03.2026].

[40] GitLab Docs. Import a project and its data. https://docs.gitlab.com/user/project/settings/import_export/#import-a-project-and-its-data. [Zugriff 18.03.2026].

[41] GitLab Docs. Backup and restore. https://docs.gitlab.com/18.3/charts/architecture/backup-restore/. [Zugriff 15.03.2026].

[42] ATLASSIAN. Sourcetree Security Advisory 2020-09-02. https://support.atlassian.com/sourcetree/kb/sourcetree-security-advisory-2020-09-02/. [Zugriff am 11.03.2026].

[43] HAW Hamburg Corporate Design. HAW_Marke_CMYK_300dpi.jpg. [JPG]. https://www.haw-hamburg.de/hochschule/hochschuleinheiten/presse-und-kommunikation/corporate-design/. [Zugriff 03.03.2026].

[44] CardioSecur Presse. CardioSecur-Positive.svg. [Microsoft Edge HTML Document]. https://www.cardiosecur.com/meta-navigation/footer/company/press. [Zugriff 03.03.2026].

[45] ChatGPT. Skript-Erstellung des Dockerfiles für die Generierung eines PDF-Dokuments aus einem LaTeX-Dokument in GitLab. https://chatgpt.com/. [Zugriff 18.03.2026].

[46] ChatGPT. Skript-Erstellung der .gitlab-ci.yml-Datei für den Build-Job in GitLab. https://chatgpt.com/. [Zugriff 18.03.2026].

[47] git. 7.11 Git Tools - Submodules. https://git-scm.com/book/en/v2/Git-Tools-Submodules. [Zugriff 15.03.2026].

[48] GitLab Docs. Update a submodule reference. https://docs.gitlab.com/api/repository_submodules/#update-a-submodule-reference. [Zugriff 16.03.2026].

[49] GitLab Docs. Using Git submodules with GitLab CI/CD. https://docs.gitlab.com/18.3/ci/runners/git_submodules/. [Zugriff 16.03.2026].

Anhang

Anhang I: Branch-Regeln

Anleitung für die Einstellungen eines *protected branches* eines Projekts

1. In der Sidebar unter **Search or go to…** das entsprechende Projekt auswählen
2. Auf **Settings → Repository** klicken
3. *Branch Rules* erweitern und bei dem *Main branch* (oder einem anderen beliebigen *Branch*) auf **View details** klicken
4. In dem Abschnitt *Protect branch* bei *Allowed to merge* auf **Edit** klicken
5. In der rechts auftauchenden Sidebar **Developers and Maintainers** auswählen und auf **Save changes** klicken
6. Bei *Allowed to push and merge* auf **Edit** klicken
7. In der rechts auftauchenden Sidebar **No one** auswählen und auf **Save changes** klicken

Anhang II: Docker Container

Anleitung zur Docker-Installation

In der Virtuellen Maschine (z.B. VirtualBox) muss `docker` mit folgender Eingabe in der Konsole bzw. dem Terminal installiert werden:

```
sudo apt install docker.io
```

Anleitung zur Erstellung des Dockerfiles

Dateiname in GitLab: Dockerfile [45]

```
FROM ubuntu:22.04

ENV DEBIAN_FRONTEND=noninteractive

RUN apt-get update && apt-get install -y \
       latexmk \
       texlive-latex-base \
       texlive-latex-recommended \
       texlive-latex-extra \
       texlive-fonts-recommended \
       texlive-fonts-extra \
       texlive-lang-german \

WORKDIR /data
```

`ubuntu:22.04` ist in diesem Fall ein gewähltes Linux-Betriebssystem des erstellten *Containers* mit der Version `22.04`. `ENV DEBIAN_FRONTED=noninteractive` besagt, dass der Benutzer nie Nachfragen von dem Betriebssystem bekommt. `RUN apt-get update` bringt das Paketverzeichnis auf den aktuellen Stand. `apt-get install` installiert die angegebenen Pakete und das `-y` beantwortet alles mit *yes* (ja). Mit `latexmk` werden aus LaTeX-Dokumenten PDF-Dokumente generiert, wofür die anderen angegebenen Pakete notwendig sind. `WORKDIR /data` gibt das Startverzeichnis an, wenn man den Container startet.

Anleitung zur Container Registrierung

Die *Container Registry* ist standardmäßig nicht aktiviert. Sie muss in der GitLab-Konfiguration aktiviert werden und benötigt einen freien Port. Die GitLab-Konfiguration liegt bei Unix-basierten Betriebssystemen wie Linux oder macOS üblicherweise in `/etc/gitlab/gitlab.rb`.

Folgende Einträge sind hinzuzufügen:

```
registry['enable'] = true
registry_external_url 'http://192.168.1.48:5055' # Freier Port, hier 5055
gitlab_rails['registry_enabled'] = true
registry_nginx['enable'] = true
registry_nginx['listen_port'] = 5055
registry_nginx['listen_http'] = true
```

Die IP-Adresse (hier als Beispiel 192.168.1.48) muss mit dem Eintrag in der `external_url`, also der Adresse, unter der GitLab vom Browser aus erreichbar ist, übereinstimmen.

Für das HTTPS-Protokoll müssen ggf. `ssl_certificate` und `ssl_certificate_key` für `registry_nginx` mit denselben Einträgen wie beim Host gesetzt werden.

Nach jeder Änderung der GitLab-Konfiguration muss ein *Reconfigure* und *Restart* gemacht werden.

```
gitlab-ctl reconfigure
gitlab-ctl restart
```

Damit ist die Container Registry aktiviert.

Anleitung zur Erstellung von Docker-Images

Ein *Docker-Image* wird üblicherweise in einer Datei beschrieben, die `Dockerfile` heißt. Befindet sie sich im aktuellen Verzeichnis, kann das *Image* mit

```
docker build -t <registry>/<namespace>/<project>/<image>:<tag> .
```

gebaut werden.

Dabei ist `registry` die URL der *Container Registry* (hier als Beispiel 192.168.1.48:5055). Es folgen Namespace und Projektname. `image` ist eine Bezeichnung für das *Image*. Als `tag` wird häufig „latest“ verwendet. Es ist nur bedingt sinnvoll hier Versionsnummern zu verwenden, denn das *Tag* muss später in der `.gitlab-ci.yml` gewählt werden.

Um das fertig gebaute *Image* hochzuladen, muss der Benutzer sich zunächst bei der *Container Registry* einloggen:

```
docker login <registry> -u <username in GitLab>
```

Jetzt kann das *Image* hochgeladen werden:

```
docker push <registry>/<namespace>/<project>/<image>:<tag>
```

In GitLab sind die *Images*, die für ein Projekt hochgeladen worden sind, unter **Project → Deploy → Container Registry** sichtbar.

Anhang III: E-Mail-Benachrichtigungen

Anleitung Mitarbeiter

1. In der Sidebar oben auf das **Profil-Bild → Edit profile** klicken
2. In der Sidebar auf **Notifications** klicken
3. Als *Global notification email* wäre die Firmen-E-Mail-Adresse sinnvoll zu wählen
4. Das *Global notification level* auf **Custom** stellen, um individuelle Optionen auszuwählen
5. Klick auf die Schaltfläche mit der Glocke
6. Auswahl als Mitarbeiter vornehmen:
   Comment is added, Issue is closed, Issue is reassigned, Issue is reopened, Merge request is closed, Merge request is merged, Merge request is reassigned, Merge request is reopened, Merge request reviewers are changed, Project is moved, Release is created

Anleitung Reviewer

s. Schritte 1 bis 5 der Anleitung für Mitarbeiter

6. Auswahl als Reviewer vornehmen:
   Comment is added, Issue is closed, Issue is reassigned, Issue is reopened, Merge request is created, Merge request is reassigned, Merge request is reopened, Merge request is set to auto-merge, Merge request receives a push, Merge request reviewers are changed, Project is moved, Release is created

Anleitung Geschäftsführung/Admin

s. Schritte 1 bis 5 der Anleitung für Mitarbeiter

6. Auswahl als Geschäftsführung/Admin vornehmen:
   Issue is created, Issue is due tomorrow, Issue is reassigned, Merge request is merged, Pipeline fails, Pipeline is fixed, Project is moved, Release is created

Anleitung für die Benachrichtigung nach einer Tag-Erstellung

1. In der Sidebar unter **Search or go to…** das entsprechende Projekt auswählen
2. In der Sidebar auf **Settings → Integrations** die Integration *Emails on push* auswählen
3. Unter *Enable integration* den Haken bei **Active** setzen
4. Unter *Trigger* den Haken bei **Tag push** setzen
5. In das Eingabefeld *Recipients* die gewünschten Personen (E-Mail-Adresse) eintragen, die über den *Tag push* informiert werden sollen
6. Anschließend können die Einstellungen mit Klick auf die Schaltfläche **Test settings** getestet werden (es wird eine E-Mail mit dem aktuellen Stand des Repos an die eingetragenen Personen versendet)
7. Auf die Schaltfläche **Save changes** klicken und die Integration wird aktiviert.
   Alle Einstellungen können jederzeit beliebig angepasst werden

Anhang IV: Historie

Anleitung für die Einstellungen der Historie in Projekten

Als Vorschlag können in den einzelnen Projekten folgende Einstellungen vorgenommen werden, damit der *Merge-Workflow* bei allen Projekten identisch ist:

1. In der Sidebar unter **Search or go to…** das entsprechende Projekt auswählen
2. In der Sidebar auf **Settings → Merge requests** klicken
3. Unter *Merge method* die Option **Merge commit with semi-linear history** auswählen
4. Unter *Merge options* den Haken bei **Show link to create or view a merge request when pushing from the command line** und **Enable "Delete source branch" option by default** setzen
5. Unter *Squash commits when merging* die Option **Do not allow** auswählen
6. Unter *Merge checks* den Haken bei **Pipelines must succeed** und **All threads must be resolved** setzen
7. Abschließend auf die Schaltfläche **Save changes** klicken

Anhang V: PDF-Dokument generieren

Anleitung zur PDF-Generierung aus einem LaTeX-Dokument

Dateiname in GitLab: `.gitlab-ci.yml` [46]

```
build-pdf:
	tags:
		- docker
	image: <registry>/<namespace>/<project>/<image>:<tag>
	script:
		- latexmk -pdf -interaction=nonstopmode -halt-on-error test.tex
	artifacts:
		paths:
			- test.pdf
```

`build-pdf` ist der Name des *Jobs* (Beschreibung der durchzuführenden Tätigkeit des *Jobs*) und wird als ausgeführter *Job* bei den Artefakten angezeigt. Unter `tags` wird festgelegt, dass `docker` verwendet wird. Das `image` gibt das verwendete *Docker-Image* an. Unter `script` wird die auszuführende Tätigkeit des *Jobs* geschrieben. In diesem Fall soll ein PDF-Dokument (hier: `test.pdf`) aus einem LaTeX-Dokument (hier: `test.tex`) generiert werden. Dabei soll der Name der beiden Dokumente der gleiche sein. `artifacts` gibt an, dass das erstellte PDF-Dokument (hier: `test.pdf`) als Artefakt in GitLab abgelegt werden soll.

Anhang VI: Erstellen eines Repos

Anleitung zur Erstellung eines neuen Projekts/Repos in GitLab

1. Links in der Sidebar auf das „+“ klicken und **New project/repository** auswählen
   (Alternativ: unter dem Reiter *Projects* rechts oben auf die Schaltfläche **New project** klicken)
2. Art des Repos auswählen (leer, Vorlage, Projekt importieren)
   (Bei leerem Repo immer den Haken bei **Initialize repository with a README** setzen)
3. In dem angelegten Projekt in der Sidebar auf **Manage → Members** klicken und Personen oder Personengruppen hinzufügen und deren Rollen sowie zeitliche Zugangsberechtigung festlegen

Anhang VII: Submodules in GitLab

Submodules sind eigenständige Repos, die bestimmte Dateien beinhalten, auf die andere Repos per `.gitsubmodule`-File [47] zugreifen und deren Inhalt in den eigenen Dateien verwenden. Es werden z.B. verschiedene Dateitypen von einem LaTeX-Dokument verwendet, indem u.a. auf Bild-Dateien zugegriffen wird, die in der PDF-Version des LaTeX-Dokuments angezeigt werden sollen. Der genaue Pfad muss korrekt in dem LaTeX-Dokument angegeben werden, da sonst nicht auf den Inhalt des Submodules zugegriffen und kein PDF-Dokument generiert werden kann. Der Vorteil bei der Verwendung von Submodules ist, dass die Dateien, die als Vorlage dienen, nur ein Mal in GitLab vorhanden sind. Somit wird das mehrfache Vorhandensein der gleichen Vorlage-Dateien vermieden. Auch Änderungen an den Vorlage-Dateien sind globaler umzusetzen, da nur ein Mal die Dateien geändert werden müssen. Die Repos, die das Submodule-Repo verwenden, können dann einfach in GitLab aktualisiert werden [48]. Dadurch ist die Aktualisierung der einzelnen LaTeX-Dokumente in verschiedenen Repos effizienter, als wenn die zu ändernden Dateien in jedem Repo einzeln aktualisiert werden müssen. [49]

Anhang VIII: Hinzufügen/Entfernen einer Datei

Anleitung für das Hinzufügen/Entfernen einer Datei zu/aus einem Repo

1. In der Sidebar unter **Search or go to…** das entsprechende Projekt auswählen
2. Auf die Schaltfläche **Code** und bei *Clone with HTTPS* auf **Copy URL** klicken
3. Sourcetree öffnen
4. Auf **Datei → Klonen/Neu…** klicken
5. Die kopierte URL in *Quellpfad/URL* einfügen
   Zielpfad und Name werden automatisch vorausgefüllt, können aber geändert werden
   [Wenn mit Submodules gearbeitet wird, unter „*Erweiterte Optionen*“ den Haken bei „**Untermodule rekursiv**“ setzen (alle verbundenen Submodules werden mit geklont)]
6. Auf **Klone** klicken und ein lokales Repo wird angelegt
7. Auf **Zweig** klicken und einen neuen *Branch* mit sinnvollem Namen erstellen
8. In der Sidebar per Doppelklick zu dem erstellten *Branch* navigieren
9. Auf **Explorer** klicken und der Pfad des lokalen Repos wird geöffnet
10. Eine Datei in das lokale Repo hinzufügen
11. Sourcetree erkennt die Änderung/en automatisch und zeigt die Datei/en in dem Fenster *Nicht vorgemerkte Dateien* an

12. Durch Klick auf die Datei wird der Inhalt daneben angezeigt (bei textbasierten Dokumenten)
13. Mit Klick auf **Alles vormerken**, **Ausgewählte vormerken** oder auf das „+“ einer einzelnen Datei werden die jeweiligen Dateien in *Vorgemerkte Dateien* verschoben
14. Oben links auf die Schaltfläche **Commit** klicken, anschließend eine Beschreibung der Änderungen unten in das Feld eine *Commit-Beschreibung* hinzufügen und unten rechts auf **Commit** klicken
15. Oben auf die Schaltfläche **Push** klicken, den Haken bei dem entsprechenden lokalen *Branch* zum *Pushen* setzen und auf **Push** klicken
    [evtl. Eingabe von Benutzername und Passwort (bzw. andere Authentifizierungsmethode)]
16. Nun ist der neue *Branch* mit den neuen Dateien in GitLab vorhanden (‚aber noch nicht freigegeben)

Anhang IX: Ändern einer vorhandenen Datei im Repo

Anleitung zum Ändern einer vorhandenen Datei in einem Repo

s. Anhang VIII, Schritte 1 bis 9 der Anleitung für das Hinzufügen/Entfernen einer Datei zu/aus einem Repo

10. Die zu ändernde Datei öffnen, die gewünschten Änderungen vornehmen und speichern
11. Sourcetree erkennt automatisch, dass der Inhalt der Datei verändert wurde und zeigt die Datei in dem Fenster *Nicht vorgemerkte Dateien* an
12. Mit Klick auf die Datei werden Differenzen zur vorigen Version der Datei farblich gekennzeichnet (bei textbasierten Dokumenten: grün = neuer Inhalt; rot = vorhandener Inhalt einer Zeile wurde verändert)
13. Mit Klick auf **Alles vormerken**, **Ausgewählte vormerken** oder auf das „+“ einer einzelnen Datei werden die jeweiligen Dateien in *Vorgemerkte Dateien* verschoben
14. Oben links auf die Schaltfläche **Commit** klicken, anschließend unten in das Feld eine *Commit-Beschreibung* hinzufügen und unten rechts auf **Commit** klicken
15. Oben auf die Schaltfläche **Push** klicken, den Haken bei dem entsprechenden lokalen *Branch* zum *Pushen* setzen und auf **Push** klicken
    [evtl. Eingabe von Benutzername und Passwort (bzw. andere Authentifizierungsmethode)]
16. Nun ist der neue *Branch* mit den neuen Dateien in GitLab vorhanden (‚aber noch nicht freigegeben)

Anhang X: Freigabeprozess

Anleitungen zum Freigabeprozess

M: Mitarbeiter, der die Datei bearbeitet; R: Reviewer

Anleitung zur direkten Genehmigung der Änderungen

1. M meldet sich in GitLab an
2. In der Sidebar unter **Search or go to…** das entsprechende Projekt auswählen (Alternativ oben in der Sidebar auf **Merge requests** klicken und rechts oben in dem Drop-Down-Menü **New merge request in …** das entsprechende Projekt auswählen, danach weiter bei Schritt 5)
3. In der Sidebar in dem Projekt auf **Merge requests** klicken (Alternativ unter **Code → Merge requests**)
4. Anschließend auf die Schaltfläche **New merge request** klicken
5. Es werden nun der bearbeitete *Branch* als *Source branch* und der Ziel-Branch als *Target branch* (i.d.R. der *Main branch*) ausgewählt
6. Anschließend auf die Schaltfläche **Compare branches and continue** klicken
7. Auf der nächsten Seite einen Titel bei *title* der MR hinzufügen, den bzw. die Reviewer auswählen und eine *MR-Beschreibung* hinzufügen
8. Anschließend auf **Create merge request** klicken
9. Die MR wurde nun erstellt und die ausgewählten Reviewer bekommen automatisch eine Benachrichtigung per E-Mail
10. R meldet sich in GitLab an
11. In der Sidebar auf die Schaltfläche **Merge requests** klicken
12. Auf der folgenden Ansicht sind alle offenen MRs aufgelistet
13. Mit Klick auf den Titel der entsprechenden MR zur nächsten Ansicht wechseln
14. Der Reiter *Changes* beinhaltet alle Änderungen und macht diese sichtbar. Diese gilt es zu prüfen. An dieser Stelle können einzelne Zeilen eines textbasierten Dokuments oder z.B. ein Bild (z.B. jpg-Datei) kommentiert werden (bei Nicht-Genehmigung erforderlich) (Optional können die *Commits* und *Pipelines* angesehen werden)
15. In dem *Overview* auf die Schaltfläche **Approve** klicken, um sicherzustellen, dass der Inhalt überprüft wurde
16. Anschließend auf die Schaltfläche **Merge** klicken. Dabei die Optionen **Delete source branch** und **Edit commit message** auswählen
17. Die MR ist nun *gemerged*
18. M bekommt eine automatische Benachrichtigung per E-Mail, dass die MR erfolgreich *gemerged* wurde
19. Die neue Version des Repos befindet sich in GitLab und kann heruntergeladen werden. Ein neuer Eintrag in der Historie wurde erstellt

Anleitung zur Nicht-Genehmigung der Änderungen

s. Schritte 1 bis 14 der Anleitung zur direkten Genehmigung der Änderungen

15. Den ersten fertigen Kommentar mit **Start a review** bzw. ab dem zweiten Kommentar mit **Add to review** bestätigen
16. Als gesamtes Review im *Overview* oben rechts auf die Schaltfläche **Your review** klicken und dort **Request changes** auswählen. Dort wird ein Kommentar für die angeforderten Änderungen geschrieben
17. Die angeforderten Änderungen werden mit Klick auf **Submit review** an M per E-Mail geschickt
18. M bearbeitet die angeforderten Änderungen im lokalen Repo
19. M *committet* die Änderungen und *pusht* diese
20. In der MR wird der *Push* erkannt und R klickt im *Overview* unter *Activity* auf **compare with previous version** des getätigten *Commits*
21. R überprüft, ob die geforderten Änderungen wie gewünscht bearbeitet wurden
22. Wenn ja, klickt R bei den einzelnen angeforderten Änderungen auf **Resolve thread**
    [Bei Nicht-Genehmigung werden die Schritte ab Schritt 15 (Nicht-Genehmigung der Änderungen) wiederholt]
23. Nun wird bei Schritt 15 (Direkte Genehmigung der Änderungen) bis zum Ende fortgefahren

Anhang XI: PDF-Dokument zur Verfügung stellen

Anleitung für das zur Verfügung stellen eines PDF-Dokuments

1. In der Sidebar unter **Search or go to…** das entsprechende Projekt auswählen
2. In dem Drop-Down-Menü „+“ auf **New tag** klicken
3. Bei *Tag name* wird ein Name des *Tags* eingetragen (i.d.R. eine Versionsnummer, z.B. V2.0)
4. Bei *Create from* wird der *Branch* ausgewählt, von dem der *Tag* erstellt werden soll
5. Eine *Tag-Beschreibung* hinzufügen, um einen *annotated tag* zu erstellen
6. Auf **Create tag** klicken
7. Nun wird der *Build-Job* von GitLab zur Generierung von PDF-Dokumenten automatisch ausgeführt
8. In dem Projekt in der Sidebar auf **Build → Artifacts** klicken (dort werden alle *Jobs*, die ausgeführt wurden, aufgelistet, egal ob erfolgreich oder nicht)
9. Rechts neben dem entsprechenden erfolgreich ausgeführten *Job* auf die Schaltfläche **Download** klicken
   (vorher kann mit Klick auf die Schaltfläche **Browse** eingesehen werden, welche Dokumente sich in dem ZIP-Archiv befinden)
10. Einen gewünschten Pfad zum Speichern des ZIP-Archivs auswählen und auf **Speichern** klicken
11. Die entsprechenden PDF-Dokumente aus dem ZIP-Archiv an dem gewünschten Ablageort ablegen und evtl. veraltete vorhandene Versionen ersetzen

Hiermit erkläre ich, dass ich die vorliegende Bachelorarbeit selbstständig und ohne fremde Hilfe angefertigt habe. Alle verwendeten Quellen sind ordnungsgemäß angegeben. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ort, Datum Melissa Wilde