Wilhelm Büchner Hochschule
Private Fernhochschule Darmstadt

Fachbereich Informatik
Ostendstraße 3
D-64319 Pfungstadt

Konzeption und Entwicklung einer Versionierungssoftware zur besseren Verwaltung von Macro- und SPS-Programmen als Bestandteil der Firmware von Präzisionsmessmaschinen

Betreuer:	Prof. Dr. Freytag
Autor:	Maik Ledwina
Matrikelnummer:	880767
Anschrift:	Ringstraße 30
	76437 Rastatt
Abgabetermin:	2. Juli 2016

Zusammenfassung

Dies ist die Zusammenfassung.

Inhaltsverzeichnis

1 Einleitung

1.1 Vorwort

Softwareentwicklung, ob für Anwendungen oder Firmeware, unterliegt einer stetigen Evolution. Neues kommt hinzu, bestehendes wird erweitert oder verbessert und Bugs werden beseitigt. Bei Software ist der Rhythmus, in welchem die Neuerungen und Erweiterungen zum Kunden kommen, meist durch einen vom Hersteller geplanten Produktlebenszyklus bestimmt. Nur Bugs werden in kürzeren Abständen durch Updates behoben. Bei der Firmware für Controller ist dies häufig anders, hier bestimmen die Anforderungen von Hardwareentwicklung und Kundenwünschen die Veränderungen und Entwicklung. Wird ein Problem bei einem Kunden festgestellt, werden Fehlerbehebungen nicht mit Updates innerhalb von Wochen sondern von Stunden benötigt bzw. erwartet. Denn das Gerät, dass durch den Controller betrieben wird, muss weiter arbeiten.

1.2 Untersuchungsgegenstand

Der Untersuchungsgegenstand dieser Diplomarbeit ist die Firmware eines Präzisionsmessmaschinencontroller und die mit ihr arbeitenden Macros und SPS Programme.

1.3 Ausgangspunkt

Die Software für den Controller wurde in einem kleinen Entwicklerteam entwickelt und über Jahre hinweg immer weiter ausgebaut und verbessert. Änderungen werden stets direkt integriert und den Kundenwünschen und Anforderungen angepasst. Bugs werden so schnell wie möglich behoben und die Lösung bei den betroffenen Kunden integriert. Jeder der Entwickler hat seinen festen Aufgabenbereich und damit auch das wissen über alle Neuerungen und Fehlerbeseitigungen. Bei Überschneidungen wird zusammengearbeitet jedoch jeder in seinem Aufgabengebiet. Dadurch wurde die Dokumentation des häufigeren auch etwas stiefmütterlich behandelt und es ist teilweise auch schwierig bis überhaupt nicht nachzuvollziehen, welche Änderungen wann integriert wurden bzw. welcher Kunde dieser bereits erhalten hat.

1.4 Aufbau

Abkürzungsverzeichnis

bzw.	Beziehungsweise
evtl.	eventuell
z.b.	zum Beispiel
SCCS	Source Code Control System
RCS	Revision Control System
CVS	Concurrent Version System
SVN	Subversion
VSS	Visual Source Safe
CAA	Computer Aided Accuracy
SPS	Sepicher Programmierbare Steuerung
UML	Unified Modelling Language
EPK	Ereignis gesteuerte Prozesskette
eEPK	erweiterte Ereignis gesteuerte Prozesskette
ER	Entity-Relationship
BPMN	Business Process Modelling Notation
FSFS	File System - File System
DB	Datenbank

Bezeichnungen und Begrifflichkeiten

Repository	Als Repository wird in der Versionsverwaltung, der Ablageort bezeichnet, an welchem die durch das System versionierten Projekte gespeichert werden. Ein Repository kann dabei mehrere Projekte enthalten.
Head	Als Head wird die aktuellste Version im Trunk Zweig des Repository bezeichnet, die im Normalfall mittels Checkout oder update in die lokale Arbeitskopie überführt wird)
checkout	Der Kopiervorgang zum erstellen einer lokalen Arbeitskopie, wird als checkout bezeichnet.
branch	Ein Branch ist ein Zweig der parallel zum Trunk Zweig geführt und gepflegt wird. Dafür kann es verschiedene Gründe geben, evtl. soll die Software für unterschiedliche Betriebssysteme verfügbar sein und hat dabei einen großen Teil an identischen Dateien und einige Betriebssystem spezifische oder weil eine grundlegende Überarbeitung stattfinden soll, welche die parallel weiterlaufende Entwicklung nicht beeinflussen soll.

Mergen	Damit wird das zusammenführen zweier gleicher Dateien mit unterschiedlichen Ständen gemeint. Der Vorgang kann automatisch erfolgen, wenn die Dateien an unterschiedlichen Stellen verändert wurden. Wurden beide an gleicher Stelle geändert, so kommt es zu einem Konflikt der Vorgang wird abgebrochen und eine Fehlermeldung ausgegeben, wo es zu einem Konflikt in der Zusammenführung kam. So dass der Entwickler den Konflikt von Hand beseitigen kann.
Revision	Jede Änderung, die in das Repository commited wird, erzeugt eine neue Revision. Dabei ist es unerheblich, ob diese Datei in den Trunk, einen Branch oder in ein anderes Projekt im gleichen Repository übertragen wird. Bei Subversion gibt es keine Revisionsnummern für die einzelnen Dateien sondern nur für das Repository im Gesamten.
Tag	Ein Tag wird dazu verwendet um bestimmte Revisionen des Repository mit einem speziellen Namen zu versehen. Diese Funktion kann man zum Beispiel dafür verwenden, wenn zum Ausliefern eines Produktes immer eine bestimmte Version erzeugt werden soll, bei dem der Namen nicht einfach nur eine Nummer einer Revision sein soll sondern z. B. einen ßprechenden"Namen erhalten soll wie Version 1.0 oder ähnliches.
commit	Bei einem commit, werden lokale Änderungen an das Repository übertragen und erzeugen dort eine neue Revision. Der Entwickler, der den commit ausführt muss noch eine Commit Message eingeben, die die durchgeführten Änderungen dokumentieren sollen.
Trunk	Als Trunk wird der Hauptentwicklungszwei eines Projekts bezeichnet, welches in einem Repository verwaltet wird.
Update	Mittels dem Befehl Update, kann man sich seine working copy aktuell halten. Der Befehl vergleicht die lokale Kopie mit dem Head im Repository und führt bei den veralteten Dateien ein update durch. Sollte eine Datei auf dem Server erneuert worden sein, die auf der lokalen Kopie auch verändert wurde, so versucht das System die beiden Dateien zusammen zu führen.

Push	text
Klonen	tex)
Open-Source	Tex)
Stakeholder	Als Stakeholder bezeichnet man die Personen oder Personenkreise, die beim Erheben von Anforderungen ein direktes oder indirektes Interesse an der Lösung haben.

2 theoretischer Teil

2.1 Versionsverwaltung

2.1.1 Anfänge der Versionsverwaltung

Die Anfänge der Versionsverwaltung reichen bis in die 70er Jahre des vergangenen Jahrhunderts zurück. Anfang der 70er wurde bei dem amerikanischen Telekommunikationsunternehmen AT & T eines der ersten Versionsmanagement System entwickelt, Source Code Control System. Es war für die Verwaltung einzelner Dateien ausgelegt, die nur von einem Entwickler bearbeitet wurden. Ganze Projekte und Entwicklung im Team konnte mit diesem System nicht verwaltet werden. Etwa 10 Jahre später wurde an der Purdue University durch Walter F. Tichy das Versionierungstool RCS entwickelt. Dabei handelte es sich wieder um ein Dateien verwaltendes Versionierungsprogramm. Dies war bereits für die Entwicklung im Team ausgelegt, vom Aufbau her jedoch so, dass wenn ein Entwickler an einer Datei gearbeitet hat diese für den Schreibzugriff durch andere Entwickler blockiert war. Beide Systeme hatten gemein, dass sie für die Versionsverwaltung lokaler Dateien waren. Ein Jahr nach der Entwicklung von RCS wurde mit der Entwicklung von CVS (1986) begonnen. Ursprünglich als Servererweiterung für RCS gedacht, wurde das zuerst auf UNIX Shell-Skripten basierende System im Jahre 1989 in C neu programmiert. CVS verfolgte einen anderen Ansatz als seine Vorgänger, die Dateien und Projekte waren zentral auf einem Server abgelegt und die einzelnen Entwickler konnten sich ihre Arbeitskopien von der herunter laden. So konnten

auch mehrere Entwickler an der gleichen Datei arbeiten. Auch war CVS in der Lage Verzeichnisstrukturen, wenn auch noch sehr eingeschränkt, zu verwalten. Viele der später entwickelten System bauen auf diesen Systemen auf.

2.1.2 Grundlagen der Versionsverwaltung

Der Ursprüngliche Gedanke der Versionsverwaltung war, dass man damit erfolgte Änderungen in unterschiedlichen Versionen sichtbar und differenzierbar machen konnte. Es sollte möglich sein eine Historie der erfolgten Änderungen abzubilden und für den/die Entwickler sichtbar zu machen. Noch größere Bedeutung bekam das ganze als nicht mehr nur ein Entwickler ein Thema bearbeitete sondern mehrere Entwickler im Team arbeiteten. Die Entstehung von Open-Source-Projekten mit weltweit verteilten Entwicklern hat die Entwicklung entsprechender Versionierungstools beschleunigt und deren Funktionsumfang und Herangehensweise maßgeblich beeinflusst. Die Funktionsweise der Systeme lässt sich in Lokale, Zentrale und Verteilte Versionsverwaltung unterteilen.

- Lokale Versionsverwaltung
  Bei der Lokalen Versionsverwaltung waren die zu versionierenden Dateien lokal auf dem Rechner des Entwickler abgelegt. In diesen Bereich fallen auch nur die beiden zuerst entwickelten Systeme SCCS und RCS.

- Zentrale Versionsverwaltung
  Bei der Zentralen Versionsverwaltung werden die Daten bzw. Repositorien auf einem zentralen Server abgelegt und die Entwickler können sich von diesem Server eine lokale Arbeitskopie herunter laden und nach dem Bearbeiten wieder zurück spielen. Die Versionshistorie ist nur auf dem Server ersichtlich und wenn man eine Information aus einer früheren Version benötigt ist zwingend eine Verbindung zum Server notwendig. In diesen Bereich fallen zwei der am bekanntesten Open-Source-Systeme CVS und Subversion. Aber auch viele der kommerziellen Lösungen, die auf dem Markt sind, beruhen auf die-

sem System.

- Verteilte Versionsverwaltung
  Bei der Verteilten Versionsverwaltung besitzt jeder Entwickler lokal ein eigenes Repository des Projekts an dem er arbeitet. Die einzelnen Repository werden zwischen den unterschiedlichen Entwicklern immer wieder synchronisiert und somit alle Änderungen verteilt bzw. zusammengeführt. Ein zentraler Server ist hier nicht notwendig. Sehr häufig wird jedoch ein zentrales Repository eingesetzt um von dort aus das Produktivsystem zu bilden oder neuen Entwicklern einen Zentralen Punkt zu schaffen an dem sie sich ihre erste Arbeitskopie herunterladen können. Bei diesem System hat jeder Entwickler immer die komplette Versionshistorie mit in seiner Kopie des Repository und kann evtl. später entdeckte Bugs lokal in den früheren Versionen suchen und beheben.

Bei den unterschiedlichen Versionierungssystemen, gibt es auch unterschiedliche Konzepte, was die Arbeitsweise angeht. Man unterscheidet hier zwischen:

- Lock Modify Write
  Bei dieser Arbeitsweise, erstellt sich der Entwickler eine Arbeitskopie der Datei, welche er ändern möchte. Das System sperrt in dieser Zeit die Datei für Schreibzugriffe anderer Entwickler. So können von einer Datei nie zwei unterschiedliche Versionen entstehen und auch keine Konflikte beim Zusammenführen der Dateien. Der Nachteil daran ist, dass nur ein Entwickler an der Datei arbeiten kann und ein anderer, der evtl. dringende Änderungen einfügen muss, dies erst tun kann, wenn der erste Entwickler die Datei wieder freigegeben hat. Ein weiterer Nachteil, wenn eine Datei gesperrt war und es bei dem Entwickler der die Datei gesperrt hatte, einen Systemverlust gab, war die Sperre noch immer vorhanden und musste umständlich entfernt werden. Bekannteste Vertreter für diese Arbeitsweise sind RCS und VSS von Microsoft. Bei den Verteilten Versionsverwaltung ist diese Arbeitsweise ausgeschlossen.

- Copy Modify Merge
  Bei dieser Arbeitsweise, erstellt sich jeder Entwickler seine lokale Arbeitskopie und entwickelt in dieser, hat er einen Teil fertig gestellt und will diesen den anderen zur Verfügung stellen oder ihn in das Haupt Repository übertragen, führt er einen Merge aus. Das heißt, die von ihm geänderte Datei wird auf den Server überspielt. Sollte auf dem Server eine andere Dateiversion verfügbar sein, als die Basis auf der der Entwickler seine Kopie herunter geladen hatte, so entsteht ein Konflikt. Je nach Versionierungssystem, kann dieses Konflikte zum teil automatisch lösen. Kann ein System die wird nun nach einer gemeinsamen Basis beider Dateien gesucht und überprüft ob es die beiden Dateien zusammenfügen kann. Wurde der Code beider Dateien an unterschiedlichen stellen geändert so können diese System den Konflikt durch zusammenführen beider Dateien selbständig beheben. Wurden die gleichen Stellen im Code geändert, so muss dies manuell durch einen der beteiligten Entwickler behoben werden.

2.2 Stand der Technik

Die beiden aktuell am verbreitetsten Systeme sind Subversion und Git. Als Hauptgründe hierfür kann man sehen, dass beide einen sehr großen Umfang an Möglichkeiten und Flexibilität mitbringen und dadurch eines der beiden für die meisten aller zu versionierenden Projekte passend ist. Aber beiden haben ihre weite Verbreitung auch dadurch erfahren, dass sie Open-Source Projekte sind und beide bei großen Open-Source Projekten zum Einsatz kommen.

2.2.1 Subversion

Geschichtliches zu Subversion

Subversion wurde als Nachfolger für CVS, das bis dahin am weites verbreitetste Versionskontrollsystem, nicht zuletzt wegen seiner freien Verfügbarkeit und entsprechendem Einsatz in Open-Source-Projekten, entwickelt. Im Jahr 2000 war es die amerikanische Softwarefirma CollabNet, welche bis dahin selbst CVS im Einsatz hatte. CVS hatte jedoch einige Schwachstellen, was CollabNet dazu veranlasste Kontakt zu Karl Fogel aufzunehmen und ihn zur Mitarbeit an einem neue System zu bewegen. Karl Fogel selbst hatte nicht nur ein Buch[1] über CVS geschrieben sondern mit seinem Freund Jim Blandy hatte er auch eine CSV Beratungs Firma gegründet. Diese war zwar bereits verkauft doch beide arbeiteten noch immer mit CVS und hatten ebenfalls schon über einen Nachfolger gesprochen und sowohl einen Namen dafür wie auch den Entwurf für das Datenablagesystem, es hieß Subversion. Im Mai 2000 war Karl Fogel angestellter bei CollabNet, zusammen mit einigen anderen Entwicklern starteten Sie mit der Entwicklung von Subversion. Es fand sich schnell Unterstützung, da viele Entwickler die gleichen Probleme mit CVS hatten und eine alternative suchten.

Die ersten 14 Monate lang wurde die Softwareentwicklung mit CVS verwaltet, danach waren sie mit der Entwicklung von Subversion soweit fortgeschritten, dass die Versionsverwaltung der Weiterentwicklung von Subversion auf Subversion selbst verwaltet werden konnte.

Subversion ist kein revolutionär neues Versionsverwaltungssystem sondern vielmehr eine Evolutionäre Weiterentwicklung von CVS, was auch den Umstieg von CVS Usern zu Subversion erleichtern sollte.

[1] Karl Fogel und Moshe Bar. *Open Source-Projekte mit CVS : [verteilte Softwareentwicklung mit dem Concurrent Versions System; Grundprinzipien und Anwendung verteilter Softwareentwicklung; CVS-Server installieren und verwalten; jetzt mit umfassender Befehlsreferenz und praktischer Schnellreferenz]*. 2., erw. u. überarb. Aufl. Bonn: MITP-Verl., 2002. ISBN: 3-8266-0816-X.

Obwohl Subversion ein Open-Source-Projekt ist, finanziert CollabNet noch immer einen Großteil dessen Entwicklung und auch einige Vollzeitentwickler des Systems und hat 2009 auch alles daran gesetzt, Subversion in die Familie der Apache Software Foundation (ASF) zu integrieren. Dies gelang im Jahr 2010 auch und aus Subversion wurde Apache Subversion.

Aufbau

Subversion gehört zu den zentralen Versionsverwaltungssystemen. Das heißt es liegt ein Repository auf einem zentralen Server und auf dieses können der oder die Entwickler zugreifen.

Auf dem Server werden die Daten des Repository entweder in einer Berkeley Datenbank oder dem Dateisystem FSFS (FileSystem-FileSystem). Dabei ist die Berkeley DB Data Store die Speicherlösung, mit der man seit beginn der Entwicklung von Subversion arbeitet. Es wurde seinerzeit ausgewählt, da es unter anderem ein stabiles und zuverlässiges System war, welches ebenfalls eine Open-Source-Lizenz hat und daher auch noch kostenlos war. Das FSFS Dateisystem wurde erst später mit der Version 1.1 eingeführt. Seit der Version 1.8 wird die Berkeley DB als veraltet betrachtet und FSFS als alleiniges Speichersystem angenommen.

Verwendet man Subversion alleine auf einem lokalen Rechner, so ist keine Subversion Server notwendig. Sollen jedoch mehrere Entwickler über das Netzwerk auf das Repository zugriff haben, kommt man um einen Server nicht herum. Dabei sind auch hier wieder mehrere Möglichkeiten gegeben. Zum einen lässt sich auf dem Server ein Apache HTTP-Server betreiben und dann mittels HTTP Protokoll über das Netzwerk auf das Repository zugreifen. Zum anderen gibt es einen Subversion eigenen Server, den svnserve und ein dafür entwickeltest SVN Protokoll.

Über die Kommandozeile oder verschiedene Client Anwendungen, lässt sich so mit dem Versionierungssystem arbeiten.

Die Arbeit mit Subversion

Die einfachste Erklärung zur Arbeit mit Subversion ist, der Entwickler der mit dem Projekt arbeiten möchte, erzeugt sich eine lokale Kopie, die working copy genannt wird vom, auf dem Server liegenden Stammprojekt, welches als Trunk bezeichnet wird.Der Kopiervorgang wird als Checkout bezeichnet. Mit der lokalen Kopie arbeitet der Entwickler, hat er eine Änderungen oder Erweiterung fertig so führt er einen commit aus, bei dem alle geänderten Dateien in das Repository übertragen werden und dort für alle anderen zugänglich sind.

So einfach ist es in der Realität kaum und daher existieren noch viele weitere wichtige Funktionen, welche für die tägliche Arbeit mit Subversion benötigt werden.

- checkout
- repository
- head
- branch
- merge
- revision
- tag
- commit
- update
- trunk

Subversion wurde so entwickelt, dass damit mehrere Entwickler am gleichen Projekt arbeiten können. Wenn natürlich mehrere Entwickler an einem Projekt arbeiten, werden auch von mehreren die gleichen Dateien bearbeitet. Da jeder mit sei-

ner lokalen Kopie arbeitet, stellt dies auch kein Problem dar. Werden aber nun die Änderungen von jedem Entwickler zurück gespielt, so haben die Dateien unterschiedliche Stände. Haben alle Entwickler an unterschiedlichen Stellen einer Datei Änderungen durchgeführt, so führt der Server die Änderungen zusammen. Dieser Vorgang nennt sich mergen. Wurden die Dateien jedoch an gleicher Stelle geändert, so entsteht ein Konflikt, der Commit wird mit einer Fehlermeldung abgebrochen. Der Entwickler muss nun schauen, was die Unterschiede beider Versionen ist und diesen Konflikt beheben, es kann erforderlich sein, dass er sich mit dem anderen Entwickler in Verbindung setzt, der die Datei vor Ihm bearbeitet hat. Wurde der Konflikt behoben, muss der Commit erneut durchgeführt werden um die Dateien im Repository zu aktualisieren.

In bestimmten Fällen, kann es sein, dass Dateien nicht von mehreren Entwicklern verändert werden können. Bei Bilddateien ist dies zum Beispiel der Fall. Bei diesen Dateien gibt es keinen lesbaren Code, den das System auf Gleichheit direkt vergleichen kann bzw. entscheiden kann, ob er die Änderungen zusammenführen kann. Hier muss die ausgelagerte Datei vor der Veränderung gesperrt werden, so dass in dieser Zeit kein anderer die Datei verändern kann. Der Entwickler, der die Datei gesperrt hat, muss diese nach Bearbeitung aber wider freigeben, da er sonst die Arbeit anderer Entwickler blockieren kann.

Werden Entwicklungsarbeiten durchgeführt, welche einer längeren Entwicklungszeit bedürfen, so empfiehlt es sich, dies in einem Branch im Repository, parallel zum eigentlichen Entwicklungszweig zu tun. Dadurch hat man die Möglichkeit, das andere die Fortschritte der Entwicklung testen können, diese aber nicht direkt in den Hauptentwicklungszweig integriert sind, für den Fall, dass sie Fehler oder Inkompatibilität erzeugen. Subversion an sich unterstützt die Entwicklung in mehreren Zweigen nicht. Für Subversion handelt es sich dabei eigentlich um eine Kopie des Trunk. Hierfür werden die Dateien nicht wirklich in ein neues Verzeichnis auf dem Server kopiert. Es werden im ersten Schritt nur Verknüpfungen zu den im Trunk befindlichen Dateien erzeugt und nur bei erfolgten Änderungen wird die Datei im neuen Verzeichnis erzeugt. Dieses Vorgehen spart Speicherplatz. Dies

wird gerne als *"billige Kopie"*[2] bezeichnet.

Mit der Update Funktion sollte man seine Kopie, egal ob in einem Branch oder die lokale Working Copie, immer recht aktuell halten. So hat man immer einen Überblick darüber, ob Neuerungen anderer Entwickler die eigenen beeinflussen oder umgekehrt. Es kann sonst dazu führen, dass sich die Entwicklungen zu weit von einander weg entwickeln und dann wird das Zusammenführen schwierig, wenn es zu einer Vielzahl von Konflikten kommt.

Es empfiehlt sich, immer wenn eine für den Kunden bestimmte Revision erzeugt wurde, diese zusätzlich mit einem Tag zu versehen. Dadurch wird dieser Entwicklungsstand noch einmal gesichert und man erhält die Möglichkeit diesem einen Namen zu geben, der nicht nur eine Revisionsnummer wie z. B. 3578 sondern Version 1.9 ist. Zusätzlich wird dann dieser Tag, ähnlich wie bei einem Branch, als Kopie im Repository gespeichert und ist so für spätere Zugriffe einfacher zu finden. Man kann diesen dann auch als Ausgangspunkt verwenden, wenn z.B. ein Fehler in der Software entdeckt wurde und für diese Auslieferungsversion ein passender Patch oder Release bereit gestellt werden soll.

2.2.2 Git

Geschichtliches zu Git

Die Geburtsstunde von Git war im Jahre 2005 und ergab sich aus der Not heraus. Linus Torvalds, der Erfinder von Linux benötigte für die Entwicklung von Linux ein neues Versionierungssystem. Von 2002-2005 setzte man auf BitKeeper, welches eigentlich ein kommerzielles System ist, für die Entwicklung von Linux jedoch kostenfrei war. 2005 kam es aber zu Problemen, da ein Entwickler von

[2] Ben Collins-Sussman, Brian W. Fitzpatrick und C. Michael Pilato. *Versionskontrolle mit Subversion : Software-Projekte intelligent koordinieren*. 2. Aufl. Beijing: OReilly, 2006. ISBN: 3-89721-460-1; 978-3-89721-460-6, S. 57.

BitKeeper die Linux Entwickler beschuldigte, dass sie durch Reverse Engineering versuchen würden die Mechanismen von BitKeeper offen legen wollen.[3] Daraufhin wurde die Erlaubnis BitKeeper kostenfrei nutzen zu dürfen widerrufen. Auf dem Markt gabe es jedoch nichts, was den Ansprüchen von Linus Torvalds genügte und so beschloss er, dass eine Eigenentwicklung her musste. Für diese gab es einige Grundlegende Anforderungen:[4]

- Geschwindigkeit
- Einfaches Design
- Gute Unterstützung von nicht-linearer Entwicklung (tausende paralleler Branches, d.h. verschiedener Verzweigungen der Versionen)
- Vollständig verteilt
- Fähig, große Projekte wie den Linux Kernel effektiv zu verwalten (Geschwindigkeit und Datenumfang)

[3] **Git**.
[4] **PGit**.

2.3 Grundlagen zum untersuchten Einsatzgebiet

2.3.1 Päzisionsmessmaschinen

Strenggenommen handelt es sich dabei eigentlich um Präzisionsmessgeräte in der Umgangssprache hat sich jedoch die Bezeichnung als Maschine durchgesetzt und wird fast einheitlich von allen Herstellern verwendet. Zum Einsatz kommen Präzisionsmessmaschinen überall dort wo an Produkte ein hohes Maß an mechanischer Qualität gefordert ist. So werden mit Präzisionsmessmaschinen alle möglichen gefertigten Bauteile, von kleine Schrauben und Zahnrädern bis hin zu Kompletten LKW, Schiffsmotoren oder Teile von Windkraftanlagen vermessen. Üblicherweise werden Präzisionsmessmaschinen in die Gruppen Portal-, Brücken- und Ständermessmaschinen unterteilt. Die einzelnen Achsen werden wahlweise Rollen- oder, wenn größere Präzision gefragt ist, Luftgelagert. Der Typische Messbereich solcher Maschinen erstreckt sich im Bereich der

- Zahnradmesstechnik
  bei einem Zylindrischen Messbereich mit Durchmessern von ø280mm bis ø4000mm

  und

- Koordinatenmesstechnik
  bei einem Quaderförmigen Messbereich von 500mm x 500mm x 400mm bis 4000mm x 16000mm x 3000mm in Portal- und Brückenbauweise und 1000mm x 600mm x 1000mm bis 56000mm x 3600mm x 3600mm in Ständerbauweise.

Von Präzisionsmessmaschinen spricht man, da die Genauigkeit mit der die Geräte messen können bei einem Maschinengrundfehler von 1µ und teilweise weniger

beginnen.

2.3.2 Controller

Als Controller wird im Allgemeinen eine Steuerung bezeichnet, welche in der Lage ist, durch integrierte oder dazugehörende Firmware oder Software, eine Maschine zu steuern und zu regeln. Im Falle des Untersuchungsgegenstand sind ein hohes Maß an Genauigkeit, Zuverlässigkeit und Qualität gefordert. Messmaschinen müssen, Positionen auf Zehntel Mikrometer halten, verfahren und positionieren können. Jede Präzisionsmessmaschine hat zusätzlich zur hohen Fertigungsgenauigkeit noch ein CAA Feld, welches die mechanischen Restfehler der Maschine enthält und entsprechend vom Controller ausgeglichen werden muss. An allen Achsen der Messmaschine befinden sich Positionsmesssysteme und an den Motoren der Achsen zusätzliche Tachometer. An jeder Messmaschine werden Messköpfe zur Aufnahme von Tastpunkten angebracht. Aufgabe des Controllers ist es diese Komponenten zusammen zu betreiben, synchronisieren und zu überwachen.

2.3.3 SPS-Programme

Speicherprogrammierbare Steuerungen finden in der Steuerung von in der Industrie betriebenen Fertigungsmaschinen eine weite Verbreitung, sie ermöglichen es viele Sensoren zu überwachen und die Maschine entsprechend zu steuern. Im Minimalfall enthält eine SPS-Steuerung Eingänge, Ausgänge, einen Mikrocontroller und Speicher. Für die Steuerungen werden Programme geschrieben und integriert, welche die über die Eingänge ankommenden Informationen verarbeiten und entsprechende Aktionen an die Ausgänge weitergeben. Im Untersuchungsgegenstand, ist eine Software simulierte SPS integriert und dazugehörend eine Vielzahl von SPS-Programmen die den Controller in der Verarbeitung verschiedenster Sensoren unterstützen und die Ergebnisse und Informationen an den Controller weiter geben. Hierdurch ist es möglich, den Controller die Maschinen zu erweitern und

um die beim Kunden geforderten Erweiterungen anzupassen. Dabei kann es sich um Erweiterungen wie die Überwachung einer automatischen Bestückung, Überwachung zusätzlicher Sicherheitseinrichtungen in oder um die Maschinen oder die Ansteuerung zusätzlicher Komponenten für die fertigungsintegrierte Werkstückprüfung handeln.

2.3.4 Makros

Zur Software des Controller im Untersuchungsgegenstand gehören eine Vielzahl von Makros, welche als Erweiterungen bzw. zusätzliche Optionen zur Software des Controller gehören und welche mit in die Versionsverwaltung integriert werden müssen. Die Makros sind vom Controller ausführbare Dateien, welche Funktionen die im Controller enthalten sind zu bestimmten Abläufen zusammenfassen oder auch weitere Funktionen enthalten um den Funktionsumfang des Controllers zu erweitern.

2.4 Besonderheiten der Industriellen Fertigung

Durch den Einsatz der Messmaschinen in der industriellen Produktion wird an diese und an die Software, welche diese betreibt einiges an Anforderungen gestellt, welcher Software, die im Bereich einer Verwaltung oder Prozessteuerung eingesetzt wird nicht unbedingt unterliegt.

2.4.1 Rückführbarkeit

Messgeräte haben die Vorgabe, dass die mit Ihr ermittelten Messergebnisse Rückführbar sind. Das heißt, die Anbieter von Messgeräte sind dazu verpflichtet, die Messgeräte nach bestimmten Normen zu prüfen und müssen dafür speziell kalibrierte Prüfmittel verwenden und die in Software vorkommenden Berechnungs- und Filtermethoden müssen entsprechend auch zertifiziert sein. In der Software des Controller, sind ebenfalls Berechnungsmethoden z.B. nach Gauss und Filter, für die Filterung von Messpunkten enthalten. Zusätzlich müssen für die Fehlerkorrektur der Maschine (CAA), entsprechende Modelle berechnet werden und diese über die Abnahmen der Messgeräte verifiziert werden.

2.4.2 Anforderung an die Wartbarkeit der Systeme

Bei Messgeräten handelt es sich nicht um kleine Investitionen für ein Unternehmen. Entsprechend sind diese auch immer über recht lange Zeiträume im Einsatz. Dabei sind 20 und mehr Jahre keine Seltenheit. Nach Investitionen für bestehende Maschinen sind immer schwierig, daher muss es mit den Systemen möglich sein, auch Updates und Erweiterungen integrieren zu können ohne das weitere Investitionen getätigt werden müssen. Dies birgt im Umkehrschluss jedoch auch das Risiko, dass man über einen sehr langen Zeitraum die Systeme abwärtskompatibel gestalten muss. So ist es keine Seltenheit, das man bei Kunden noch PC-Systeme

mit Windows 98, 2000 oder NT antrifft. Diese Systeme müssen jedoch auch weiterhin gepflegt und supportet werden, soweit dies möglich ist. Sollten Hardwarekomponenten des Rechners getauscht werden müssen, ist ein Upgrade auf ein aktuelles System unumgänglich. In diesem Fall ist mit aktuell jedoch nicht immer das neuste zur Verfügung stehende Betriebssystem gemeint. Als Lieferant von Maschinen, muss man sich immer auch an sehr viele Vorgaben seitens der Kunden halten. So hat zum Beispiel das derzeit aktuelle Betriebssystem Windows 10 von Microsoft in der Industrie noch nicht wirklich Einzug erhalten. Dies ist keine Seltenheit, einige Betriebssysteme haben es in ihrem kompletten Produktlebenszyklus nie wirklich oder nur sehr vereinzelt in Industriebetriebe geschafft.

2.4.3 Reaktionszeiten im Fehlerfall

Standzeiten von Maschinen kosten die betreibenden Unternehmen viel Geld. Daher gibt es an Lieferanten der fertigenden Industrie, sei es als Zulieferer oder als Maschinenlieferant immer die Anforderung, dass Stillstandszeiten so gering wie irgend möglich gehalten werden. So sind Verträge, in denen Maschinenlieferanten Forderungen von Reaktionszeiten innerhalb 12h und Reparaturzeiten von maximal 24h keine Seltenheit. Größere Betreiber legen sich auch gerne verschiedene Komponenten auf Lager um im Bedarfsfall schneller reagieren zu können. Gleichzeitig ist der Lieferant gefordert, Servicepersonal bereitstellen zu können, die entsprechend schnell agieren können. Datensicherungen der beim Kunden installierten Controller Versionen mit allen Konfigurations- und Korrekturdaten sind daher auch beim Lieferant gespeichert, um evtl. neu benötigte Rechnersysteme schneller aufgesetzt und Einsatzbereit zu bekommen.

2.5 Anforderungsmanagement

Das Anforderungsmanagement, ist eine Management Disziplin die das Software Engineering unterstützt. Es setzt sich damit auseinander eine optimale Lösung durch strukturierte und organisierte Planung und Definition der Anforderung zu erreichen.

Durch das Anforderungsmanagement soll vermieden werden, dass durch schlechte, unzureichende oder missverständliche Definitionen, eine Entwicklung am Ende nicht dem Entspricht, was der Auftraggeber haben wollte.

Das Anforderungsmanagement lässt sich in unterschiedliche Arbeitspakete unterteilen, hierzu lassen sich in der Literatur die unterschiedliche Ausprägungen und Anzahlen finden. Bei Marcus Grande[5] wird in vier Haupttätigkeiten unterteilt. Für die Diplomarbeit habe ich mir beispielhaft die Punkte ausgesucht, wie sie im Buch Änforderungsmanagement in sieben Tagen"[6] zu finden sind.

2.5.1 Arbeitspakete im Anforderungsmanagement

- Anforderungsmanagement planen
  Das Anforderungsmanagement dient der Planung und so ist es auch nicht weiter verwunderlich, dass auch in der Vorbereitung eine Planung für das Anforderungsmanagement sinnvoll ist. Es stehen eine Vielzahl von Tools, welche das Anforderungsmanagement unterstützen, die Art und weise der Dokumentation sollte festgelegt werden und die Anforderungsattribute sollten festgelegt werden.

- Ausgangspunkt finden

[5] Marcus Grande. *100 Minuten für Anforderungsmanagement : Kompaktes Wissen nicht nur für Projektleiter und Entwickler*. 2., aktualisierte Aufl. 2014. Springer Vieweg, 2014, S. 10.
[6] Thomas Niebisch. *Anforderungsmanagement in sieben Tagen*. Springer-Verlag, 2013, S. 30-31.

Um das Ziel zu finden, sollte man wissen wo man los geht
Bevor man mit der Zielfindung beginnen kann, sollte man wissen, wie der aktuelle Stand ist. Man kann das mit einem Orientierungsmarsch bei der Bundeswehr vergleichen, bei dem Soldaten mit einem Kompass und einer Geländekarte ein Ziel erreichen müssen, wenn sie nicht wissen wo sie los gehen, ist es unwahrscheinlich, dass sie das ausgegebene Ziel erreichen. Wie schaut der zu bearbeitende Prozess aus und wer ist darin Akteur und/oder Betroffener. Durch das ermitteln des IST-Zustand kann man seine Stakeholder ermitteln, lernt den Prozess im aktuellen Zustand, seine Grenzen und die von außen einwirkenden Elemente kennen.

- Anforderungen erheben
  Für die Erhebung von Anforderungen existieren verschiedene Methoden und Quellen.

  Quellen für Anforderungen sind

  - Stakeholder, die Aufgrund ihrer Mitarbeit am Prozess oder durch Auswirkungen und Ergebnissen dieses, betroffen sind.
  - Prozessmodelle, in denen der Ablauf des Prozesses dokumentiert ist
  - Beobachtungen des Prozessalltags
  - Prozessbeschreibungen oder Problemberichte

  Methoden zur Erhebung von Anforderungen sind

  - Interviews mit Stakeholdern oder Verantwortlichen
  - Workshops mit Stakeholdern
  - Analysen von Berichten und Beschreibungen

  Je nach zur Verfügung stehender Quelle ist eine dazu passende Methode zur

Erhebung dieser anzuwenden.

- Anforderungen dokumentieren
  Mit dem Erheben von Anforderungen wird auch das dokumentieren dieser zu einem Thema. Wenn die Disziplin des Anforderungsmanagement in einem größeren Unternehmen schon weit fortgeschritten ist wird es für die Dokumentation evtl. schon geeignete Tools im Unternehmen geben. Bei kleinen Firmen wird dies in der Regel wohl nicht der Fall sein. Für die Anforderungsdokumentation stehen verschiedene Methoden, Darstellungsformen und Tools zur Verfügung. Diese dienen zum einen der besseren Visualisierung der Anforderungen, evtl. der Gestaltung beispielhafter Oberflächen als Entwurfsmuster können aber im Falle einer rein Verbalen Funktionsbeschreibung auch in Form einer Tabelle sein. Wichtig bei der Dokumentation ist immer, dass man einen Mix aus den zur Verfügung stehenden Methoden wählt um alle Aspekte, die durch die Anforderungen erhoben werden auch passend und widerspruchsfrei abbilden zu können.[7] Es ist darauf zu achten, dass man sich mit dem gewählten gut auskennt um das gesamte Potenzial ausschöpfen zu können und bei der Gestaltung darauf achten das auch die beteiligten Stakeholder das Ergebnis verstehen.

- Anforderungen Qualitätssichern Nachdem alle Anforderungen zusammengetragen wurden, müssen diese vom Anforderungsmanager auf Integrität, Vollständigkeit und Widersprüchlichkeit überprüft werden. In der Regel ist eine Iteration der vorangegangen Schritte notwendig um Lücken zu schließen und Widersprüchlichkeiten auszumerzen. Hierzu sind Interviews und erneute Workshops ein probates mittel. Am Ende muss eine Lücken freie Prozessbeschreibung durch die Anforderungen vorliegen und alle Stakeholder diese Anforderungen unterschreiben.

- Anforderungen verwalten Im Buch 100 Minuten für Anforderungsmanagement[8] wird dieser Punkt als Pflege und Verwaltung von Anforderungen be-

[7] Niebisch, *Anforderungsmanagement in sieben Tagen*, S. 93-137.
[8] Grande, *100 Minuten für Anforderungsmanagement : Kompaktes Wissen nicht nur für Projekt-*

zeichnet, was auch zutreffender ist. Denn es handelt sich hierbei nicht nur um ein reines Verwalten der Anforderungen.

Anforderungen können sich innerhalb eines Projektverlaufs ändern, es können Anforderungen hinzu kommen oder auch weg fallen. Hierbei ist es immer notwendig, dass jede Veränderung dokumentiert wird und die Anforderungen Versioniert sind, dadurch hat man bei Terminverschiebungen etwas in der Hand, da durch verändern oder erweitern der Anforderungen auch immer eine Verschiebung der Termine möglich sein kann. Neben dem Namen des ändernden, einem Datum und einer Versionsnummer sollte auch immer eine Erklärung, was geändert wurde, mit erfasst werden. Stakeholder sollten Zugriff auf die verwalteten Dokumente haben um jederzeit noch einmal nachlesen zu können, wie die Anforderungen getroffen wurde. Wenn es in der Entwicklungsphase zu viele Änderungen in den Anforderungen gibt, läuft man Gefahr, dass das Projekt zu sehr in die Länge gezogen wird und im Extremfall zu scheitern droht. Es ist darauf zu achten, dass die Änderungen mit der Zeit abnehmen und ab einem gewissen Zeitpunkt stabil sind und keine zusätzlichen Forderungen oder Änderungen mehr hinzu kommen.

2.6 Anforderungen

Anforderungen unterliegen verschiedenen Attribute, Arten und bedürfen bestimmter Formulierungen, damit sowohl Anwender als auch Entwickler unmissverständlich das gleiche aus ihr ableiten können.

leiter und Entwickler, S. 103.

2.6.1 Definition der Anforderung

In der Literatur finden sich die unterschiedlichsten Definitionen für Anforderungen und so kommt auch eine Analyse des KIT zur Schlussfolgerung, dass keine von ihnen als allgemein gültig betrachtet werden kann.[9] Es lässt sich dabei festhalten, dass eine Anforderung immer etwas definiert, was man von einem Produkt erwartet und diese vorzugsweise in irgendeiner schriftlichen Beschreibung festgehalten und dokumentiert werden sollte.

2.6.2 Arten der Anforderungen

Anforderungen lassen sich in viele unterschiedliche Arten unterteilen. Das hängt auch mit dem jeweiligen Projekt zusammen. So gibt es Arten von Anforderungen die immer auftreten und andere, die vorkommen können oder auch nicht. Bei der Unterteilung der Anforderungen ist man sich in der Literatur auch nicht immer einig. Während die einen als grobe Trennung in Anwenderanforderungen und Systemanforderungen unterteilen[10] wählen die anderen Funktionale-Anforderungen und Nicht-funktionale Anforderungen[11].[12] es werden die Anforderungen bei beiden Varianten noch in weitere Unterkategorien unterteilt.

- Funktionale- & Nicht-Funktionale Anforderungen
  Bei der Unterteilung in Funktionale und Nicht-funktionale Anforderungen werden alle Anforderungen, die direkt die Funktion und ihre Auswirkung beschreiben in Gruppe der Funktionalen Anforderungen einsortiert und alle

[9] 9 Christina Zehnter, Alexander Burger und Jivka Ovtcharova. *Key-Performance-Analyse von Methoden des Anforderungsmanagements*. Techn. Ber. Karlsruhe, 2012. URL: `http://digbib.ubka.uni-karlsruhe.de/volltexte/1000028693`, S. 2.3.1.

[10] Gerhard [Hrsg.] Versteegen und Alexander Heßeler, Hrsg. *Anforderungsmanagement : formale Prozesse, Praxiserfahrungen, Einführungsstrategien und Toolauswahl; [mit Praxisbeiträgen von BearingPoint (KPMG), HOOD Group und Telelogic; mit 11 Tab.]* Xpert.press. Berlin: Springer, 2004.

[11] Grande, *100 Minuten für Anforderungsmanagement : Kompaktes Wissen nicht nur für Projektleiter und Entwickler*.

[12] Niebisch, *Anforderungsmanagement in sieben Tagen*.

anderen Anforderungen in die Kategorie der Nicht-funktionalen Anforderungen. Diese werden dann noch in weitere Kategorien unterteilt.

- Anwender- & Systemanforderungen
  Bei der Gliederung in Anwenderanforderungen und Systemanforderungen, werden in die Kategorie der Anwenderanforderung alle einsortiert die beschreiben wie eine Software für den jeweiligen Anwender bedienbar sein sollte, sie beschreiben ein konkretes Problem,[13] Systemanforderungen beschreiben wie die Lösung eines Problems umgesetzt werden soll, was es an Schnittstellen zu anderen Prozessen gibt, wie die Struktur von Daten sein sollen. Anwenderanforderungen werde in der Regel durch die späteren Nutzer erhoben. Unterschiedliche Nutzergruppen, werden auch unterschiedliche Anforderungen hervor bringen, da sie den Prozess aus ihrer eigenen für sie erforderlichen Sicht betrachten und entsprechend die Anforderungen erheben. Systemanforderungen werden dagegen eher von den Entwicklern erhoben, die das System als ganzen betrachten müssen, was sind an Schnittstellen zu anderen Prozessen gegeben, wie können bestimmte Anforderungen umgesetzt werden, dass vom gewünschten Input auch der vom Anwender geforderte Output erzeugt wird.

Es ist für den Erfolg eines Projektes nicht ausschlaggebend, an welche Art der Unterteilung man sich hält. Wichtig dabei ist, dass man seine Anforderungen in irgendeiner Art und weise unterteilt und so die in großer Stückzahl auftretenden Anforderungen strukturiert.

2.6.3 Attribute von Anforderungen

Jede Anforderung muss bestimmte Attribute besitzen. Sie dienen zum einen der eindeutigen Unterscheidung, Beschreibung und der Rückführbarkeit. Dabei sollte

[13] Versteegen und Heßeler, *Anforderungsmanagement : formale Prozesse, Praxiserfahrungen, Einführungsstrategien und Toolauswahl; [mit Praxisbeiträgen von BearingPoint (KPMG), HOOD Group und Telelogic; mit 11 Tab.]*

beachtet werden, dass so viel Attribute wie erforderlich gewählt werden aber eben auch so wenig wie möglich. Denn Attribute müssen auch gepflegt werden und der Aufwand an Pflege und die Fehlermöglichkeit nehmen mit steigender Anzahl an Attributen zu. Bei Michael Grande[14] sind es fünf Pflichtattribute für Anforderungen welche auf jeden Fall vorhanden sein müssen.

- Identifikationsnummer (ID)
- Name
- Beschreibung
- Status
- Version

Nur mit diesen Attributen, lassen sich die Anforderungen eindeutig identifizieren und haben gerade ausreichend Informationen um verstanden werden zu können.

Durch die **Identifikationsnummer** ist jede Anforderung eindeutig identifizierbar. Es darf jede ID nur einmal geben und wenn eine ID einmal vergeben wurde, ist diese für immer belegt. Auch wenn Anforderungen gelöscht wurden, darf die Identifikationsnummer nicht erneut vergeben werden.

Der **Name** sollte so gewählt werden, dass er widerspiegelt, was die Anforderung enthält. Er sollte eindeutig sein.

Die **Beschreibung** einer Anforderung ist sehr wichtig, hier sollte drauf geachtet werden, dass je nach Art der Anforderung, das richtige Werkzeug für die Beschreibung gewählt wird. Es ist zwar möglich, einen kompletten Geschäftsprozess verbal zu beschreiben, die Darstellung mittels eines z.B. (e)EPK ist jedoch sehr wahrscheinlich einfacher und für jeden besser nachvollziehbar.

[14] Grande, *100 Minuten für Anforderungsmanagement : Kompaktes Wissen nicht nur für Projektleiter und Entwickler*, S. 40.

Der **Status** soll beim Abarbeiten helfen zu erkennen, in welchem Status sich die Anforderung befindet. Ist sie neu, freigegeben, geändert, in Arbeit, umgesetzt oder abgeschlossen. Dies kann einem beim Filtern und selektieren von Anforderungen behilflich sein und verhindert auch, das Tätigkeiten mehrfach ausgeführt werden.

Die **Version** erfüllt mehrere Aufgaben. Zum einen, der klassischen Dokumentation der Historie in der damit eingehenden Veränderungen. Zum anderen, dient sie jedoch auch zur Überwachung der Beständigkeit. Wenn eine Versionsnummer sich ständig ändert, heißt das, dass die entsprechende Anforderung einer dauernden Veränderung unterliegt. Hier sollte dann ein besonderer Augenmerk darauf gelegt werden, ob die Anforderung nicht richtig erhoben oder dokumentiert wurde oder andauernd neue Änderungswünsche hinzukommen. Dadurch können Projekte unnötig in die Länge gezogen werden und dadurch auch wesentlich teurer werden.

Je nach Projekt ist es sinnvoll weitere Attribute hinzu zu nehmen, so können z.b. Attribute wie Autor, Verantwortlicher, Ursprung dabei helfen, eine Rückverfolgbarkeit zu gewährleisten, wer eine Anforderung erhoben oder ein besonderes Interesse an dieser hat.

2.6.4 Methoden zum Erheben von Anforderungen

Für das Erheben von Anforderungen gibt es neben unterschiedlichen Quellen auch unterschiedliche Methoden, die je nach Projekt variieren. **Quellen** können neben den schon erwähnten Stakehodlern auch Dokumente, Beobachtungen oder vorhandenen Systeme sein.

Methoden zur Erhebung von Anforderungen sind das Interview, der Fragebogen, Workshop, Fokusgruppen, Brainstorming, Analyseverfahren für Prozesse und Dokumente.

Je nach Projekt muss man entscheiden, welche Quellen zur Verfügung stehen und mit welchen Methoden man die Anforderungen am besten aus den Quellen heraus bekommt. Bei den hier erwähnten Methoden handelt es sich nur um einen kleinen Teil der zur Verfügung stehenden Methoden, weitere sind im KIT SCIENTIFIC REPORTS 7620[15]

2.6.5 Dokumentieren von Anforderungen

Zum Dokumentieren von Anforderungen gibt es ja nach Verfahren, unterschiedlichste Tools als Hilfsmittel. Je nach Art der Anforderung eignen sich manche Dokumentationsformen mehr als andere, wichtig ist, das am Ende alle verstehen was gemeint ist.

- Natürlichsprachliche Anforderungen
  Für das verfassen natürlichsprachlicher Anforderungen gibt es einige Regeln, die beachtet werden sollten. Formulierungsregeln nach Marcus Grande[16]

  - Bilden Sie kurze Sätze
  - Formulieren Sie nur eine Anforderung pro Satz
  - Schreiben Sie kurze Sätze
  - Begründen Sie Anforderungen
  - Achten Sie auf die Eindeutigkeit des Subjekts
  - Vermeiden Sie Passiv
  - Beschreiben Sie im Text einer Anforderung niemals einen Lösungsweg

[15] Zehnter, Burger und Ovtcharova, *Key-Performance-Analyse von Methoden des Anforderungsmanagements*, S. 16-25.
[16] Grande, *100 Minuten für Anforderungsmanagement : Kompaktes Wissen nicht nur für Projektleiter und Entwickler*, S. 71.

– Schreiben Sie nur exakt testbare Anforderungen

Das soll heißen, man soll Anforderungen so formulieren, dass sie von keinem fehlinterpretiert werden können. Dabei sollen keine lange Sätze entstehen. Auch Dinge, die einem Teil der Projektbeteiligten *eh klar* sind, müssen erfasst werden und sie sollten im aktiv formuliert sein, so dass eindeutig ist von wem eine Aktion ausgeht. Die Anforderungen dürfen sich nicht gegenseitig widersprechen und jede Anforderung darf auch nur eine Anforderung enthalten.

Um die natürlichsprachliche Anforderungen übersichtlicher zu halten, ist es empfehlenswert, sie eine Satzschablone bereit zu legen, nach der man die Sätze Formuliert. Chris Rupp und die SOPHISTen haben dies in mehreren Schablonen verdeutlicht. Sie haben für die verschiedenen Arten von Anforderungen Schablonen entwickelt. Abbildung 2.1 zeigt beispielhaft die Eigenschafts-Master Schablone für Nicht-Funktionale Anforderungen.

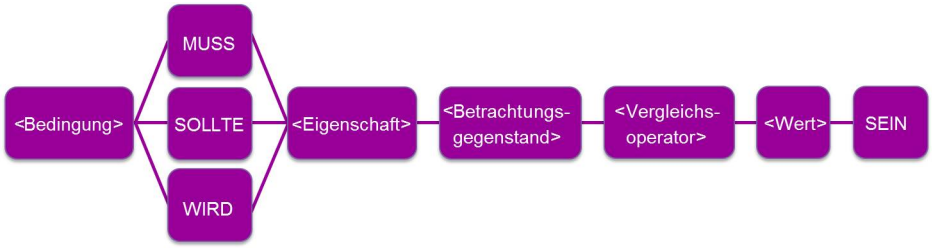

Abbildung 2.1: Eigenschaftsschablone[a]

[a] Chris Rupp, Hrsg. *Requirements-Engineering und -Management : aus der Praxis von klassisch bis agil*. 6., aktualisierte und erw. Aufl. München: Hanser, 2014. ISBN: 978-3-446-43893-4, S. 235.

- Diagramme und Modelle

Die Alternative zur Natürlichsprachlichen Dokumentation, ist die Visualisierung mittels Diagrammen und Modellen. Hierfür wurden im Laufe der Jahre eine Vielzahl von unterschiedlichen Standards entwickelt, welche zur Dar-

stellung von unterschiedlichsten Anforderungsarten geeignet sind. Es gibt Tools zum beschreiben von Oberflächen, Datenstrukturen, Prozessen usw.. Die folgende Auflistung von Diagrammen und Modellen stellt eine kleine Auswahl dar und hat nicht den Anspruch auf Vollständigkeit.

- Mock-Up
- UML
- USE-Case
- BPMN
- (e)EPK
- Klassendiagramme
- ER Diagramme

Hier ist auf die richtige Auswahl des Werkzeugs zu achten. Es muss nicht nur zum beschreiben der Anforderung geeignet sein sondern derjenige der die Anforderung damit umzusetzen versucht muss es auch bedienen können. So eignen sich Mock-Up's zum beschreiben und dokumentieren von Oberflächen. Mit Hilfe von Use-Case, UML (Unified Modelling Language), (e)EPK (erweiterte Ereignis gesteuerte Prozesskette) , BPMN (Business Process Modelling Notation) lassen sich Funktionale Anforderungen in Form von z.B. Prozessbeschreibungen visualisieren wobei für Daten, Schnittstellen und Berechtigungen neben Klassen- und ER-Diagramm ebenfalls BPMN und UML verwendet werden können.

3 Abbildungsverzeichnis

4 Literaturverzeichnis

Collins-Sussman, Ben, Brian W. Fitzpatrick und C. Michael Pilato. *Versionskontrolle mit Subversion : Software-Projekte intelligent koordinieren*. 2. Aufl. Beijing: OReilly, 2006. ISBN: 3-89721-460-1; 978-3-89721-460-6.

Fogel, Karl und Moshe Bar. *Open Source-Projekte mit CVS : [verteilte Softwareentwicklung mit dem Concurrent Versions System; Grundprinzipien und Anwendung verteilter Softwareentwicklung; CVS-Server installieren und verwalten; jetzt mit umfassender Befehlsreferenz und praktischer Schnellreferenz]*. 2., erw. u. überarb. Aufl. Bonn: MITP-Verl., 2002. ISBN: 3-8266-0816-X.

Grande, Marcus. *100 Minuten für Anforderungsmanagement : Kompaktes Wissen nicht nur für Projektleiter und Entwickler*. 2., aktualisierte Aufl. 2014. Springer Vieweg, 2014.

Niebisch, Thomas. *Anforderungsmanagement in sieben Tagen*. Springer-Verlag, 2013.

Rupp, Chris, Hrsg. *Requirements-Engineering und -Management : aus der Praxis von klassisch bis agil*. 6., aktualisierte und erw. Aufl. München: Hanser, 2014. ISBN: 978-3-446-43893-4.

Versteegen, Gerhard [Hrsg.] und Alexander Heßeler, Hrsg. *Anforderungsmanagement : formale Prozesse, Praxiserfahrungen, Einführungsstrategien und Toolauswahl; [mit Praxisbeiträgen von BearingPoint (KPMG), HOOD Group und Telelogic; mit 11 Tab.]* Xpert.press. Berlin: Springer, 2004.

Zehnter, Christina, Alexander Burger und Jivka Ovtcharova. *Key-Performance-Analyse von Methoden des Anforderungsmanagements*. Techn. Ber. Karlsruhe, 2012. URL: `http://digbib.ubka.uni-karlsruhe.de/volltexte/1000028693`.

Eidesstattliche Erklärung

Studierender: Maik Ledwina
Matrikelnummer: 880767

Hiermit erkläre ich, dass ich diese Arbeit selbstständig abgefasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

....

Ort, Abgabedatum Unterschrift (Vor- und Zuname)