IB 232-2015 J 11

IB 232-2015 J 11

Entwicklung eines Software-Qualitätskonzepts zur Definition von Entwicklungsprozessen im wissenschaftlichen Umfeld

Julian Springer

Institut für Aeroelastik

DLR

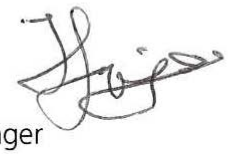
Dokumenteigenschaften

Titel	Entwicklung eines Software-Qualitätskonzepts zur Definition von Entwicklungsprozessen im wissenschaftlichen Umfeld
Betreff	Bachelorarbeit für die Prüfung zum Bachelor of Engineering an der Dualen Hochschule Baden-Württemberg Mannheim
Institut	Institut für Aeroelastik
Erstellt von	J. Springer
Geprüft von	Dr.-Ing. Y. Govers
Freigabe von	Prof. Dr.-Ing. L. Tichy
Datum	18.12.2015
Version	1.0
Dateipfad	DLR IB 232-2015 J 11 v1.0.pdf

Institutsleiter:

Prof. Dr.-Ing. L. Tichy

Autor:



J. Springer

Abteilungsleiter:

Dr.-Ing. M. Böswald

Gruppenleiter:

Dr.-Ing. Y. Govers

Zusammenfassung

Durch die wachsende Bedeutung von Software steigen auch die Qualitätsanforderungen, die an sie gestellt werden. Täglich verlassen wir uns auf ihre einwandfreie Funktionalität, wobei Fehler in der Software zu direkten Konsequenzen für uns führen können. Dies gilt insbesondere auch für jene Software, welche im Deutschen Zentrum für Luft- und Raumfahrt e.V. für die Durchführung von Schwingungsuntersuchungen an Flugzeugen entwickelt wird. Nicht nur ein großer Funktionsumfang auf neuestem Stand der Wissenschaft ist für diese Software unerlässlich, sondern auch eine hohe Qualität.

Ein konkretes, ganzheitliches Konzept zur Qualitätssicherung für die Softwareentwicklung ist momentan nicht vorhanden. Um die gestellten Vorgaben auch in Zukunft erfüllen zu können, ist es die Aufgabe dieser Bachelorarbeit, ein Qualitätskonzept für die Softwareentwicklung zu erstellen. Als Hauptbestandteil dieses Konzepts wird ein zugeschnittener Entwicklungsprozess entworfen, der Maßnahmen zur Qualitätssicherung enthält. Dieser Prozess ist zu einem hohen Grad automatisiert und stellt die Bearbeitung von Anforderungen der Benutzer in den Vordergrund. Für den konzipierten Prozess werden geeignete Softwarewerkzeuge ausgewählt und evaluiert. Zusätzlich wird er exemplarisch für die Entwicklung einer Software umgesetzt und in den produktiven Einsatz überführt.

Unterstützt werden soll der verbesserte Prozess durch Richtlinien zur Softwareentwicklung, die die Einhaltung der Qualitätssicherungsmaßnahmen sicherstellen sollen. Um dies gewährleisten zu können, müssen die Richtlinien kontinuierlich verbessert und an den Prozess angepasst werden.

Abstract

Along with the growing importance of software, the quality requirements for software also increase constantly. Every day, we rely on the correct functionality of software we use. Therefore, errors in the software can have a direct impact on us. This also applies to the software developed at the German Aerospace Center, which is used for vibration testing of airplane structures. The software has to meet both, requirements for state-of-the-art functionality and a high quality standard, at the same time.

Currently, no such quality assurance concept for the development of software exists. To fulfil the compulsory requirements, it is the purpose of this bachelor thesis to create a quality concept for future software developments. The main component of the concept is a tailored development process, which contains quality assurance measures. The process is largely automated and focusses on the processing of tasks resulting from user feedback. For the implementation of the development process, suitable software tools are selected and evaluated. Furthermore, the implementation is put into operation for a selected development project.

In addition to the improved process, software development guidelines are supposed to guarantee compliance with the quality assurance measures. To ensure this, the guidelines have to be improved continuously and changes in the development process must be applied to the guidelines.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AE	Institut für Aeroelastik
API	Application Programming Interface
CHM	Compiled HTML Help
CI	Continuous Integration
DIN	Deutsches Institut für Normung e. V.
DLR	Deutsches Zentrum für Luft- und Raumfahrt e.V.
EN	Europäische Norm
GPL	GNU General Public License
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ID	Identifikator
IDE	Integrated Development Environment
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
LDAP	Lightweight Directory Access Protocol
MAML	Microsoft Assistance Markup Language
MIT	Massachusetts Institute of Technology
Ms-PL	Microsoft Public License
Ms-RL	Microsoft Reciprocal License
MSBuild	Microsoft Build Engine
MSDN	Microsoft Developer Network
MVVM	Model View ViewModel
PHID	Phabricator Identifikator
QMH	Qualitätsmanagement-Handbuch
RTF	Rich Text Format
SAS	Abteilung Strukturdynamik und aeroelastische Systemidentifikation

SHFB	Sandcastle Help File Builder
URL	Uniform Resource Locator
VDI	Verein Deutscher Ingenieure
WiX	Windows Installer XML
WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language
XML	Extensible Markup Language
XP	Extreme Programming

1 Einleitung

In nahezu allen Lebenslagen, im privaten und geschäftlichen Bereich, spielt der Einsatz von Software eine immer größere Rolle (vgl. [IfD14], [MB14], [Hof13, S. 1 ff.]). Entwickler stellen immer mehr Software zur Verfügung, gleichzeitig steigt der Umsatz mit dem Vertrieb von Software stetig an (vgl. [AG15], [IDA15]). Mit der wachsenden Menge an Software, die weltweit eingesetzt wird, steigt auch die Bedeutung, die sie für uns Menschen hat. Bei einem Großteil an technischen Systemen verlassen wir uns täglich auf die einwandfreie Funktionalität der Software.

Sollten jedoch Fehler in einer Software auftreten, deren korrekte Funktionsweise zur Notwendigkeit geworden ist, kann dies direkte Konsequenzen für Prozesse, Anwender und Beteiligte haben. Ob große Softwarefehler, wie sie bei der Steuerungssoftware eines Flugzeuges zum Absturz führen können (vgl. [Fri15]), oder kleinere Softwarefehler, die ein Unternehmen Geld und Zeit kosten können (vgl. [Cla01]): Die Anwendung eines Konzepts zur Software-Qualitätssicherung bietet viele Möglichkeiten, Fehler in Software frühzeitig aufzufinden und zu reduzieren (vgl. [Hof13, S. 19 ff.]).

Die Entwicklung eigener Software erfolgt auch in der zum Institut für Aeroelastik (AE) gehörenden Abteilung Strukturdynamik und aeroelastische Systemidentifikation (SAS) des Deutschen Zentrums für Luft- und Raumfahrt e.V (DLR). Die Abteilung AE-SAS beschäftigt sich mit den Schwingungsvorgängen an Strukturen, die in der Luft- und Raumfahrt eingesetzt werden. Im Betrieb treten, ganz natürlich, Schwingungen auf, indem die Rückstell- und Trägheitskräfte mit den aerodynamischen Kräften interagieren. Zur Identifikation des strukturdynamischen Verhaltens werden die Strukturen experimentell in Standschwingungsversuchen analysiert. Standschwingungsversuche gehören zu den Tests, die ein Flugzeug auf dem Weg zur Zulassung durchlaufen muss (vgl. Abbildung 1). Diese Untersuchungen sind jedoch nicht nur für Flugobjekte unerlässlich, sondern auch für Strukturen von Windkraftanlagen und Automobilen notwendig. Die Abteilung SAS führt Standschwingungsversuche an Flugzeugstrukturen durch und entwickelt spezielle Software, die dabei zum Einsatz kommt.

Abbildung 1: Standschwingungsversuch an einem Airbus A380 (Copyright Airbus S.A.S.)

1.1 Zielsetzung

An die Funktionalität der Software, die von der Abteilung AE-SAS zur Durchführung der Standschwingungsversuche verwendet wird, werden hohe Anforderungen gestellt. Aus dem Selbstverständnis als Forschungsinstitut heraus versucht das DLR, immer auf dem neuesten Stand der Forschung zu sein und diese voranzutreiben. Da keine industrielle Software verfügbar ist, welche den gestellten Anforderungen gerecht wird, besteht die verwendete Software zum Großteil aus Eigenentwicklungen. Diese Eigenentwicklungen werden von einer kleinen Gruppe an Softwareentwicklern in Zusammenarbeit mit den Wissenschaftlern der Abteilung erarbeitet.

Der Funktionalität auf dem neuesten Stand der Wissenschaft stehen hohe Qualitätsanforderungen gegenüber, die beispielsweise von industriellen Auftraggebern gestellt werden. Während die Einhaltung dieser Anforderungen auf Seiten der Verfahren, Messtechnik und Hardware durch umfassende Maßnahmen sichergestellt wird, ist die Einhaltung selbiger Anforderungen an die Softwarequalität ein Thema, welches mit dieser Arbeit vorangetrieben werden soll. Zu beachten ist, dass der Ausfall einer Softwarekomponente einen ebenso großen Schaden anrichten kann wie der Ausfall einer Hardwarekomponente.

Das Ziel dieser Bachelorarbeit ist es, ein Qualitätskonzept für die Softwareentwicklungsprozesse in der Abteilung AE-SAS aufzustellen. Dieses Konzept soll einen hohen Qualitätsstandard der zu entwickelnden Software garantieren. Dadurch soll gewährleistet werden, dass Software zuverlässig funktioniert, sofern sie nach dem ausgearbeiteten Konzept und den Richtlinien entwickelt wurde.

Passend zu den vorherrschenden Rahmenbedingungen müssen die künftigen Entwicklungsmethoden ausgewählt und an die Bedürfnisse angepasst werden. Zu beachten ist hierbei, dass neben den Qualitätsanforderungen weitere Faktoren, wie die geringe Anzahl an Softwareentwicklern, in das Konzept des Entwicklungsprozesses miteinbezogen werden.

1.2 Überblick

In der vorliegenden Bachelorarbeit werden zu Beginn in Kapitel 2 grundlegende Prozesse und Vorgehensmodelle der Softwareentwicklung betrachtet, die für das zu erstellende Qualitätskonzept von Bedeutung sind. Es folgt in Kapitel 3 eine Analyse der bisherigen Umsetzung und der Anforderungen an das Qualitätskonzept. Kapitel 4 beinhaltet mit dem Konzept eines Entwicklungsprozesses für die Abteilung AE-SAS den Grundbestandteil des zu erstellenden Qualitätskonzepts. In Kapitel 5 wird die Umsetzung des zuvor vorgestellten Konzepts beschrieben. Die Überarbeitung der im Prozess eingesetzten Software VersionInfo wird in Kapitel 6 dargestellt. Ein weiterer Bestandteil des Qualitätskonzepts sind die Richtlinien zur Softwareentwicklung, deren Entwurf in Kapitel 7 erläutert wird und die den Entwicklungsprozess ergänzen. Eine Auswertung des mit dieser Arbeit eingeführten Qualitätskonzepts findet zum Abschluss in Kapitel 8 statt.

2 Grundlagen

Zur Erstellung eines Software-Qualitätskonzepts bietet sich die Betrachtung einiger Grundlagen der Softwareentwicklung an. Sie können, angepasst auf die konkreten Anforderungen, in die Entwicklung eines Konzeptes für die Abteilung AE-SAS mit einfließen. Zu Beginn wird in diesem Kapitel betrachtet, wie der Begriff Qualität in der Softwareentwicklung zu verstehen ist. Nachfolgend werden in Abschnitt 2.2 einige Vorgehensmodelle erläutert, die für die Konzeption eines neuen Entwicklungsprozesses herangezogen werden. Des Weiteren werden in Abschnitt 2.3 und 2.4 etablierte Konzepte der Softwareentwicklung vorgestellt, die ebenfalls in den neuen Entwicklungsprozess einfließen.

2.1 Qualität in der Softwareentwicklung

Um den Begriff der Qualität, die vom aufzustellenden Qualitätskonzept an der Software garantiert werden soll, fassen zu können, bietet es sich an, etablierte Deutungen dieses Begriffes zu betrachten. Normen des Deutschen Instituts für Normung e. V. (DIN) und der International Organization for Standardization (ISO) stellen allgemein akzeptierte und weit verbreitete Dokumente dar, die von Organisationen deutschland- bzw. weltweit als Bezugspunkt für Anforderungen an Produkte und Verfahren eingesetzt werden (vgl. [DIN15]). Sie werden auch vom zentralen Qualitätsmanagementsystem des DLRs adressiert und umgesetzt. Aus diesem Grund werden sie zur Definition des Begriffs Qualität miteinbezogen.

Die Norm DIN EN ISO 9000 [DIN14] definiert den Begriff Qualität wie folgt:

> „Grad, in dem ein Satz inhärenter Merkmale eines Objekts Anforderungen erfüllt“[1]

Diese allgemein gehaltene Definition wird von der Norm DIN ISO 9126 [DIN01] für Softwareprodukte spezifiziert. Dort lautet die Definition des Begriffs Software-Qualität:

> „Software-Qualität ist die Gesamtheit der Merkmale und Merkmalswerte eines Software-Produkts, die sich auf dessen Eignung beziehen, festgelegte Erfordernisse zu erfüllen.“[2]

[1] DIN Deutsches Institut für Normung e.V.: *Qualitätsmanagementsysteme – Grundlagen und Begriffe (ISO/DIS 9000:2014)*. Berlin, 08.2014 ([DIN14])

[2] DIN Deutsches Institut für Normung e.V.: *Software-Engineering - Qualität von Software-Produkten - Teil 1: Qualitätsmodell (ISO/IEC 9126:2001)*. Berlin, 06.2001 ([DIN01]), zit. nach Hoffmann, Dirk W.: *Software-Qualität*. 2., aktualisierte und korrigierte Aufl. Berlin : Springer Vieweg, 2013 (eXamen.press). – ISBN 978–3–642–35699–5 ([Hof13])

Die Qualität einer Software kann, auf Basis der Definitionen in den DIN ISO Normen 9000 und 9126, nicht aufbauend auf einem einzigen Merkmal betrachtet werden. Stattdessen besteht die Qualität einer Software aus einem Satz an Merkmalen (vgl. [Hof13]). Für jedes der Merkmale ist zu analysieren, wie weit die betrachtete Software sich dort zur Erfüllung der Anforderungen eignet.

In der Literatur wird eine Vielfalt an verschiedenen Qualitätsmerkmalen genannt, die einen Beitrag zu der Qualität einer Software liefern. Die Norm DIN ISO 9126 nennt sechs Qualitätsmerkmale für Software (nach [Bal09, S. 468 ff.]):

- Funktionalität
- Zuverlässigkeit
- Benutzbarkeit
- Effizienz
- Wartbarkeit
- Portabilität

In allen diesen Merkmalen existieren Anforderungen an die zu entwickelnde Software. Je mehr die Anforderungen erfüllt sind, desto höher ist das Qualitätsniveau anzusehen (vgl. [Bal09, S. 466]).

Da es die Aufgabe des geforderten Software-Qualitätskonzepts ist, einen gewissen Qualitätsstandard der entwickelten Software zu garantieren, bedeutet dies, dass es für einen möglichst hohen Grad der Erfüllung jedes Qualitätsmerkmals sorgen muss. Dafür müssen möglichst viele Anforderungen, die zu den einzelnen Qualitätsmerkmalen gehören, erfüllt werden. Die gestellten Anforderungen für das Qualitätskonzept sind den Rahmenvorgaben und den Spezifikationen der jeweiligen Software zu entnehmen. Für die Aufgabenstellung dieser Bachelorarbeit sind sie in Kapitel 3 zu finden.

2.2 Vorgehensmodelle

Die Organisation eines Projektes ist ein wichtiger Faktor für dessen Erfolg (vgl. [KWV14, S. 15 ff.]). Gleiches gilt auch für Softwareprojekte. Eine wesentliche Aufgabe der Organisation von Softwareprojekten ist es, einen geeigneten Entwicklungsprozess für das Projekt festzulegen (vgl. [BK13, S. 85]). Werkzeuge für die Definition eines Entwicklungsprozesses und einer Organisationsstruktur stellen Vorgehensmodelle dar (vgl. [AS14, S. 29]). Sie bilden die

Grundlage für alle Phasen des Projektverlaufs und beschreiben Vorgehensweisen zur Lösung von Aufgaben im Projekt (vgl. [BK13, S. 85 f.]).

In der Softwareentwicklung werden, neben den klassischen Vorgehensmodellen wie Wasserfall- und Spiralmodell, zunehmend flexiblere Vorgehensmodellen eingesetzt. In Abschnitt 2.2.1 werden die, unter diese Kategorie fallenden, agilen Vorgehensmodelle näher erläutert. Es folgen in Abschnitt 2.2.2 Vorgehensmodelle der schlanken Softwareentwicklung. Agile und schlanke Vorgehensmodelle bieten sich besonders an, wenn Anforderungen zu Beginn eines Softwareprojekts noch nicht klar definiert wurden, sondern während der Entwicklung einer Software entstehen (vgl. [BK13, S. 99]). Aus diesem Grund werden sie in dieser Arbeit genauer betrachtet.

2.2.1 Agile Softwareentwicklung und Scrum

Um bessere Wege zur Softwareentwicklung zu finden, wurden 2001 von Beck et al. mit dem sogenannten *Agilen Manifest* [BBv+01] vier Werte niedergeschrieben, die als Basis der agilen Softwareentwicklung dienen. Sie sind in Anhang A zu finden. Grundgedanke ist, dass die Entwickler im Mittelpunkt der Softwareentwicklung stehen. Wichtig für die Entwicklung sind eine enge Einbeziehung des Auftraggebers und eine schnelle Reaktionsfähigkeit auf Anforderungsänderungen (vgl. [EEG14, S. 75]). Die Werte des agilen Manifests sollen lediglich eine Orientierung für die Softwareentwicklung bieten. Es ist nicht ihr Ziel, dass die als weniger wichtig angegeben Punkte (in Anhang A auf das Wort „over“ folgend) außer Betracht gelassen werden.

Ein häufig genannter Vertreter der agilen Vorgehensmodelle ist Scrum. Bei Scrum handelt es sich um einen iterativen und inkrementellen Prozess (vgl. [EEG14, S. 90 ff.]). Es wird zu Beginn eines Projektes ein grober Rahmen definiert, der im Verlauf der Entwicklung verfeinert wird. Die Anforderungen an die zu realisierende Software werden im sogenannten Product Backlog festgehalten. Die Organisation des Entwicklerteams erfolgt selbstständig (vgl. [BK13, S. 99 ff.]). Die Entwicklungen werden in Iterationen fester Länge durchgeführt, den Sprints. Für die Sprints ist eine Länge von maximal 30 Tagen vorgesehen. Die täglich stattfindenden Meetings in Scrum werden als Daily Scrum bezeichnet.

In jedem Sprint werden nun zu Beginn Aufgaben aus dem Product Backlog ausgewählt, die innerhalb des Sprints bearbeitet werden sollen. Sie landen damit im Sprint Backlog und werden von dort von den Entwicklern aufgegriffen. Am Ende jedes Sprints soll eine lauffähige

Softwareversion vorliegen. Änderungsanforderungen an dieser Version werden in das Product Backlog eingefügt (vgl. [BK13, S. 101]).

Scrum beinhaltet eine Einteilung der beteiligten Personen zu drei Rollen (nach [EEG14, S. 92 f.]):

Der Scrum Master hat die Aufgabe, dafür zu sorgen, dass das Entwicklerteam arbeitsfähig bleibt. Er sorgt dafür, dass die Regeln von Scrum eingehalten werden und moderiert die Scrum Meetings. Er hat keine Weisungsbefugnis.

Der Product Owner trägt die Verantwortung über das Projekt. Er stellt die Schnittstelle zum Auftraggeber dar.

Das Entwicklerteam setzt die Anforderungen aus dem Product Backlog um.

Die Vereinigung mehrerer Rollen in einer Person ist grundsätzlich möglich. Trotzdem kann es sich, je nach Voraussetzungen und Charakter der beteiligten Personen, schwierig gestalten (vgl. [Klo12]).

2.2.2 Schlanke Softwareentwicklung und Kanban

Die schlanke Softwareentwicklung ist eine Denkweise, die auf dem Entwicklungs- und Produktionsvorgehen des japanischen Automobilherstellers Toyota basiert (vgl. [PP03, Kap. Introduction]). Im Unterschied zur agilen Softwareentwicklung existiert keine feste Definition der zugrundeliegenden Werte. In der Literatur wird sich für die Grundlagen der schlanken Softwareentwicklung zumeist auf die sieben Werte von Mary und Thomas Poppendieck [PP03] bezogen, die in Anhang B zu finden sind. Schlanke Softwareentwicklung kann als Erweiterung der agilen Softwareentwicklung verstanden werden, indem die Werte und Ideen des schlanken Vorgehens auf die theoretischen Grundlagen der agilen Softwareentwicklung angewendet werden (vgl. [PP03, Kap. Introduction]).

Eine Möglichkeit, schlanke Softwareentwicklung in einem Vorgehensmodell umzusetzen, stellt Kanban dar. Kanban definiert keine Iterationen oder Zyklen, sondern wird kontinuierlich angewendet. Es wird ein gleichmäßiger Arbeitsfortschritt angestrebt (vgl. [Epp11, S. 23 f.]). Im Vergleich zu Scrum entfällt somit auch die Auswahl von Aufgaben, die während einer Iteration zu erledigen sind. Außerdem werden bei Kanban keine, speziell für den Prozess notwendigen, Rollen definiert.

Mit Hilfe von Kanban werden die Werte der schlanken Entwicklung durch verschiedene Techniken umgesetzt. Im Folgenden werden für diese Bachelorarbeit relevante Kanban-Techniken nach Epping [Epp11] genannt, die zur Umsetzung der Werte dienen (aus [Epp11, S. 87 ff.], Erklärungen vgl. ebd.):

Testautomatisierung

Die Testautomatisierung beschreibt die Möglichkeit zur automatisierten Ausführung von Tests. Sie dient dazu, Validierungen gegenüber bestimmten Kriterien regelmäßig und wiederholbar durchzuführen. Diese Kriterien können beispielsweise Abnahmekriterien sein und werden in Testfällen definiert.

Code Reviews

Code Reviews sind Überprüfungen von Teilen des Quellcodes, die der Fehleraufdeckung und dem Informationsaustausch dienen (vgl. Abschnitt 4.4)

Continuous Integration

Siehe Abschnitt 2.3.

Kanban-Board

Ein Kanban-Board stellt die Phasen einer Wertschöpfungskette zusammen mit den Anforderungen dar, die sich in der jeweiligen Phase befinden. Die Anforderungen werden auf Karten vermerkt (vgl. [Epp11, S. 115]). Abbildung 2 stellt ein beispielhaftes Kanban-Board dar, welches in der Softwareentwicklung verwendet wird.

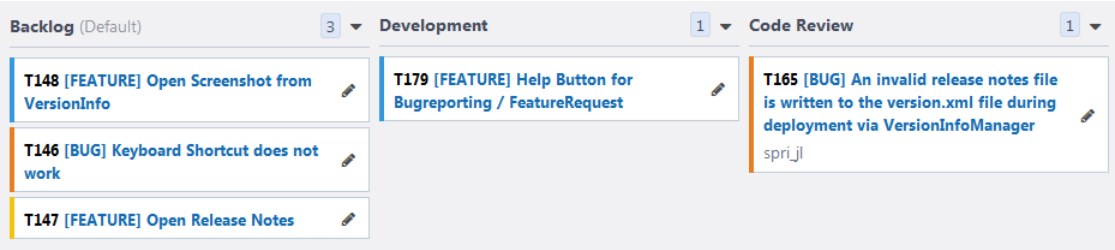


Abbildung 2: Bildschirmfoto eines einfachen Kanban-Board in der Software Phabricator.

Da in Kanban keine Iterationen existieren, werden die Aufgaben auch nicht zu Beginn einer Iteration verteilt. Stattdessen wählen die Entwickler selbstständig Aufgaben aus, die sie bearbeiten möchten. Dieses Prinzip wird als Pull bezeichnet und lässt sich durch ein Kanban-Board realisieren (vgl. [Epp11, S. 54 ff.]).

2.3 Kontinuierliche Integration

Das Konzept der kontinuierliche Integration (auch als *Continuous Integration* bezeichnet) entstammt einer Veröffentlichung zum agilen Vorgehensmodell Extreme Programming (XP) von Beck aus dem Jahr 1999 [Bec99]. Ziel der kontinuierlichen Integration ist es, dass neue Entwicklungen möglichst schnell in die bestehende Software integriert und getestet werden. Kombiniert mit einer ausreichenden Anzahl an automatisierten Tests zeigt kontinuierliche Integration, dass die Software auch mit neuen Änderungen wie gewünscht funktioniert (vgl. [HF11, S. 55 ff.]).

Die häufigste Umsetzung der kontinuierlichen Integration ist ein asynchroner Prozess (vgl. [BA04, Kap. 7.11]). Bei diesem werden die Änderungen auf ein Versionsverwaltungssystem geladen. Das Buildsystem überprüft das Versionsverwaltungssystem auf Änderungen und beginnt mit der Durchführung des Builds und der Tests, sobald es die Änderungen bemerkt hat. Falls beim Build oder den Tests Probleme auftreten, wird der Entwickler der Änderungen benachrichtigt.

Nach [HF11, S. 56 f.] sind drei Voraussetzungen notwendig, um kontinuierliche Integration einsetzen zu können: Neben einer Versionsverwaltung und einem automatisierten Build mit Tests ist die Zustimmung der Gruppe entscheidend. Das Konzept zeigt nur einen Nutzen, wenn die Entwickler ihre Änderungen häufig in das Versionsverwaltungssystem integrieren und möglicherweise auftretende Fehler am Build mit höchster Priorität beheben.

2.4 Kontinuierliche Auslieferung

Eine Ergänzung zur kontinuierlichen Integration stellt die kontinuierliche Auslieferung (auch als *Continuous Delivery* bezeichnet) dar. Sie erweitert die kontinuierliche Integration um die Auslieferung der entwickelten Software (vgl. [HF11, S. 24 ff.]).

Ein zentraler Begriff der kontinuierlichen Auslieferung ist die sogenannte *Deployment Pipeline*. Die Deployment Pipeline beschreibt den Prozess, der durchgeführt werden muss, damit Änderungen vom Versionsverwaltungssystem zum zukünftigen Benutzer gelangen (vgl. [HF11, S. 105 ff.]). Dieser Prozess ist ein mehrstufiges Verfahren. Auf den Buildprozess und die Unittests folgen automatisierte Akzeptanztests, später manuelle Akzeptanztests und anschließend die Verteilung der Softwareversion (vgl. [HF11, S. 109, Abbildung 5.2]). Die entstehenden Artefakte (Programmdateien, Berichte oder Metadaten) werden jeweils archiviert.

Damit eine Umsetzung der kontinuierlichen Auslieferung effizient funktioniert, haben Humble und Farley acht Prinzipien der Softwareverteilung aufgestellt (aus [HF11, S. 24 ff.], Erklärungen vgl. ebd.):

Create a Repeatable, Reliable Process for Releasing Software
: Erstelle einen reproduzierbaren und zuverlässig funktionierenden Prozess zur Verteilung von Software.

Automate Almost Everything
: Automatisiere alles, was automatisiert werden kann.

Keep Everything in Version Control
: Lagere sämtliche Dateien in einem Versionsverwaltungssystem.

If It Hurts, Do It More Frequently, and Bring the Pain Forward
: Führe Schritte des Prozesses, die viele Probleme verursachen, häufiger durch.

Build Quality In
: Integriere Maßnahmen zur Qualitätssicherung in den gesamten Prozess, um Fehler früh zu finden.

Done Means Released
: Eine Aufgabe ist dann erledigt, wenn sie mit der Software verteilt wurde.

Everybody Is Responsible for the Delivery Process
: Jeder Entwickler sollte sich für den Verteilungsprozess verantwortlich fühlen.

Continuous Improvement
: Der Verteilungsprozess sollte stetig angepasst und verbessert werden.

Diese Prinzipien sind unabhängig von dem genauen Auslieferungsprozess und können auch ohne mehrstufige Deployment Pipeline eingesetzt werden. Sie sollen jeweils zu dem Ziel beitragen, einen nahezu vollständig automatisierten Verteilungsprozess mit einer hohen Geschwindigkeit und Qualität zu implementieren.

3 Anforderungsanalyse

Die Entwicklung von Software in der Abteilung AE-SAS hat für die Arbeit der gesamten Abteilung eine hohe Wichtigkeit. Dem entgegen steht, dass zur Sicherstellung einer hohen Qualität dieser Software kein konkretes, auf die Abteilung zugeschnittenes Qualitätskonzept vorliegt.

Auch wenn es bisher nicht zu schwerwiegenden Problemen durch die eigene Software gekommen ist, kann hierdurch keine Aussage darüber getroffen werden, ob dies ebenfalls für die Zukunft anzunehmen ist. Die Menge an in der Abteilung entwickelter Software wächst gleichzeitig mit den internen und externen Anforderungen. Es werden immer umfangreichere und qualitativ hochwertigere Anwendungen gefordert. Die Erfüllung dieser Anforderungen wird, ohne bestehendes Software-Qualitätskonzept, immer schwieriger. Um ein Qualitätskonzept aufzustellen, werden in Abschnitt 3.1 die bestehenden Abläufe der Softwareentwicklung genauer betrachtet. Anschließend erfolgt in Abschnitt 3.2 eine Analyse der Vorgaben an ein neues Konzept. In Abschnitt 3.3 werden schließlich die Zielsetzungen des Qualitätskonzepts definiert.

3.1 Entwicklungsprozess zu Beginn der Bachelorarbeit

Die Softwareentwicklung innerhalb der Abteilung verläuft aktuell rudimentär und mit einem niedrigen Automatisierungsgrad. Deutlich wird dies bei einer Betrachtung des bisherigen Softwareentwicklungsprozesses. Dafür sei in Abbildung 3 ein Ablauf gegeben, wie er typischerweise in der Abteilung auftritt, sobald eine Entwicklungsaufgabe für ein Softwareprojekt gestellt wird.

Ein konkretes, ganzheitliches Konzept zur Wahrung der Softwarequalität ist für den Entwicklungsprozess nicht vorhanden. Die Aufgabenverteilung und -verwaltung erfolgt zumeist informell und ohne standardisierten Kommunikationskanal. Gleiches gilt für Berichte über Fehler in der Software oder Anfragen nach weiterer Funktionalität.

Die Erstellung neuer Versionen der entwickelten Software erfolgt manuell durch die Entwickler. Alle Entwicklungsschritte werden lokal auf den Computern der Entwickler vorgenommen. Ein automatisiertes Buildsystem, welches sämtliche, zur Erstellung der Software notwendige Schritte übernimmt, steht ihnen dafür nicht zur Verfügung. Ebenso wird der überwiegende Teil

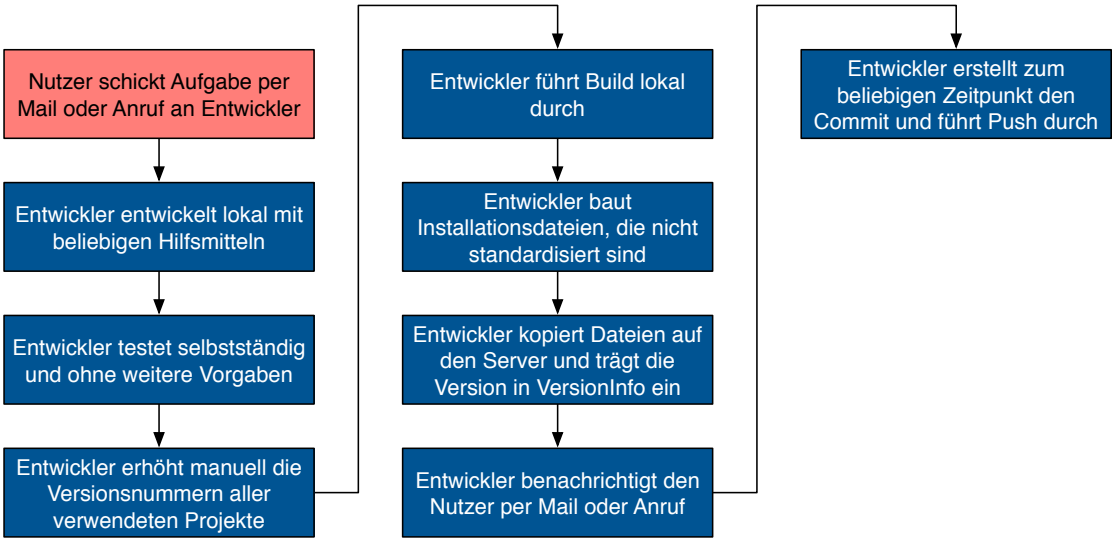


Abbildung 3: Typischer Ablauf der bisherigen Softwareentwicklung.

der Tests händisch durch die Entwickler durchgeführt, wobei ihnen selbst überlassen bleibt, ob und wie die Tests implementiert werden. Unterstützende Richtlinien oder ein standardisiertes Verfahren zur Testerstellung und –ausführung existieren für diesen Prozess nicht.

Zur Installation von Software können Benutzer in der Abteilung bereits das Werkzeug VersionInfo verwenden (vgl. Abschnitt 5.1.6). Abbildung 4 zeigt den Entwicklungsstand von VersionInfo zu Beginn der Bachelorarbeit. VersionInfo bietet bereits eine Übersicht über die Software, die in der Abteilung eingesetzt wird. Es wird außerdem dargestellt, ob die aktuell installierte Version einer Software die neueste verfügbare Version ist oder eine Aktualisierung empfohlen wird. Falls die Aktualisierung einer Software gewünscht ist, haben die Benutzer direkt in VersionInfo die Möglichkeit, die Installation der neuen Version zu starten. Auch der Start der eingetragenen Software kann über VersionInfo erfolgen.

3.2 Rahmenvorgaben

Die Softwareentwicklung der Abteilung AE-SAS unterliegt einigen Anforderungen, die aus dem Arbeitsumfeld hervorgehen. Einen Beitrag liefern die Vorgaben des DLRs, die auch für die Abteilung Gültigkeit haben. Hinzu kommen Vorgaben aus der Abteilung selbst.

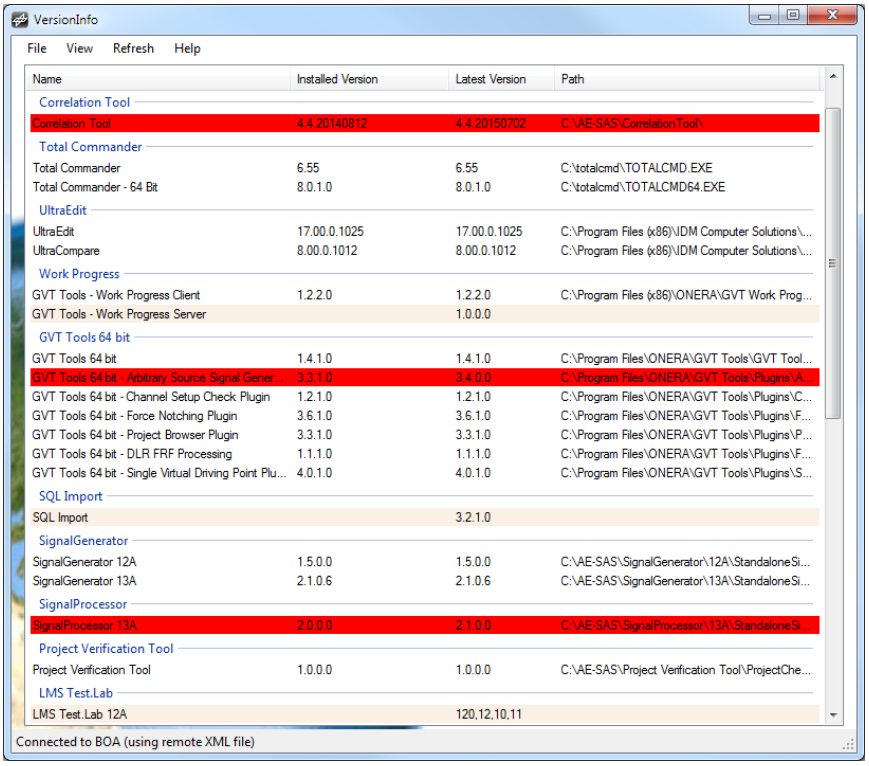


Abbildung 4: Die Oberfläche von VersionInfo zu Beginn der Bachelorarbeit.

3.2.1 Vorgaben im Deutschen Zentrum für Luft- und Raumfahrt

Im DLR werden die Vorgaben an das Qualitätsmanagement im Qualitätsmanagement-Handbuch (QMH) des QM-Rahmensystems [Deu08a] festgehalten. Das QMH entspricht den Anforderungen der Norm DIN EN ISO 9001:2008 [DIN08]. Der beschreibende Teil des QMHs beschäftigt sich primär mit der Aufgabenverteilung innerhalb der Organisation und stellt grobe Vorgaben an die Prozesse im Qualitätsmanagementsystem.

Spezifisch auf die Softwareentwicklung zugeschnittene Richtlinien zur Qualitätssicherung sind in den Rahmenrichtlinien Software-Engineering [Deu08b] des QMHs zu finden. In ihnen wird jedoch lediglich beschrieben, dass die Softwareentwicklung nach den DLR Software-Standards ablaufen soll.

Die DLR Software-Standards [Deu15b] sind Teil des DLR Software-Katalogs [Deu15a]. Der Katalog hat den Zweck, Informationen über sämtliche, im DLR entwickelte Software zu sammeln und Standards für die Entwicklung des jeweiligen Produkts an die entwickelnde

Organisationseinheit bereitzustellen. Die Software-Standards wurden zu Beginn der Bachelorarbeit für die in der Abteilung AE-SAS entwickelte Software generiert. Die darin enthaltenen Qualitätssicherungsmaßnahmen sind für die Softwareentwicklung der Abteilung anzuwenden. Die für diese Bachelorarbeit wichtigen Qualitätssicherungsmaßnahmen sind (aus [Deu15b]):

- Verteilung der Aufgaben und Verantwortlichkeiten in der Organisationseinheit
- Dokumentation des Produkts, in Form von
  - Entwicklerdokumentationen, die die Architektur und den Quellcode beschreiben
  - Nutzerhandbüchern
- Festlegung von Codier- und Kommentarstandards
- Durchführung von Tests der Software, darunter
  - Komponententests
  - Systemtests
  - Akzeptanztests
- Nutzung von Metriken der Software als Qualitätsmerkmale, beispielsweise die Testabdeckung
- Aufzeichnung und Überwachung von Problemen und Änderungsanforderungen
- Standardisiertes Verfahren zum Softwarekonfigurations- und Releasemanagement
- Einsatz von Werkzeugen zur Umsetzung der genannten Qualitätssicherungsmaßnahmen

3.2.2 Vorgaben in der Abteilung

Hauptvorgabe seitens der Abteilung ist es, verlässliche und funktionsfähige Software verfügbar zu haben, sobald diese benötigt wird. Wenn die Abteilung AE-SAS beispielsweise zu einer Messkampagne für einen Standschwingungsversuch angefordert wird, ist es notwendig, dass die dort einzusetzende Software durchgängig funktionsfähig ist. Verzögerungen, die durch fehlerhafte Software verursacht werden, können hohe Kosten zur Folge haben und die Reputation der Abteilung senken. Auch wenn vollständig fehlerfreie Software unmöglich zu garantieren ist, muss es ein wichtiges Ziel sein, die Stabilität und Qualität der Software hoch und die Anzahl auftretender Fehler klein zu halten (vgl. [Int11, S. 14]).

Das Auftreten eines Fehlers während einer Messkampagne ist auch mit einem Qualitätskonzept nie vollständig auszuschließen. Auch in diesem Fall muss ein Entwicklungsprozess anwendbar

sein, mit dem der Fehler und seine Ursache in der Software gefunden und behoben sowie eine neue Softwareversion erstellt werden kann. Es sind während der Messkampagnen jedoch Einschränkungen zu beachten: Die Verfügbarkeit der gleichen Infrastruktur, wie sie innerhalb der gewöhnlichen Umgebung im Institut AE zu finden ist, ist nur selten gegeben. Es ist häufig der Fall, dass an den Orten der Messkampagnen keine Internetverbindung vorhanden ist. Dies hat zur Folge, dass kein Zugriff auf Computersysteme im Institut möglich ist. Sämtliche, im Notfall zur Fehlerbehebung benötigten Systeme, müssen aus diesem Grund zu den Versuchen mitgenommen werden. Anzumerken ist, dass nicht alle Systeme aus dem Institut entfernt und mitgenommen werden können. Auch unter diesen Umständen ist sicherzustellen, dass die Entwickler auf den Messkampagnen mit ihren Entwicklungen fortfahren und sie nach Abschluss der Messungen in den alltäglichen Entwicklungsprozess integrieren können.

Bei der Software, die für die Durchführung der Standschwingungsversuche benötigt wird, handelt es sich um Client-Software, die auf den Microsoft Windows-Systemen der Abteilung lauffähig sein muss. Neben der Stabilität der Software ist auch ihre Funktionalität von großer Bedeutung. Da keine industrielle Software mit der benötigten Funktionalität verfügbar ist, wird die Entwicklung eigener Software überhaupt erst notwendig. Es kommen Methoden und Verfahren zum Einsatz, die zum Teil noch experimentell und in kommerzieller Software bisher nicht zu finden sind. Neue Anforderungen an die Funktionalität entstehen meist bei der alltäglichen Arbeit mit der Software oder während der Vorbereitungen auf einen größeren Versuch.

Die Anforderungen auf Grundlage der Aufgaben der Abteilung werden ergänzt durch Vorgaben der internen Struktur der Abteilung. Für die Entwicklung von Software für Messkampagnen ist ein kleines Entwicklerteam von ca. vier Personen zuständig. Dieses wird unterstützt durch die Wissenschaftler der Abteilung, die sich mit der Weiterentwicklung der eingesetzten Methoden und Erforschung neuer Verfahren beschäftigen. Anforderungen zu der Funktionalität stammen zumeist von den Wissenschaftlern, die die entwickelte Software während der Versuche einsetzen. Manuelle Tester für die Software sind, da keine exklusiven Tester in der Abteilung vorhanden sind, in der Menge der Entwickler und Wissenschaftler zu suchen.

3.3 Zielsetzungen für das Qualitätskonzept

Wie in Abschnitt 1.1 beschrieben ist es das Ziel dieser Arbeit, ein Qualitätskonzept für die Softwareentwicklungsprozesse in der Abteilung AE-SAS aufzustellen. Die Anforderungen an dieses Qualitätskonzept werden im vorherigen Abschnitt 3.2 erläutert. Vorgaben, die aus der Abteilung heraus entstehen, werden durch Vorgaben des DLRs ergänzt. Die im Qualitätsmanagement-Handbuch des DLRs geforderten Qualitätssicherungsmaßnahmen sollten im Rahmen des Qualitätskonzepts umgesetzt werden.

Der bisherige Entwicklungsprozess kann viele der Anforderungen nicht ausreichend erfüllen. Für die meisten geforderten Maßnahmen ist kein Konzept vorhanden, wodurch zumeist auch eine Umsetzung fehlt. Aus diesem Grund ist es ein wichtiges Ziel dieser Arbeit, den Entwicklungsprozess derart zu gestalten, dass er die Anforderungen sinnvoll aufgreift und umsetzt. Für diesen Zweck wird in Kapitel 4 das Konzept eines verbesserten Entwicklungsprozesses vorgestellt. Um das Konzept durch eine konkrete Umsetzung zu ergänzen, werden in Kapitel 5 Softwarewerkzeuge zur Durchführung des Entwicklungsprozesses ausgewählt. Im zweiten Teil des Kapitels folgt die Darstellung der Umsetzung an einem Entwicklungsprojekt der Abteilung AE-SAS. Der so entwickelte Prozess dient als Qualitätskonzept zur Entwicklung von Software in dem wissenschaftlichen Umfeld der Abteilung.

Damit die Softwareentwickler Unterstützung bei der Verwendung des konzipierten und umgesetzten Entwicklungsprozesses erhalten, sind Richtlinien zur Softwareentwicklung entworfen worden. Sie haben den Zweck, Entwicklern zu jedem Schritt des Prozesses Informationen bereitzustellen und mit Handlungsvorschriften einen geregelten Ablauf zu garantieren. Sie ergänzen dadurch das Software-Qualitätskonzept um ein, während der praktischen Umsetzung zu benutzendes, Dokument. Um die Richtlinien immer auf einem aktuellen Stand zu halten, handelt es sich bei ihnen um ein kontinuierlich anzupassendes Dokument. Sollten während der Umsetzung Unklarheiten, Lücken oder Fehler in den Richtlinien auffallen, müssen diese im Dokument korrigiert werden. Ebenso muss das Dokument fortlaufend an mögliche Veränderungen des Entwicklungsprozesses angepasst werden.

4 Konzept eines verbesserten Entwicklungsprozesses

Auf Basis der in Kapitel 3 beschriebenen Zielsetzungen, Vorgaben und der bestehenden Umsetzung für die Entwicklungsprozesse von Software in der Abteilung AE-SAS ist es eine grundlegende Aufgabe dieser Bachelorarbeit, ein Konzept für einen neuen, verbesserten Prozess zur Softwareentwicklung in der Abteilung zu erarbeiten. Dieser stellt einen essentiellen Bestandteil des Qualitätskonzept für die Softwareentwicklung dar. Der Prozess ist als mehrstufiger, kontinuierlich ausführbarer Prozess konzipiert, der durch den Einsatz verschiedener Softwarewerkzeuge unterstützt wird. Ein Vorgehen nach Art der kontinuierlichen Integration (vgl. Abschnitt 2.3) ist dafür vorgesehen. Abbildung 5 stellt den konzipierten Prozess grafisch dar. Die Softwareentwicklung erfolgt in einem Kreislauf, bei dem die Abarbeitung einzelner Aufgaben im Vordergrund steht.

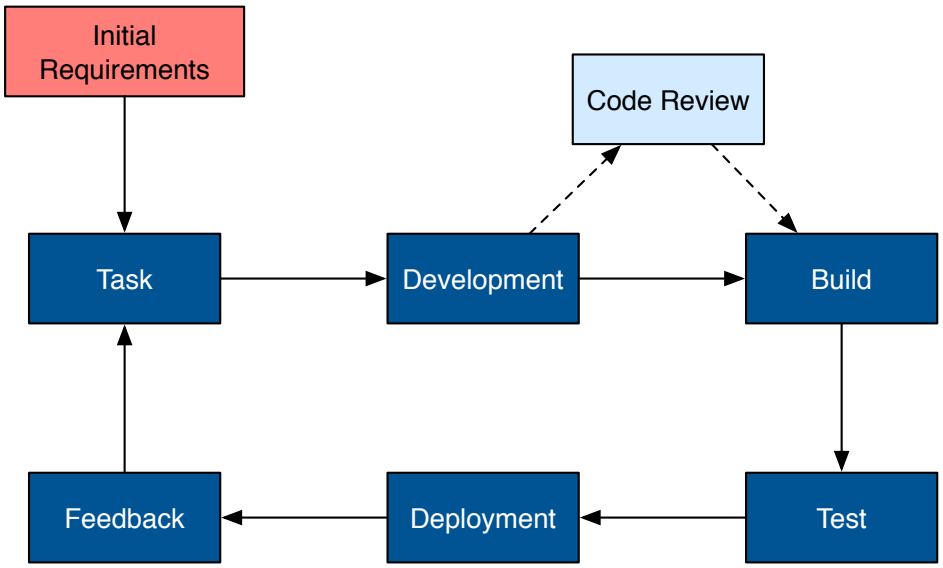


Abbildung 5: Ablauf des konzipierten Entwicklungsprozesses.

Die Entwicklung beginnt mit der Erstellung von Aufgaben (*Tasks*) auf Basis der initialen Anforderungen (*Initial Requirements*) an die zu entwickelnde Software. Vorhandene Aufgaben werden separat in der Softwareentwicklung abgearbeitet. Anschließend kann ein Code Review

(vgl. Abschnitt 4.4) stattfinden, welches für bestimmte Projekte auch als verpflichtend festgelegt werden kann. Nach der Entwicklung und dem eventuellen Code Review kann die Erstellung der Anwendung im Buildprozess durchgeführt werden. Im Erfolgsfall schließen sich Tests der kompilierten Anwendung an. Falls alle Tests fehlerfrei ausgeführt werden, kann die Verteilung der Software (*Deployment*) beginnen. Sollte es zu Fehlern während des Builds oder der Tests kommen, erhalten die Entwickler eine Benachrichtigung. Die Anwender sollen zu jedem Zeitpunkt die Möglichkeit haben, Feedback bezüglich der an sie verteilten Software zu geben. Formen des Feedbacks können unter anderem Fehlerberichte (im Folgenden als *Bug Reports* bezeichnet) oder Anfragen zu weiterer Funktionalität (im Folgenden als *Feature Requests* bezeichnet) sein. Durch die ständige Möglichkeit zur Rückmeldung an die Entwickler wird eine höhere Akzeptanz des Produktes unter den Nutzern und eine höhere resultierende Qualität der Software garantiert (vgl. [GL12, Kap. 10.1]).

Der Prozess ist angelehnt an das Kanban-Vorgehensmodell (vgl. Abschnitt 2.2.2). Er stellt, wie Kanban, Aufgaben in den Vordergrund. Die Aufgaben, wie sie bei Kanban als Karten aufgefasst werden, sollen durch den Entwicklungsprozess bewegt werden und sich auf einem Kanban-Board anzeigen lassen. Neue Aufgaben-Karten können kontinuierlich durch Feedback erzeugt werden. Außerdem werden die Kanban-Techniken Testautomatisierung, Code Reviews und kontinuierliche Integration umgesetzt. Kanban wurde dem ebenfalls vorgestellten Vorgehensmodell Scrum vorgezogen, da es für die Entwicklungsarbeit der Abteilung einfacher und mit geringerem Änderungsaufwand der Arbeitsabläufe umzusetzen ist. Bei den vier Entwicklern stellt sich eine Rollenverteilung nach Scrum als schwierig heraus, außerdem würde die Einführung der Sprints als Iterationen einen erhöhten Planungsaufwand mit sich bringen.

Über die Umsetzung einer kontinuierlichen Integration hinaus ist der Entwicklungsprozess so konzipiert, dass er Aspekte der kontinuierlichen Auslieferung aufgreift. Dabei liegt besonderer Fokus auf den Prinzipien der kontinuierlichen Auslieferung (vgl. Abschnitt 2.4 und [HF11]). Im Gegensatz zur klassischen kontinuierlichen Auslieferung wird in dem konzipierten Entwicklungsprozess das Build- und Testverfahren einfach gehalten. In [HF11, S. 105 ff.] wird der Prozess der kontinuierlichen Auslieferung als mehrstufiges Verfahren vorgestellt. Auf den Buildprozess folgen nacheinander Unittests, automatisierte und manuelle Akzeptanztests und die Verteilung (vgl. Abschnitt 2.4). Um den gesamten Entwicklungsprozess entsprechend den Vorgaben und dem kleinen Entwicklerteam nicht länger und komplexer zu gestalten, als dies für die Zielerreichung notwendig wäre, wurde auf ein mehrstufiges Verfahren verzichtet (vgl. Abschnitte 3.2 und 3.3).

4.1 Identifizierung der initialen Anforderungen

Für jedes Produkt bzw. jede zu entwickelnde Anwendung sollte vor Beginn der Entwicklung eine Analyse der Anforderungen durchgeführt werden. Basis dafür ist die Projektdefinition und die Projektplanung, die zu Projektbeginn erstellt werden müssen. Zu der Projektplanung zählen Dokumente wie Meilensteinpläne und Qualitätssicherungspläne, die zu einer formalen Projektdefinition gehören (vgl. [BK13, S. 71 ff.]). Während der Entwicklung können sich die Beteiligten des Projektes auf diese Dokumente beziehen. Der Vergleich zwischen Soll- und Ist-Ständen kann zu jeder Zeit eine Vorstellung davon geben, ob Steuerungsmaßnahmen notwendig sind und Vorgaben oder Anforderungen angepasst werden müssen.

Im Rahmen der Informationssicherheit ist eine Schutzbedarfsfeststellung für jedes Produkt sinnvoll. Üblich ist eine Einordnung in eine von drei Kategorien nach BSI-Standard 100-2 [Bun08], die in Tabelle 1 dargestellt sind. Für die Einordnung ist es erforderlich, eine persönliche Einschätzung der Benutzer des Produktes zu erfragen. Die Einordnung erfolgt für jeden der Grundwerte nach dem BSI-Grundschutz:

- Vertraulichkeit
- Integrität
- Verfügbarkeit

Normal	Die Schadensauswirkungen sind begrenzt und überschaubar.
Hoch	Die Schadensauswirkungen können beträchtlich sein.
Sehr hoch	Die Schadensauswirkungen können ein existentiell bedrohliches, katastrophales Ausmaß erreichen.

Tabelle 1: Schutzbedarfskategorien aus BSI-Standard 100-2 [Bun08, S. 49]

Eine in der Abteilung AE-SAS entwickelte Software dient häufig der Durchführung mehrerer Geschäftsprozesse. Es ist daher nicht auszuschließen, dass die Aufgaben eines Produktes unterschiedlichen Kategorien zugewiesen werden. Da besonders die geringe Anzahl an Entwicklern in der Abteilung und deren Verfügbarkeit ein Problem darstellen können, ist zu empfehlen, dass ein Softwareprodukt für seine verschiedenen Geschäftsprozesse tatsächlich unterschiedliche Schutzbedarfskategorien zugewiesen bekommt (vgl. [Bun08, S. 53]). Dadurch wird kritischen Geschäftsprozessen eine höhere Schutzbedarfskategorie zugeteilt als unkritischen Geschäftsprozessen.

Aus den Schutzbedarfskategorien kann im Verlauf der Entwicklung abgeleitet werden, welche Maßnahmen zur Sicherung und Wahrung der Qualität notwendig sind. Hierunter fallen beispielsweise die in Abschnitt 4.4 erwähnten Code Reviews oder die Art der zu verwendenden Tests.

Nach der Schutzbedarfsfeststellung besteht die Möglichkeit, eine detaillierte Risikoeinschätzung für das Produkt durchzuführen. Sie kann, nach ISO / IEC Standard 27000 [DIN11, S. 11], als Prozess aus Risikoanalyse und anschließender Risikobewertung angesehen werden. Die Risikoanalyse beinhaltet die Risikoidentifizierung und die Risikoabschätzung, d.h. sie identifiziert die Ursachen und die Höhe des Risikos. Die Risikobewertung beurteilt, wie die Bedeutung des betreffenden Risikos auf die Organisation ist (vgl. [KRSW08, S. 25 ff.]). Risikoanalysen können einen hohen Aufwand mit sich bringen, weshalb abzuwägen ist, ob sie durchgeführt werden. Allerdings werden sie wichtiger, je höher der festgestellte Schutzbedarf ist. Aus den Resultaten der Risikoeinschätzungen können Maßnahmen zur Risikominimierung abgeleitet werden, die als Aufgaben in die Entwicklung der Software mit einbezogen werden. Sie tragen im Folgenden, wie die Schutzbedarfskategorien, zur Auswahl der Qualitätssicherungsmaßnahmen bei.

4.2 Aufgabenverwaltung

Die Softwareentwickler sollen im Entwicklungsprozess ein Kollaborations- und Projektmanagementwerkzeug zur Zusammenarbeit verwenden. In diesem werden die gemeinsamen Softwareprojekte erfasst und verwaltet (vgl. [BK13, S. 356]). Ein wichtiger Anwendungszweck dieses Werkzeugs ist die Organisation von Aufgaben. Über das Werkzeug sollen die Entwickler eine Übersicht über die noch ausstehenden Aufgaben und Probleme bekommen und den Stand ihrer Arbeit mit dem Team teilen können. Dazu sollte es die Möglichkeit geben, den Aufgaben einen Zustand und eine bearbeitende Person zuzuordnen.

Um die Nutzer des Kollaborationswerkzeugs immer auf dem aktuellen Stand zu halten, ist eine Benachrichtigungsfunktion notwendig. Idealerweise sollte es verschiedene Abstufungen geben, in denen die Benachrichtigungen klassifiziert werden. Für bestimmte Aktionen, denen wenig Aufmerksamkeit geschenkt werden soll, sollten sich die Benachrichtigungen deaktivieren lassen. Beachtenswerte Mitteilungen, die keine Dringlichkeit haben, könnten beispielsweise als Meldung in der Oberfläche des Werkzeugs angezeigt werden. Für den Nutzer wichtige Mitteilungen, die ihn auch erreichen müssen, wenn er gerade nicht aktiv das Werkzeug

benutzt, sollten ihm über einen anderen Kommunikationskanal, vorzugsweise per E-Mail, zugestellt werden.

In das Kollaborationswerkzeug sollte außerdem eine Anbindung an die Repositories zur Quellcodeverwaltung der Projekte integriert sein (vgl. Abschnitt 4.3.1). Hierdurch entstehen weitere Nutzungsmöglichkeiten:

- Zentrale Entwicklungsplattform: Wenn Projektdaten direkt aus dem Werkzeug abrufbar sind, muss während der Entwicklung seltener zwischen verschiedener Software gewechselt werden.
- Verknüpfung zwischen Aufgaben und Quellcode: Bezieht sich der Quellcode eines Commits direkt auf eine Aufgabe, so sollte er mit der Aufgabe verknüpft werden. Nutzer des Systems sehen so direkt, welche Inhalte zusammen gehören und müssen nicht jeweils manuell versuchen, Aufgaben und Code zusammenzufügen.
- Code Reviews: Code Reviews, wie sie in Abschnitt 4.4 beschrieben werden, sollten direkt im Kollaborationswerkzeug durchführbar sein.

Ferner ist die Überwachung des Projektzustands eine wichtige Funktionalität eines Projektmanagementwerkzeugs. Aussagekraft über den Entwicklungsstand kann das Resultat von Builds und Tests haben. Schlagen sie fehl, kann das Produkt nicht ausgeliefert werden, bis die zugrunde liegenden Fehler behoben wurden. Auch wenn der Buildstatus keine Aussage über die Korrektheit einer Software machen kann, sollte er für alle Entwickler aus dem Kollaborationswerkzeug ersichtlich sein. Somit erhalten die Entwickler die Möglichkeit, sich auf einfachem Weg über den Status des Entwicklungsprozesses zu informieren. Hierdurch wird auch zur Umsetzung des Prinzips „Everybody Is Responsible for the Delivery Process“ der kontinuierlichen Auslieferung nach [HF11] beigetragen (vgl. Abschnitt 2.4).

4.3 Entwicklung

Die Erledigung von Aufgaben erfolgt, solange es sich um Softwareentwicklungsaufgaben handelt, im Schritt der Entwicklung. In diesem Schritt sind zum einen die zur Abarbeitung der Aufgabe notwendigen Maßnahmen enthalten, zum anderen ist auch die Erstellung oder Anpassung von Tests vorgesehen, soweit dazu Notwendigkeit besteht. Zur Entwicklung von qualitativ hochwertigem Quellcode sind einige Hilfsmittel unerlässlich. Deshalb sieht das Konzept die Verwendung eines Versionsverwaltungssystems zur Quellcodeverwaltung und

einer integrierten Entwicklungsumgebung (IDE) als Werkzeug für die Entwickler vor. Die Abschnitte 4.3.1 und 4.3.2 beschreiben die für diese Hilfsmittel festgelegten Anforderungen. Des Weiteren ist die Dokumentation der Anwendungen bereits während der Entwicklung ein Thema, worauf in Abschnitt 4.3.3 eingegangen wird.

4.3.1 Anforderungen an die Quellcodeverwaltung

Zur Verwaltung von Projektdateien wie Quellcode, Bibliotheken und Tests sieht das Konzept ein Versionsverwaltungssystem vor. Die Aufbewahrung der Dateien erfolgt in Repositories innerhalb des Versionsverwaltungssystems. Der Entwicklungsverlauf des Projektes kann dank Speicherung aller vergangenen Versionsstände zu jeder Zeit zurückverfolgt werden. Dabei ist die persistente Speicherung der Änderungen und einiger Metadaten vorgesehen, die die Änderungen in einen Kontext bringen. Bedeutung unter den Metadaten haben vor allem der Autor, der Zeitpunkt und der Grund einer Änderung. Mit der Verwendung eines Versionsverwaltungssystem wird auch das Prinzip „Keep Everything in Version Control“ der kontinuierlichen Auslieferung nach [HF11] umgesetzt, welches eine Versionskontrolle für sämtliche notwendige Projektdaten fordert (vgl. Abschnitt 2.4).

Die Vorteile von Versionsverwaltungssystemen zeigen sich besonders bei der Zusammenarbeit von mehreren Projektbeteiligten. Sie sind dafür konzipiert, dass die Nutzer gleichzeitig unterschiedliche Änderungen vornehmen und sie anschließend zusammenführen. Die Software unterstützt Anwender beim Auffinden und Beheben von Konflikten zwischen den abweichend veränderten Dateien (vgl. [O'S09, Kap. 1.1]).

Vorgesehen ist, dass das Versionsverwaltungssystem verteilt aufgebaut ist. Dadurch ist die Nutzung von mehreren Repositories, die sich beliebig untereinander synchronisieren, möglich. Vorteile bieten verteilte Versionsverwaltungen auch dann, wenn ein zentrales Repository eingesetzt wird. Bei einer verteilten Versionsverwaltung können Änderungen lokal gesichert werden, es muss nicht für jede Änderung eine Verbindung zu einem zentralen Server aufgebaut werden (vgl. [O'S09, Kap. 1.5]). Außerdem kann bei fehlender Erreichbarkeit des zentralen Servers, wie sie während auswärtiger Messkampagnen auftreten kann, ein Repository auf einem der verfügbaren Computer angelegt werden. Dieses kann zur Zusammenarbeit von mitgereisten Entwicklern verwendet werden. Bei Rückkehr der Entwickler oder Verfügbarkeit einer Verbindung können der zentrale Server und das mitgenommene Repository synchronisiert werden. Abbildung 6 illustriert diesen Aufbau. Solange die Entwickler Zugriff auf das zentrale

Repository (blau) haben, können sie ihre lokalen Repositories (weiß) mit diesem per Pull und Push synchronisieren. Die Entwickler arbeiten jeweils auf ihrem lokalen Repository. Falls das zentrale Repository nicht verfügbar ist, wie es beispielsweise bei Aufenthalten außerhalb des Instituts vorkommen kann, können die Entwickler ein portables Repository (grün) zur Synchronisation verwenden. Sobald das zentrale Repository wieder verfügbar ist, kann eine Synchronisation mit dem portablen Repository stattfinden.

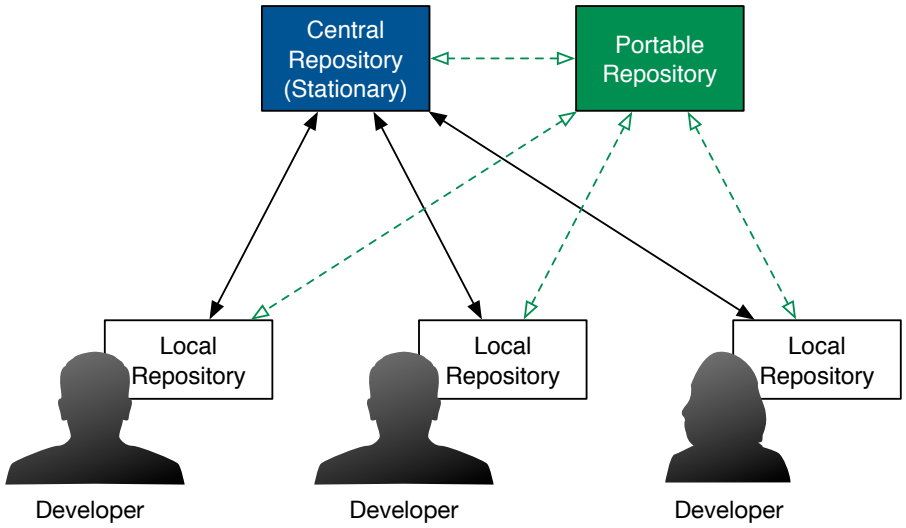


Abbildung 6: Mögliche Struktur eines verteilten Versionsverwaltungssystems.

4.3.2 Anforderungen an die Entwicklungsumgebung

Das Konzept zum Softwareentwicklungsprozess sieht die Verwendung einer integrierten Entwicklungsumgebung vor, die die Entwickler bei ihrer täglichen Arbeit umfassend unterstützt. Besondere Bedeutung bei der Auswahl einer Entwicklungsumgebung haben folgende, essentielle Funktionen und Eigenschaften:

- Unterstützung für C# und das .NET Framework: Da die Anwendungen der Abteilung AE-SAS hauptsächlich in der Programmiersprache C# geschrieben werden, ist die Unterstützung der Integrated Development Environment (IDE) für sie und für das .NET-Framework unerlässlich.
- Syntaxvervollständigung und -prüfung: Besonders für C#, aber auch für Sprachen wie XML oder XAML, welche während der Entwicklung häufig verwendet werden, sollte eine Prüfung und Vervollständigung des Quellcodes stattfinden. Die Entwickler erhalten hierdurch direkt während des Schreibens von Codes Informationen und Rückmeldungen zu den verwendeten Befehlen und Programmierkonstrukten.
- Designer für grafische Oberflächen: Viele der Anwendungen, die in der Abteilung entwickelt werden, stellen eine grafische Benutzeroberfläche bereit. Da die Erstellung von ergonomischen Oberflächen rein in Programmcode, ohne die Benutzung eines Graphical User Interface (GUI) Designers, äußerst zeitaufwendig und kompliziert wäre, ergibt sich die Notwendigkeit an ein solches Werkzeug. Um als Entwickler nicht ständig zwischen IDE und GUI Designer wechseln zu müssen, sollte die IDE diese Funktionalität integriert haben.
- Debugger: Ein Debugger ist ein Werkzeug zum Auffinden von Fehlern zu Laufzeit von Programmen. Während die Entwickler manuelle Tests durchführen, kann das getestete Programm mit einem Debugger überwacht und zum Teil verändert werden.
- Testerstellung, -ausführung und -verwaltung: Die IDE sollte die Erstellung und Nutzung von Testfällen ermöglichen. Unter den Testarten ist insbesondere der Unittest wichtig. Des Weiteren bietet sich ein automatisierbarer GUI-Test an, um auch die Bedienung der Software im Rahmen eines System- oder Akzeptanztests nach bestimmten Kriterien zu überprüfen.
- Bereitstellung von Refactoring-Werkzeugen
- Erstellung von Metriken

4.3.3 Anforderungen an die Dokumentation

Während der Entwicklung sollte bereits die Dokumentation der zu entwickelnden Anwendung stattfinden. Darunter sind in diesem Zusammenhang zweierlei Arten zu verstehen: Zum einen die Dokumentation für Entwickler, zum anderen die Dokumentation für Benutzer. Die Entwicklerdokumentation ist dabei ähnlich anzusehen wie die interne technische Dokumentation nach VDI-Richtlinie 4500 [Ver06, S. 6 f.], wohingegen die Benutzerdokumentation als externe technische Dokumentation anzusehen ist. Für beide Arten ist es entscheidend, dass in einem angemessenen Rahmen dokumentiert wird. Die Dokumentation muss während der Entwicklung erfolgen, damit im weiteren Prozess eine für Benutzer und Entwickler nutzbare Dokumentationsdatei erstellt werden kann. Außerdem ist die Dokumentation zur vollständigen Erledigung einer Aufgabe mit einzubeziehen.

Entscheidend bei der Entwicklerdokumentation ist, dass die anderen Softwareentwickler, sowie man selbst, den erzeugten Quellcode auch einige Zeit später noch nachvollziehen können. Es sollten darum an entscheidenden oder komplexen Stellen des Quellcodes Kommentare eingefügt werden, die der Verständlichkeit dienen. Zusätzlich ist die Funktion der einzelnen Klassen, Typen, Methoden und Parameter zu beschreiben. Es ist besonders darauf zu achten, dass der Quellcode so verständlich ist, dass er von anderen Entwicklern verstanden und nicht direkt bei einem Code Review abgewiesen wird (vgl. Abschnitt 4.4).

Einen Teil der Entwicklerdokumentation stellen auch die Commit-Nachrichten dar. Sie sind essentiell, um den Zweck eines Commits schnell zu erfassen, und müssen deshalb von den Entwicklern genutzt werden. Idealerweise sollten die Commit-Nachrichten eine direkte Zuordnung zu einer Aufgabe zulassen. Die konkrete Zuordnung fällt in den Aufgabenbereich des Kollaborationswerkzeugs.

Die Benutzerdokumentation hat den Zweck, die Anwender bei der Nutzung der Software zu unterstützen. Sie kann How-Tos, Schritt-für-Schritt-Erklärungen, Erklärungen zu den Funktionalitäten oder weitere hilfreiche Informationen enthalten. Der Umfang der Benutzerdokumentation ist für jedes Projekt unter Einbeziehung der vorgesehenen Benutzer abzustimmen.

Für beide Arten der Dokumentation ist festzuhalten, dass sie nicht vollständig, sondern nur zu einem gewissen Teil, von Softwarewerkzeugen automatisiert erstellt werden können. Somit liegt es in der Verantwortung der Entwickler, ihr Wissen und ihre Überlegungen einzubringen. Die Dokumentation erfordert damit ein hohes Maß an Disziplin und sollte deshalb in den Richtlinien angesprochen werden (vgl. [BK13, S. 300]).

4.4 Code Review

Einigkeit herrscht in der Fachliteratur zum Thema „Fehler in Software“ darüber, dass die Kosten von spät gefundenen Fehlern deutlich höher sind als die von früh gefundenen Fehlern (vgl. [Fag76, S. 186 ff.], [BW05, S. 16] [Kan03, S. 137 ff.], [ABL89, S. 34 f.]). Weite Verbreitung hat ebenfalls die Ansicht, dass Inspektionen des Codes zu einer früheren Auffindung von Fehlern führen und die Codequalität damit angehoben wird (vgl. [ABL89, S. 31]).

Als Code Reviews werden Inspektionen von Teilen des Quellcodes bezeichnet (vgl. [BK13, S. 141]). Sie dienen dazu, Fehler und Verbesserungsmöglichkeiten aufzudecken (vgl. [KP09, S. 535 ff.]). Mit Hilfe des Kollaborationswerkzeugs sollen die Entwickler in der Lage sein, Code Reviews durchzuführen, um den positiven Effekt der Reviews auf die Qualität des Codes nutzen zu können. Das Kollaborationswerkzeug soll dafür sowohl Code Reviews vor (*Pre-Push*) und nach (*Post-Push*) dem Laden des Codes auf das Repository unterstützen. Während der Code Reviews ist es die Aufgabe der Entwickler, gefundene Fehler oder Unklarheiten anzumerken und sie an den Autor des Quellcodes zurückzumelden. Dieser hat im Anschluss die Möglichkeit, Nachbesserungen vorzunehmen.

Für die Durchführung ist ein einfacher Zugriff auf die am Quellcode und den Projektdateien vorgenommenen Änderungen, die in ein Commit verpackt werden, notwendig. Je nachdem, wie hoch die Qualitätsanforderungen an die entwickelte Anwendung sind, muss entschieden werden, ob und wie die Code Reviews durchgeführt werden. In [DHJ14] wird ein Code Review besonders bei Sicherheitsfunktionalität empfohlen. Die Entscheidung zur Notwendigkeit von Code Reviews sollte unter Beachtung der Risikoeinschätzung nach Abschnitt 4.1 erfolgen. Bei Anwendungen mit hohem oder sogar sehr hohem Schutzbedarf empfiehlt sich ein Code Review in jedem Fall.

Falls sich gegen ein verpflichtendes Code Review vor dem Hochladen des Quellcodes auf das zentrale Repository entschieden wird, besteht noch immer die Möglichkeit, das Code Review später durchzuführen. Dabei ist wieder zu beachten, dass dies dafür sorgen kann, dass Fehler später gefunden werden als bei einem früheren Code Review. Außerdem muss die Durchführung von Code Reviews, falls kein systemseitiger Zwang zu Pre-Push Code Reviews besteht, durch Richtlinien sichergestellt werden. Selbst wenn eine automatische Zuweisung von Commits zum Code Review an einen anderen Entwickler stattfindet, obliegt es noch immer der Entscheidung dieses Entwicklers, ob und wann er das Code Review durchführt.

Nach der Frage, ob Code Reviews beim aktuellen Projekt durchgeführt werden sollen, stellen sich auch die Fragen, wie umfangreich die Reviews ausfallen und wie viele Personen involviert sein sollen. Der Aufwand von Code Reviews, in der Form eines Peer-Reviews, ist, trotz guter Ergebnisse, meist überschaubar (vgl. [DHJ14]). Untersuchungen zeigen, dass es ein Optimum an Arbeitszeit gibt, die in den Prozess einer Inspektion von Quellcode investiert wird (vgl. [McC01]). Bei zu wenig investierter Zeit werden zu viele Fehler übersehen, bei zu viel investierter Zeit werden nicht proportional mehr Fehler gefunden, die Inspektion ist nicht mehr effektiv. In der Literatur werden meist Geschwindigkeiten von 100 (vgl. [McC01]) bis 200 Zeilen pro Stunde (vgl. [KP09]) als Optimum bezeichnet, wobei sich die Angaben auf Quellcodezeilen ohne Kommentare und Leerzeilen beziehen. Diese Werte sollten lediglich als grobe Richtwerte betrachtet werden und können je nach Technologie, Projekt und Entwickler variieren. Mit zunehmender Erfahrung auf dem Gebiet der Code Reviews im Entwicklerteam können bessere Richtwerte für den Prozess gefunden werden.

Da das Entwicklerteam in der Abteilung AE-SAS mit seinen vier Personen verhältnismäßig klein ist, sollten in den meisten Fällen Code Reviews von einem einzigen Entwickler durchgeführt werden. Dieser Entwickler sollte nicht der Autor der Quellcodeänderungen sein. Bei kritischen Codeteilen können mehrere Entwickler einbezogen werden. In diesem Fall muss abgestimmt werden, ob die Inspektionen getrennt voneinander stattfinden, ein Review-Treffen einberufen oder nacheinander inspiziert werden soll.

Neben dem Review des Quellcodes der Anwendung ist auch ein Review der Dokumentation vorzusehen. Bei diesem ist ebenfalls wieder abzuwägen, wie viel Aufwand bei dem betreffenden Entwicklungsprojekt für ein derartiges Review zu rechtfertigen ist. Da eine qualitativ schlechte oder fehlerhafte Dokumentation wenig Nutzen mit sich bringt, sollte ein Code Review des Dokumentationsquelltextes angemessen zum Umfang der Dokumentation stattfinden.

4.5 Build

Während des Buildprozesses erfolgt die Erstellung der Software. Eine erledigte Entwicklungsaufgabe, deren Quellcode auf das Repository geladen wurde, stößt automatisch einen Build an. Zu der Erstellung der Software gehören zum einen die Schritte der Kompilierung des Quellcodes zur Erzeugung einer ausführbaren Datei, zum anderen die Erzeugung einer installierbaren Datei. Außerdem muss die Dokumentation erstellt werden. Abbildung 7 stellt den Weg vom Quellcode zu den Installationsdateien vereinfacht dar. Im ersten Schritt erfolgt

die Erstellung der Anwendung (grün) auf Basis des Quellcodes und abhängiger Ressourcen mit einem Compiler. Anschließend kann aus der Anwendung, zusammen mit weiteren notwendigen Dateien, mit einem Installationssystem eine Installationsdatei erzeugt werden (gelb).

Die installierbaren Dateien müssen im Rahmen der Softwareverteilung von den Benutzern ausgeführt werden können, um die Software auf den von ihnen benutzten Systemen zu verwenden. Auch eine automatisierte Installation ohne Eingreifen der Benutzer sollte möglich sein.

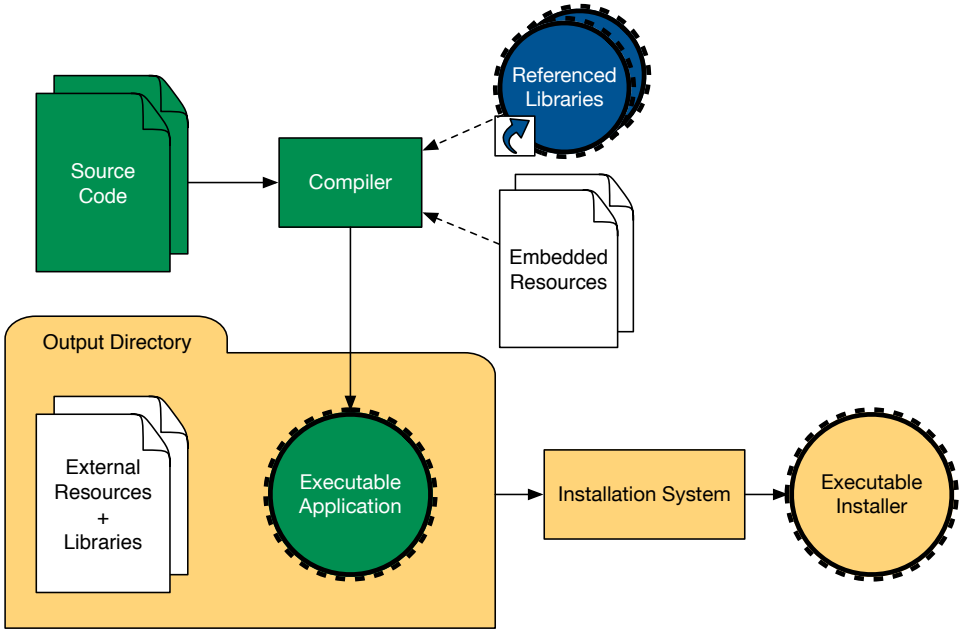


Abbildung 7: Vereinfachter Ablauf der Erstellung einer Installationsdatei aus Quellcode.

4.5.1 Erstellung der Anwendung

Die Erstellung der Anwendung soll vollautomatisch auf Basis des Quellcodes und der vorliegenden Konfigurationsdateien vorgenommen werden. Eine Anwendung kann dabei aus mehreren Codeprojekten bestehen, zwischen denen Abhängigkeiten existieren. Eine wichtige Anforderung an die Buildumgebung ist die Unterstützung des .NET-Frameworks und der Programmiersprache C#, da diese Hilfsmittel hauptsächlich zur Softwareentwicklung in der

Abteilung eingesetzt werden. Die Konfigurationsdateien haben den Zweck, die zur Erstellung benötigten Ressourcen zu definieren. Darunter fallen unter anderem der zu kompilierende Quellcode, die Ressourcen, die von der Anwendung benötigten Dateien und die notwendigen Bibliotheksreferenzen.

4.5.2 Erstellung der Tests

Während des Builds werden auch die Tests aus ihrem Quellcode erstellt. Damit die Entwickler keine weitere Programmiersprache speziell für die Erstellung von Tests lernen und nutzen müssen, sollte auch die Verwendung der Sprache C# für Tests möglich sein. Durch die Nutzung der gleichen Technologien für die Tests und die Anwendungen entsteht kein zusätzlicher Wartungsaufwand. Dies bedeutet jedoch nicht, dass Testapplikationen mit anderen Technologien ausgeschlossen sind. Ein Grund für deren Einsatz wäre beispielsweise das Aufdecken von Fehlern in gemeinsam benutzten Programmbibliotheken oder Frameworks.

Es ist nicht vorgesehen, dass die Tests von Endbenutzern ausgeführt werden. Aus diesem Grund ist es nicht notwendig, dass die Tests in den Installationsdateien (vgl. Abschnitt 4.5.4) enthalten sind. In den Installationsdateien unerwünscht sind ebenfalls die, für die Durchführung der Tests benötigten, Testdaten.

4.5.3 Erstellung der Dokumentation

Die Nutzer- und Entwicklerdokumentation wird im Build-Schritt zu einer oder mehreren Dokumentationsdateien zusammengefügt. Diese Dokumentationsdateien müssen von den Nutzern und Entwicklern gelesen werden können, wofür möglichst kein Einsatz von Zusatzsoftware notwendig ist. Denkbar sind deshalb Formate wie das Hypertext Markup Language (HTML)-Format, dass von jedem Webbrowser geöffnet werden kann, oder dass Compiled HTML Help (CHM)-Format, für welches Microsoft ein Betrachtungswerkzeug mit den Windows-Betriebssystemen mitliefert. Die fertige Dokumentation, besonders die Nutzerdokumentation, sollte mit der Installationsdatei ausgeliefert werden, um Anwendern die Möglichkeit zu geben, ohne Zugriff auf weitere Ressourcen Informationen und Hilfe zur Software zu bekommen.

Für die Erstellung der Dokumentationsdateien können als Eingabeinformationen der kommentierte Quellcode, die kompilierten Programmdateien oder dokumentationsspezifische Quelldateien verwendet werden. Des Weiteren ist es vorzusehen, dass die Commit-Nachrichten zu Änderungsprotokollen (auch als *Release Notes* bezeichnet) zusammengefasst und in die

Dokumentation integriert werden. Die Möglichkeit zur Nachverfolgung von Änderungen durch die Nutzer soll hierdurch gegeben sein.

4.5.4 Erstellung der Installationsdateien

Während des Buildprozesses werden neben den reinen Programmdateien auch Installationsdateien für die Software erzeugt. Die Installationsdateien spielen eine wichtige Rolle bei der Verteilung der Softwareprodukte (vgl. [Bal11, S. 522 f.]). Die in ihnen definierte Prozedur zur Installation wird bei jeder neuen Version einer Software ausgeführt. Wichtig ist, dass alle notwendigen Dateien einer veröffentlichten Version an ihren Bestimmungsort kopiert und bestehende Dateien, bei niedrigerer Versionsnummer, aktualisiert werden. Außerdem müssen, je nach Notwendigkeit, Verknüpfungen im Startmenü und auf dem Desktop erstellt oder erneuert werden. Zu beachten ist insgesamt, dass eventuell bestehende Installationen korrekt aktualisiert werden, damit keine Bestandteile der vorherigen Installationen auf den Computern der Anwender zurückbleiben (vgl. [Bal11, S. 523]).

Die Erstellung dieser Installationsdateien erfolgt weitgehend unabhängig von den für die Anwendung verwendeten Technologien und Programmiersprachen. Je nach verwendetem Installationssystem gibt es unterschiedliche Ausgabeformate für die Installationsdateien. Möglich ist, dass das Ergebnis eine einzelne ausführbare Datei ist. In anderen Fällen besteht die Installation aus mehreren Dateien, bei der eine oder mehrere Archivdateien die zu installierenden Ressourcen beinhalten und sich die Installation über eine zusätzlich beiliegende, ausführbare Installationsdatei starten lässt. Sofern die Installationsdateien getestet werden sollen, findet auch dieser Test innerhalb des in Abschnitt 4.6 beschriebenen Testprozesses statt.

4.6 Test

Wie bereits in Abschnitt 4.4 beschrieben ist es bei der Softwareentwicklung von Vorteil, Fehler in der Software möglichst früh zu finden. Ein Fehler, der erst nach der Verteilung der Software bemerkt wird, erfordert im Allgemeinen mehr Aufwand in der Behebung als einer, der bereits durch Tests gefunden wurde. Das Testen von Software ist eine der am häufigsten eingesetzten Methoden zur Qualitätssicherung von Software (vgl. [Hof13, S. 157]). Es hilft sicherzustellen, dass die entwickelte Software den gestellten Qualitätsansprüchen entspricht. Entscheidend ist

hierbei aber, dass Testen nur die Anwesenheit von Fehlern aufzeigen kann. Ein Test kann keinen Beweis dafür liefern, dass keinerlei Fehler in der Software vorhanden sind (vgl. [Int11, S. 14]).

Der zu investierende Aufwand, der in die Entwicklung von Tests gesteckt wird, sollte wieder abhängig von der Klassifizierung in eine Schutzbedarfsklasse ermittelt werden (vgl. Abschnitt 4.1). Anwendungsteile, deren Ausfall großen Schaden verursachen würde, sollten möglichst gut getestet werden. Hierbei empfiehlt sich ein Blick auf Metriken wie die Testabdeckung. Die Testabdeckung gibt an, welcher Anteil des Quellcodes von den ausgeführten Testfällen abgedeckt und getestet wird.

Ziel des Konzeptes ist es, dass ein möglichst großer Teil der Tests automatisiert ausgeführt wird. Das Prinzip „Automate Almost Everything“ nach [HF11] wird hiermit adressiert (vgl. Abschnitt 2.4). Da in dem kleinen Entwicklerteam der Abteilung AE-SAS, welches keine dedizierten Tester beinhaltet, jedoch nicht ausreichend Ressourcen zur vollständigen automatisierten Testabdeckung aller entwickelten Software vorhanden sind, sollte abgewogen werden, an welchen Stellen umfangreiche, automatisierte Tests notwendig sind und an welchen Stellen auf sie verzichtet werden kann. In kleineren Softwareteilen, in denen wenig automatisierte Tests benutzt werden, bietet sich als Alternative die Durchführung von manuellen, explorativen Tests an (vgl. [PJR09, Kap. II.5]).

Wichtig ist, dass an die Entwicklung des Quellcodes von Tests die gleichen Ansprüche gestellt werden wie an die Entwicklung von Anwendungsquellcode. Tests müssen, genauso wie der Programmcode auch, immer wieder angepasst werden. Wenn die Tests nun schlecht konzipiert oder schwer zu warten sind, werden sie mit der Zeit zu einer größeren Belastung. Eine Folge kann sein, dass die Tests nicht mehr verwendet werden und so ein Grundstein der Qualitätssicherung fehlt (vgl. [Mar09, Kap. 9.2]).

Alle spezifizierten Testfälle, automatisiert oder manuell, sollten zuerst von den Entwicklern während der Entwicklung ausgeführt werden, damit fehlerhafter Quellcode bereits vor dem Laden auf das Repository bemerkt wird. Sämtliche automatisierbaren Tests sollten im Anschluss an jeden Build des Buildsystems ausgeführt werden, damit mögliche Fehler im Quellcode der Anwendung oder der Tests zu diesem Zeitpunkt auffallen. Testfälle, die auf dem Buildsystem nicht automatisiert durchgeführt werden können, dürfen nicht für die Ausführung im Rahmen eines automatisierten Builds ausgewählt werden. Andernfalls hätten sie ein ständiges Fehlschlagen der Tests auf dem Buildsystem zur Folge.

Die verschiedenen Ebenen, auf denen getestet werden kann, stellt Abbildung 8 dar. In der untersten Ebene sind die Tests mit den kleinsten zu testenden Einheiten zu finden (Unit Tests bzw. Component Tests), nach oben wächst der Umfang eines einzelnen Tests. Auf die Tests einzelner Komponenten folgt der Test von zusammengesetzten Komponenten zu einem Dienst (Service Tests bzw. Integration Tests). Darüber sind Tests der Benutzeroberfläche zu finden (GUI Tests). Manuelle Tests sind separat über den GUI Tests dargestellt, da sie noch einfacher beeinflusst werden können. Je höher die Ebene, desto zerbrechlicher und wartungsintensiver sind die Tests. Tests von kleineren Programmeinheiten, wie sie beispielsweise von Komponententests umgesetzt werden, sind einfacher zu warten als umfangreiche Tests höherer Ebenen. Unter den Tests der oberen Ebenen sind die GUI Tests zu finden, die von jeder Änderung in einer unterliegenden Programmeinheit betroffen sein können. Um ein solides und möglichst einfaches Testkonzept zu erhalten, empfiehlt es sich, mehr Aufwand in Tests der unteren Ebenen zu investieren als in Tests der höheren Ebenen (vgl. [Van13, I.1.3]).

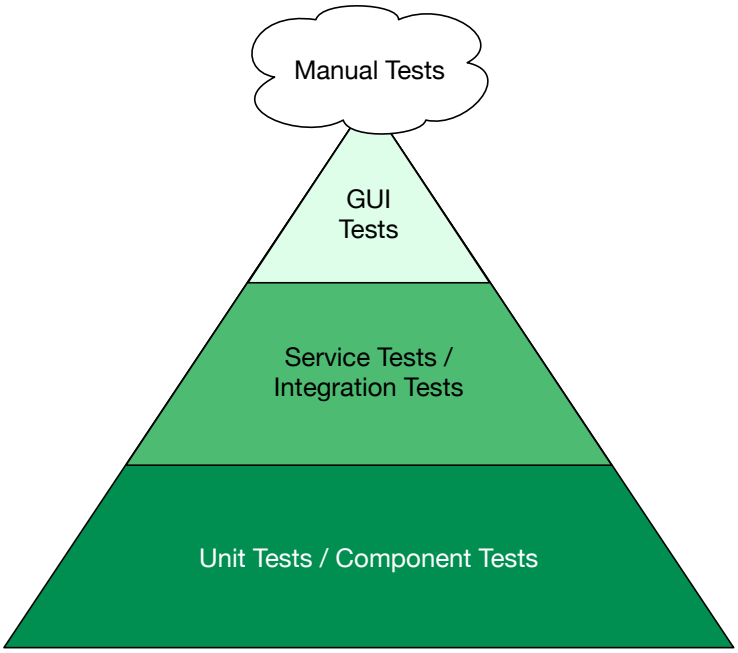


Abbildung 8: Pyramide mit den verschiedenen Ebenen des automatischen Testens. Abbildung nach [Coh10, Kap. III.16.2, Figure 16.1] und [Van13, Kap. I.1.3, Figure 1-1].

4.6.1 Komponententests

Die Überprüfungen einzelner Komponenten werden als Komponententests bezeichnet. Dabei werden möglichst kleine Programmteile separat getestet. Ein solcher Test bestimmter, isolierter Programmteile wird als Unittest bezeichnet. Das Konzept sieht vor, dass Komponententests vollständig automatisiert auf dem Buildsystem ausgeführt werden. Da die getesteten Programmteile isoliert betrachtet werden, fällt eine automatisierte Ausführung derartiger Testfälle meist leichter, als dies bei Integrationstests oder GUI Tests der Fall ist (vgl. Abbildung 8). Je nach Größe der Programmteile kann die Unterscheidung zu Integrationstests in der Praxis schwierig werden (vgl. [Hof13, S. 159]).

4.6.2 Integrationstests

Integrationstests werden an Programmteilen durchgeführt, die sich aus mehreren Komponenten zusammensetzen. Sie testen das Zusammenspiel einzelner Teilkomponenten, wobei die Teilkomponenten nicht voneinander isoliert sind.

Die Durchführung solcher Integrationstests kann im Rahmen des Konzepts über Testfälle erfolgen, die in Unittests realisiert werden. Dabei werden nun in dem Unittest, im Gegensatz zu reinen Komponententests, keine möglichst kleinen Programmteile getestet, sondern mögliche Zusammensetzungen dieser kleinen Programmteile. Integrationstests sind damit eine Ebene über den Komponententests zu sehen. Auch Integrationstests sollten automatisiert auf dem Buildsystem ausgeführt werden.

4.6.3 System- und Akzeptanztests

In Systemtests wird ein System vollständig, im Abgleich mit den an das System gestellten Anforderung, getestet. Zentraler Unterschied zwischen einem Integrationstest und einem Systemtest ist, dass bei einem Systemtest nur die Funktionalität eine Rolle spielt. Die konkrete Umsetzung der Funktionalität im Quellcode ist hierfür irrelevant.

Akzeptanztests, auch als Abnahmetests bezeichnet, haben die gleiche Zielsetzung wie Systemtests. Das entwickelte System wird mit den Anforderungen und Vorgaben abgeglichen. Trotzdem unterscheiden sich die beiden Testebenen in der Art ihrer Durchführung. Akzeptanztests liegen nicht in der Verantwortung des Softwareentwicklers, sondern der späteren Benutzer der Software (vgl. [Hof13, S. 168 f.]). Daraus folgt auch, dass ein Akzeptanztest in

einem Umfeld stattfindet, wie es bei den Benutzern vorzufinden ist. Dies bezieht sich zum einen auf die eingesetzten Systeme, zum anderen auf die für die Tests verwendeten Daten.

Nach dem Konzept des Entwicklungsprozesses ist es vorgesehen, dass auch System- und Akzeptanztest durchgeführt werden. Um sie zu automatisieren, bieten sich GUI Tests an. Wie in Abbildung 8 dargestellt, liegen GUI Tests auf einer höheren Ebene als beispielsweise Integrationstests. Da bei GUI Tests mehr Quellcode in die Ausführung involviert ist als es bei isolierten Komponenten- oder Integrationstests der Fall ist, sind sie deutlich wartungsintensiver. Der Aufwand für die Wartung von GUI Tests sollte deshalb nicht unterschätzt werden. Bei Oberflächen, deren Design häufig verändert wird, oder bei solchen, die auf häufig zu überarbeitenden Quellcode basieren, sollten GUI Tests eher selten oder gar nicht verwendet werden. Gerade im Anbetracht der Tatsache, dass in der Abteilung die Softwareentwickler für die Erstellung und Wartung von Tests zuständig sind, ist auf zusätzliche, häufig auftretende Wartungsaufgaben zu verzichten. In diesen Fällen sollten mehr Komponenten- und Integrationstests, ergänzt durch manuelle Tests, verwendet werden. Bei Oberflächen, die kaum Änderungen erfahren, können natürlich trotz dessen GUI Tests zum Einsatz kommen.

Weil die Automatisierung von System- und Akzeptanztest zuweilen schwer fallen oder sich als sehr zeit- und ressourcenintensiv herausstellen kann, ist es, wie zuvor erwähnt, auch möglich, diese Tests manuell durchzuführen. Bei manuellen System- und Akzeptanztests ist es besonders wichtig, dass eine Dokumentation des Testvorgehens erfolgt. Neben einer Angabe der Testdaten ist von Bedeutung, dass sämtliche Schritte reproduzierbar und mit erwarteten Ergebnissen notiert werden. Zusätzlich ist es sinnvoll, mögliche Fehlerquellen und Hinweise zum Vorgehen in die Dokumentation einfließen zu lassen.

4.6.4 Auswahl der Testfälle

Die Art der Daten, die für die Tests verwendet werden, ist spezifisch für das Entwicklungsprojekt und die Testfälle zu definieren. Die Benutzung von realitätsnahen Testdaten, wie sie in der Abteilung durch Simulationen oder vergangene Messungen vorliegen, bietet sich in vielen Fällen an. Für die Erstellung von Testfällen haben sich verschiedene Techniken etabliert (nach [Gra15], [Hof13, S. 173 ff.], [Int11, S. 37 ff.]):

Die Black-Box Technik Beim Black-Box Testentwurf werden die Testfälle ohne Kenntnis des Quellcodes oder der internen Struktur entworfen. Stattdessen wird die äußere Spezifikation der zu testenden Codeteile zu Hilfe genommen. Mögliche Verfahren sind

beispielsweise der Äquivalenzklassentest, die Grenzwertanalyse, der Entscheidungstabellentest oder der zustandsbasierte Test.

Die White-Box Technik Beim White-Box Testentwurf ist die innere Struktur des zu testenden Quellcodes bekannt. Aus dieser inneren Struktur werden Testfälle gebildet. Mögliche Verfahren sind beispielsweise die Anweisungs-, Entscheidungs- oder Pfadüberdeckung.

Die Gray-Box Technik Der Gray-Box Testentwurf kombiniert die Black- mit der White-Box Technik. Sowohl die innere Codestruktur als auch die äußere Spezifikation sind beim Testentwurf bekannt.

Um eine hohe Testabdeckung zu erreichen, bietet sich die Nutzung von White-Box Testfällen an. Für die Ermittlung der Testabdeckung können passende Softwarewerkzeuge verwendet werden. Diese sogenannten *Coverage Tools* messen, welcher Anteil des Quellcodes von den Testfällen ausgeführt wird. Die errechneten Werte werden als Überdeckungsmetriken bezeichnet (vgl. [Hof13, S. 231 f.]). Coverage Tools können auch in einen automatisierten Prozess integriert werden. In diesem Fall kann ein verpflichtender Wert einer Überdeckungsmetrik festgelegt werden, der von den Testfällen erreicht werden muss. Falls er nicht erreicht wird, wird der Testprozess als fehlgeschlagen angesehen. Vorteil einer solchen, verpflichtenden Testabdeckung ist, dass eine hohe Anzahl an erfolgreich ausgeführten Tests garantiert wird. Nachteilig ist zu sehen, dass der gesamte, auf die Testausführung folgende Prozess, bei zu geringer Testabdeckung nicht ausgeführt wird. Falls zusätzlich die Zeit zur Entwicklung einer ausreichenden Menge an Testfällen fehlt, kann dies dazu führen, dass die Akzeptanz des Entwicklungsprozesses im Entwicklerteam sinkt. Aus diesem Grund wird in dem konzipierten Prozess auf eine verpflichtenden Testabdeckung verzichtet.

Die weitere Auswahl der Testentwurfsverfahren muss passend zu der zu testenden Software erfolgen, deshalb wird an dieser Stelle auf eine detailliertere Betrachtung der Verfahren verzichtet. Grundsätzlich gilt das Konzept zum Testprozess für alle Tests, die über ein beliebiges Entwurfsverfahren erstellt werden. Es ist zu beachten, dass das verwendete Entwurfsverfahren und die damit erzeugten Testfälle und -daten übereinstimmend mit Abschnitt 4.6.5 dokumentiert und archiviert werden.

4.6.5 Management der Testdaten

Die notwendigen Testdaten sind in einer Form zu hinterlegen, die auf Entwicklungssystemen und dem Buildsystem nutzbar sind. Die integrierte Entwicklungsumgebung der Entwickler

muss genauso auf die Daten zugreifen können wie das Test Framework auf dem Buildsystem. Dafür sind Art und Ort der Daten zu betrachten.

Die Testdaten sollten, wie der Quellcode auch, in einer Versionsverwaltung abgelegt werden. Idealerweise wird das gleiche Repository eingesetzt wie für den Quellcode. Falls die Testdaten in mehreren Entwicklungsprojekten eingesetzt werden sollen, können sie in vom Quellcode der Tests getrennten Repositories abgelegt werden. Unter diesen Umständen ist es allerdings deutlich schwieriger sicherzustellen, dass auf jedem System zu den implementierten Testfällen passende Testdaten verfügbar sind.

In jedem Fall ist es vorgesehen, dass die verfügbaren und verwendeten Testdaten zentral dokumentiert werden. Die dafür einzusetzende Testdatenbank soll folgende Aspekte abdecken:

- Übersicht über verfügbare Testdaten: Entwickler, die für neu erstellte Testfälle Daten benötigen, bekommen eine Übersicht über verfügbare Testdaten bereitgestellt. Enthalten sein sollten:
  - Art der Daten: Entstammen die Daten einer Messung oder wurden sie simuliert? Nach welchem Schema wurden sie erstellt?
  - Ursprung der Daten: Wo wurden die Daten aufgenommen? Wer hat sie simuliert bzw. bei welcher Messung wurden sie aufgezeichnet?
  - Vorhandene Daten: Welche Verarbeitungsformen und Formate der Daten liegen vor?
  - Zeitpunkt der Datenerstellung.
- Übersicht über die Verwendung der Testdaten: Falls ein Fehler in verwendeten Testdaten auffällt, sollte bekannt sein, wo die fehlerhaften Testdaten eingesetzt werden.
- Kein überflüssiger Aufwand für die Verwaltung der Daten: Für die Verwaltung der Testdaten sollte ein einfaches Verfahren eingesetzt werden, welches möglichst wenig zusätzliche Software, Einarbeitungszeit und Kosten verursacht.

4.7 Softwareverteilung

Nachdem die entwickelte Software erstellt, getestet und eine Installationsdatei erzeugt wurde, wird die Verteilung der Software angestoßen. Die somit erstellte Software ist, wie in Abschnitt 3.2.2 beschrieben, ausschließlich Client-Software für Microsoft Windows-Betriebssysteme. Ziel ist es, den Nutzern der Anwendung die nun erstellte Version zur Verfügung zu stellen.

Da es sich bei den Softwareentwicklungsprojekten in der Abteilung AE-SAS um Software handelt, die nahezu ausschließlich intern verwendet wird, steht die interne Softwareverteilung im Vordergrund. Die internen Nutzer einer Software sollten, bei Verfügbarkeit einer neuen Version, über diese Aktualisierung informiert werden. Wichtig ist weiterhin, dass für sie die Beschaffung der neuen Version möglichst einfach abläuft.

Eine automatische Aktualisierung der Software auf den Systemen der Nutzer kann für einige, unkritische Anwendungen implementiert werden. Da unerwünschte Nebenwirkung einer stillen Aktualisierung nicht auszuschließen sind, ist eine Verwendung dieser Funktionalität in Anwendungen mit hohem Schutzbedarf (vgl. Abschnitt 4.1) nicht vorzusehen.

Ungeachtet dessen, ob eine stille Aktualisierung der Software durchgeführt wird, sollten die Anwender zu jeder Zeit eine Möglichkeit dazu haben, sich über die Aktualität ihrer Software zu informieren. Ihnen sollte ersichtlich sein, welche Versionen sie installiert haben und welche weitere Versionen zur Verfügung stehen. Dabei sollte klar zu erkennen sein, ob eine Aktualisierung der Software erforderlich ist. Über die reine Anzeige an Versionsinformationen hinaus sollte auch die Möglichkeit gegeben sein, gewünschte Versionen einer Software zu installieren. Für diese Aufgaben der Anzeige und Aktualisierung sollte ein System zum Softwarekonfigurationsmanagement eingesetzt werden, welches auch die Softwareverteilung regelt.

Neben der Anwendung allein ist auch die Auslieferung der Benutzerdokumentation eine Aufgabe der Softwareverteilung. Dabei nehmen die Änderungsprotokolle eine besondere Rolle ein, da die Veränderungen an der Software den Anwendern bereits vor einer Installation ersichtlich sein müssen. Ein einfacher Zugriff auf diese Dokumente ist deshalb unerlässlich.

4.8 Feedback an die Entwickler

Für den Fall, dass die Anwender auf Fehler in der entwickelten Software stoßen, sollten die Entwickler der Software über diese Fehler in Kenntnis gesetzt werden. Nur so kann im Rahmen des Fehlermanagements sichergestellt werden, dass die Fehler in Zukunft tatsächlich behoben werden (vgl. [BK13, S. 280 f.]). Wichtig ist, dass den Entwicklern eine strukturierte Übersicht über die Meldungen der Nutzer zur Verfügung steht. Durch Fehlerberichte über unterschiedliche Kommunikationskanäle nimmt der Verwaltungsaufwand der Berichte zu, die Entwickler können leicht den Überblick über ihre Aufgaben verlieren. Abhilfe kann hier ein zentrales System zur Verwaltung der Meldungen von Nutzern bieten (vgl. [BK13, S. 126 ff.]).

Diese Möglichkeit zur Rückmeldung an die Entwickler sollte neben der Meldung von Fehlern auch für die Anfrage nach weiterer Funktionalität in der entwickelten Software umgesetzt werden.

Damit das Feedback für die Entwickler nachvollziehbar ist, sollte es bestimmten Kriterien genügen. Wichtig ist, dass eine Beschreibung des Feedbacks, ein meldender Benutzer und eine Versionsnummer, zu der das Feedback erstellt wird, vorhanden sind. Bei Nachfragen kann der Entwickler sich, dank der Namensangabe, direkt an den Benutzer wenden, der die Meldung erstellt hat. Durch eine ausführliche Beschreibung, die bei Fehlermeldungen beispielsweise eine detaillierte Erklärung der notwendige Schritte zur Reproduktion beinhalten kann, sollte eine Kontaktierung des meldenden Benutzer jedoch nicht grundsätzlich notwendig sein. Um das Feedback mit weiteren hilfreichen Informationen anzureichern, ist es vorzusehen, dass Nutzer Dateien, insbesondere Bildschirmfotos, anfügen können.

5 Umsetzung des Entwicklungsprozesses

Das in Kapitel 4 beschriebene Konzept des neuen Softwareentwicklungsprozesses wurde im Rahmen der Bachelorarbeit auf einem Testsystem implementiert. Zur Umsetzung wurde dabei verschiedene freie und kommerzielle Software evaluiert und eingesetzt, die nach Bedarf untereinander gekoppelt wurde. In Abschnitt 5.1 wird die untersuchte Software vorgestellt. Abschnitt 5.2 beschreibt, wie die Software eingerichtet wurde, um das vorgestellte Konzept umzusetzen.

5.1 Softwarewerkzeuge

Um den konzipierten Entwicklungsprozess mit kontinuierlicher Integration umzusetzen, wurde eine Reihe von Softwarewerkzeugen evaluiert. Die gesamte Werkzeugpalette wurde zur Evaluierung auf Microsoft Windows 7-Testsystemen installiert und konfiguriert. Sofern es sich um Software handelt, die später auch die Entwickler verwenden sollen, wurde die Software auf einem System evaluiert, dass den Entwicklersystemen entspricht. Die serverseitige Software zur kontinuierlichen Integration wurde auf einem separaten Windows 7-System aufgesetzt, welches für diesen Zweck vorgesehen ist.

5.1.1 Aufgabenverwaltung und Code Review mit Phabricator

Als Werkzeug für die Verwaltung der anfallenden Aufgaben wird in diesem Abschnitt die Webanwendung Phabricator beschrieben. Neben der Nutzung als Aufgabenverwaltung lässt sich das Kollaborationswerkzeug Phabricator auch zum Code Review verwenden.

Phabricator [Pha15c] ist ein Open Source Kollaborationswerkzeug zur Entwicklung von Software. Es entstammt internen Werkzeugen des Unternehmens Facebook und wird von Phacility [Pha15d] unter der Apache Lizenz Version 2 [The04] entwickelt. Phabricator ist als Webanwendung in der Sprache PHP geschrieben und kann so von Browsern auf beliebigen Plattformen aufgerufen werden. Die Authentifizierung bei Phabricator kann über eine externe Lightweight Directory Access Protocol (LDAP)-Benutzerverwaltung erfolgen.

Phabricator besteht aus mehreren Anwendungsteilen, die jeweils für eine Funktionalität zuständig sind. Dabei können Objekte verschiedener Anwendungsteile untereinander verknüpft werden, um logische Abhängigkeiten abzubilden. Einen zentralen Punkt für die Verwaltung

von Objekten stellen die Projekte dar. Viele Objekttypen können einem oder mehreren Projekten zugeordnet werden, so dass ein Phabricator-Projekt wie eine Übersicht über ein reales Entwicklungsprojekt angesehen werden kann.

Objekte, seien es Aufgaben oder Commits, können in Phabricator von Benutzern kommentiert werden. Die Kommentarvarianten können, je nach Objekttyp, verschieden sein. Von allen Objekttypen werden Textkommentare unterstützt. Textkommentare in Phabricator verwenden eine eigene Auszeichnungssprache, mit denen auch Bilder und Anhänge eingebunden werden können.

Die im Kontext dieser Arbeit wichtigen Anwendungsteile von Phabricator sind:

Maniphest stellt einen zentralen Teil von Phabricator dar. Es dient der Organisation von Aufgaben. Aufgaben, in Phabricator als *Tasks* bezeichnet, lassen sich mit Projekten verknüpfen und können beliebigen Benutzern zugewiesen werden. Der Status einer Aufgabe kann in Maniphest ebenso festgelegt werden wie deren Priorität. Um eine Aufgabe genau zu definieren, kann ein Titel und eine Beschreibung angegeben werden, es können Abhängigkeiten zwischen Aufgaben definiert und Commits referenziert werden.

Die Maniphest-Aufgaben eines Projekts können von Phabricator auf einem Kanban-Board mit frei definierbaren Spalten angezeigt werden (vgl. Abschnitt 2.2.2). Damit ist, neben dem Status und der Priorität, eine weitere Möglichkeit vorhanden, wie Nutzer Aufgaben organisieren können.

Diffusion ist die in Phabricator enthaltene Anwendung zur Verwaltung und Betrachtung von Repositories. Phabricator, und damit auch Diffusion, unterstützen Repositories von mehreren Versionsverwaltungssystemen: Git, Mercurial und Subversion. Die Repositories können entweder direkt von Phabricator gehostet werden oder von einer externen Quelle importiert werden. Falls der Import gewählt wird, wird das Repository an seinem ursprünglichen Ort belassen und Änderungen am Repository werden von Phabricator gelesen. Für von Phabricator gehostete Repositories steht eine HTTP-URL als Repository-Pfad zur Verfügung.

Differential dient in Phabricator dem Review von Code, bevor er auf ein Repository geladen wird (*Pre-Push*). Über das Kommandozeilenwerkzeug Arcanist oder die Weboberfläche von Phabricator muss ein sogenanntes *Diff* eines Commits, in dem alle im Commit enthaltenen Änderungen abgebildet sind, angelegt werden. Aus dem Diff wird ein Differential-Review erzeugt, dass von einem Mitglied des Entwicklerteams bearbeitet

werden kann. Ein Reviewer kann unter anderem Quellcode annehmen, ablehnen oder Änderungen am Code verlangen.

Audit ist Phabricators Anwendung zum Review von Code, der bereits auf ein Repository geladen wurde (*Post-Push*). Post-Push Code Reviews werden im Phabricator-Umfeld als *Audits* bezeichnet. Im Gegensatz zu Differential-Reviews, die vor einem Push ausgeführt werden, sind die Audits in Phabricator nicht blockierend. Sie können mit geringerem Aufwand eingeführt werden, da der Übergang eines Arbeitsablaufes gänzlich ohne Code Reviews zu einem Arbeitsablauf mit Phabricator-Audits wenig Veränderungen erfordert (vgl. [Pha15b]).

Herald dient der Ausführung von Aktionen auf Basis von festgelegten Regeln. Dabei lassen sich für viele Objekte in Phabricator Regeln definieren, nach deren Auslösung zuvor definierte Aktionen ausgeführt werden. Beispiele für Herald-Regeln können sein:

- Wenn eine Aufgabe auf ein bestimmtes Projekt bezogen erstellt wird, werden alle Mitglieder des Projektes benachrichtigt.
- Wenn ein Commit auf ein ausgewähltes Repository gepusht werden soll, wird dies nur erlaubt, falls der Commit ein Differential-Review durchlaufen hat.
- Wenn ein Commit auf ein ausgewähltes Repository gepusht wird, wird anschließend durch Harbormaster ein Build mit einem bestimmten Build Plan ausgeführt.

Harbormaster ist das Buildmanagement-Werkzeug von Phabricator. In Harbormaster, welches noch nicht in einer stabilen Version veröffentlicht wurde und daher bisher nur als Prototyp getestet werden konnte, können Buildaufträge in sogenannten *Build Plans* definiert werden. Ein Build Plan besteht aus einer Abfolge an Schritten, die für einen Build erforderlich sind. Aufgrund der frühen Entwicklungsphase sind die meistens Buildschritte noch unzureichend implementiert. Als nützlich stellte sich der Schritt zur Ausführung einer HTTP-Anfrage heraus, von der aus beispielsweise ein Build auf einem Jenkins-Server ausgelöst werden kann.

Conduit ist die Bezeichnung für das Application Programming Interface (API) von Phabricator. Für den Zugriff auf den Großteil der Methoden muss sich der Nutzer der API authentifizieren. Die Authentifizierung kann unter anderem über ein Authentifizierungstoken vorgenommen werden, welches in jeder Anfrage an die Phabricator-API enthalten ist. Die Kommunikation mit der Conduit-API erfolgt über HTTP unter Verwendung von JavaScript Object Notation (JSON)-Objekten.

Arcanist ist ein Kommandozeilenwerkzeug, welches ergänzend zu der Phabricator-Webanwendung bereitgestellt wird. Es bietet eine Reihe von Kommandos an, über die Aktionen durchgeführt werden können, die ansonsten nur in der Weboberfläche im Browser verfügbar sind (vgl. [Pha15a]). Ein verbesserter Arbeitsablauf für Entwickler kann erreicht werden, indem Arcanist der Weboberfläche für einige Anwendungszwecke vorgezogen wird. Erwähnt sei hier die Bereitstellung von Differential-Diffs, die per Arcanist mit nur einem Befehl ausgeführt werden kann und ansonsten manuell über das Kopieren der Ergebnisse einiger Kommandozeilenbefehle in die Weboberfläche durchgeführt werden müsste.

5.1.2 Versionsverwaltung

Das Konzept zum Entwicklungsprozess sieht es vor, dass zur Verwaltung aller Dateien eines Entwicklungsprojekts, inklusive dem Quellcode, ein Versionsverwaltungssystem eingesetzt wird. In diesem Abschnitt erfolgt die Vorstellung der beiden verteilten Versionsverwaltungssysteme Mercurial und Git. Zentrale Versionsverwaltungssysteme wurden nicht betrachtet, da diese nicht die notwendige Flexibilität bieten, um bei fehlender Verfügbarkeit des zentralen Servers die Funktionalität aufrecht zu erhalten (vgl. Abschnitt 4.3.1).

5.1.2.1 Mercurial

Mercurial [Mac15] ist ein verteiltes System zur Versionskontrolle. Es ist mit seinen Implementierungen in Python und C plattformunabhängig entwickelt worden. Mercurials Entwicklung begann 2005 und wird seitdem als Open Source-Projekt unter der Lizenz GNU General Public License (GPL) Version 2 [Fre91] geführt. Wie das Versionskontrollsystem Git ist es von der Software Monotone beeinflusst worden und verwendet eine ähnliche Peer-to-Peer-Architektur (vgl. [O'S09, Kap. 1]). Durch diese verteilte Architektur muss nicht ein einziger, zentraler Server alle Änderungen verwalten, stattdessen können die Änderungen in Repositories auf verschiedenen Systemen gelagert und nach Wunsch untereinander synchronisiert werden. Ein Entwickler kann so beispielsweise eine Änderung im Code, ohne Verbindung zum zentralen Server, lokal als Commit speichern. Sobald eine Verbindung zum Server vorhanden ist, kann der Commit auf den Server übertragen werden.

5.1.2.2 Git

Git [Sco15] ist, wie Mercurial, ein verteiltes Versionskontrollsystem. Auch Git ist unter der Open Source-Lizenz GPL Version 2 [Fre91] verfügbar. Die Entwicklung von Git begann nicht ohne Grund nahezu zeitgleich mit der von Mercurial im Jahr 2005: Beide Projekte wurden zu der Zeit ins Leben gerufen, als für die Entwicklung des Linux-Kernels ein neues Versionskontrollsystem benötigt wurde (vgl. [CS14, S. 31]).

Auch wenn Git und Mercurial in einem Großteil der Funktionalitäten Überschneidungen aufweisen, gibt es einige Unterschiede zwischen den Systemen. Git weist unter Unix-basierten Betriebssystemen eine höhere Leistungsfähigkeit auf als Mercurial, dafür ist dies unter Windows-Betriebssystemen umgekehrt der Fall (vgl. [O'S09, Kap. 1]). Auch unterscheidet sich die Syntax der Befehle der beiden Systeme, wenngleich der Funktionsumfang nahezu identisch ist. Ein weitere Unterschied zeigt sich in der Beliebtheit: Git ist weitaus verbreiteter als Mercurial (vgl. [Bla15], [Sof15]). Daraus folgt in der Praxis auch, dass Anwendungen häufiger Unterstützung für Git bieten als für Mercurial.

5.1.3 Entwicklungsumgebung

Als integrierte Entwicklungsumgebung, die für die Umsetzung des Entwicklungsprozesses eingesetzt wird, kommt besonders das bereits für bisherige Entwicklungen verwendete Microsoft Visual Studio infrage.

5.1.3.1 Microsoft Visual Studio

Die integrierte Entwicklungsumgebung Visual Studio [Mic15j] wird seit 2007 von Microsoft entwickelt. Am 20. Juli 2015 ist die aktuelle Version 2015 alias 14.0 erschienen.

In Visual Studio gibt es zwei Container zur Organisation von Entwicklungsprojekten: Die Projekte und die Projektmappen. Eine Projektmappe enthält dabei beispielsweise alle Visual Studio-Projekte, die für ein zu entwickelndes Softwareprodukt benötigt werden. Ein Visual Studio-Projekt enthält den zur Entwicklung benötigten Quellcode und erzeugt eine Ausgabe. Dies kann etwa eine ausführbare Programmdatei oder eine Programmbibliothek sein.

Visual Studio ist in mehreren Varianten mit unterschiedlichem Funktionsumfang erhältlich, darunter auch in der kostenfreien Variante Visual Studio Community (vgl. [Mic15g]). Erweiterte Funktionalität, wie die Möglichkeit zur Durchführung von Tests der Benutzeroberfläche,

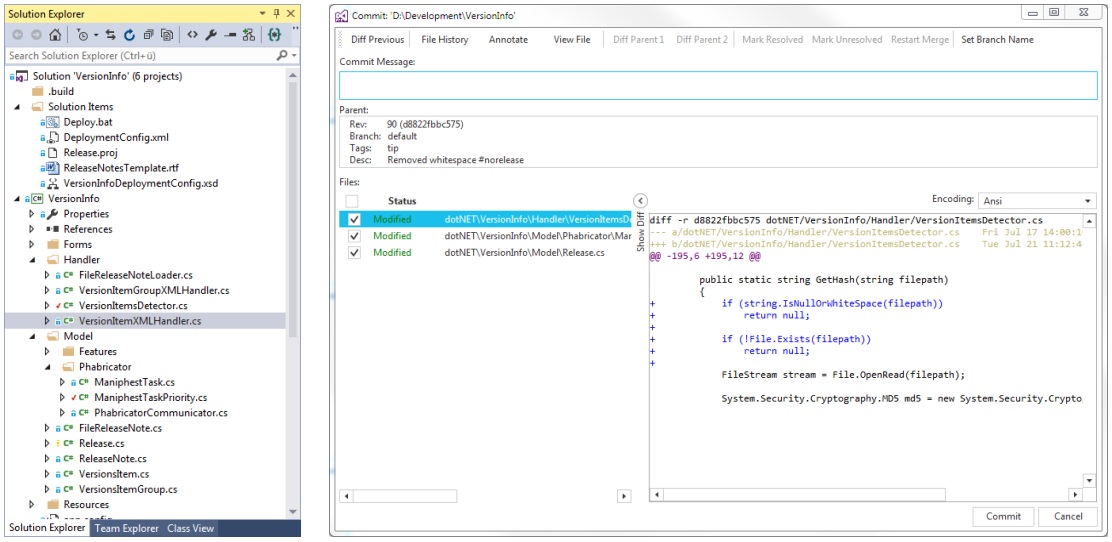
bieten die kostenpflichtigen Varianten. Die Variante mit dem größten Funktionsumfang ist Visual Studio Enterprise, welche mit sogenannten *Coded UI Tests* die Erstellung und Durchführung von solchen Benutzeroberflächentests ermöglicht. Außerdem ermöglicht Visual Studio Enterprise die Ermittlung der Testabdeckung. Unten stehende Funktionen sind in allen Varianten von Visual Studio 2015 enthalten:

- Unterstützung für C# und das .NET Framework
- Syntaxvervollständigung und -prüfung für eine große Zahl an Programmier- und Markupsprachen
- Designer für grafische Oberflächen im XAML-Format oder, passend für Windows Forms, in C#
- Debugging vieler Programmiersprachen, darunter C#
- Erstellung, Ausführung und Verwaltung von Unittests
- Bereitstellung von Refactoring-Werkzeugen
- Erstellung von Metriken
- Erweiterbarkeit

Visual Studio bietet über den integrierten Paketmanager NuGet [.NE15] außerdem die Möglichkeit, existierende Programmbibliotheken in eigene Entwicklungsprojekte einzubinden. Hierzu stellt die NuGet Gallery ein zentrales Paket-Repository bereit.

Die Erweiterung HgSccPackage [Ser15] für Visual Studio ermöglicht es, direkt von Visual Studio auf Mercurial-Repositories zugreifen zu können. HgSccPackage integriert sich dafür, wie in Abbildung 9 dargestellt, in die grafische Oberfläche von Visual Studio. Über neue Steuerelemente können die wichtigsten Mercurial-Kommandos aufgerufen werden. Auch die Projektübersicht wird um Symbole zum Synchronisierungsstatus erweitert (vgl. Abbildung 9a). Im Solution Explorer von Visual Studio stellen die farbigen Symbole neben Dateinamen und Dateiicons den Zustand der Datei im Bezug auf die aktuellste Version im Repository dar. Möglich ist unter anderem die Erstellung von Commits (vgl. Abbildung 9b) und die Synchronisierung mit dem entfernten Repository mittels Pull und Push. In der Commit-Oberfläche werden links die geänderten Dateien angezeigt, rechts der geänderte Inhalt der ausgewählten Datei.

Während bei älteren Versionen von Visual Studio noch eine Erweiterung eines Drittanbieters notwendig war, um den Zugriff von Visual Studio auf Git Repositories zu ermöglichen, wird



(a) Solution Explorer von Visual Studio.

(b) Die Oberfläche von HgSccPackage zur Erstellung eines neuen Commits.

Abbildung 9: Screenshots von Visual Studio 2015 mit der HgSccPackage-Erweiterung.

dafür seit 2013 eine Lösung von Microsoft angeboten (vgl. [Har15]). Für Visual Studio 2012 wurde von Microsoft eine eigene Erweiterung bereitgestellt, seit Version 2013 wird die Git Unterstützung direkt mit Visual Studio ausgeliefert.

5.1.3.2 ReSharper

ReSharper [Jet15] ist eine, von JetBrains entwickelte, kommerzielle Erweiterung für Microsoft Visual Studio. Sie ergänzt Visual Studio um eine Reihe an Funktionen, die es von Haus aus nicht bietet. Einige dieser Funktionen sind (vgl. [Jet15], [Gas14, Kap. 2 ff.]):

- Statische Quellcodeanalyse: ReSharper analysiert den Quellcode und zeigt den Entwicklern kontinuierlich Verbesserungsmöglichkeiten und potentielle Fehlerquellen an. Es wird zwischen Warnungen und Vorschlägen unterschieden, sodass der Entwickler weiß, welche Wichtigkeit eine Anmerkung hat. Für die meisten Anmerkungen werden direkt Lösungsvorschläge und -werkzeuge angeboten.
- Überprüfung des Programmierstils: Nach festgelegten Regeln überprüft ReSharper den Quellcode und zeigt Hinweise an, falls der definierte Programmierstil verletzt wird. Zur

Behebung der Verletzungen bietet ReSharper Werkzeuge an, die auf Wunsch auch auf mehrere Dateien oder ein ganzes Projekt angewendet werden können.

- Erweitertes Refactoring: Zusätzlich zu den in Visual Studio enthaltenen Refactoringwerkzeugen bietet ReSharper eine Vielzahl an weiteren Refactoringmaßnahmen an, die Entwicklern eine Zeitersparnis gegenüber dem manuellen Refactoring bringen können.
- Codegenerierung: Neben dem Refactoring von bestehendem Quellcode kann auch Code mit ReSharper erzeugt werden. Häufig notwendige Programmierarbeiten können so automatisiert werden.
- Projektweite Einstellungen: Die Einstellungen für ReSharper können lokal oder projektweit gespeichert werden. Bei projektweiten Einstellungen kann die Konfigurationsdatei innerhalb der Versionsverwaltung abgelegt und unter den Entwicklern ausgetauscht werden.

5.1.4 Build

Für die Umsetzung des Buildsystems wird Software benötigt, die die Erstellung der Anwendung, der Dokumentation und der Installationsdateien übernimmt. In diesem Abschnitt wird die für besagten Anwendungszweck eingesetzte Software beschrieben.

5.1.4.1 Jenkins

Jenkins [Jen15b] ist ein in Java geschriebenes Werkzeug zur kontinuierlichen Integration. Die Bedienung von Jenkins erfolgt über eine Weboberfläche. Jenkins wird unter der Open Source-kompatiblen MIT-Lizenz [Mas88] veröffentlicht. Die Entwicklung wurde innerhalb des Hudson Projektes im Jahr 2004 begonnen und wird seit 2011, nach Auseinandersetzungen zwischen Oracle und der Open Source-Community, unter dem Namen Jenkins weitergeführt (vgl. [Sma11, Kap. 1]).

Mit Hilfe von Plugins lässt sich Jenkins erweitern. Über das Plugin-Verzeichnis auf der Webseite von Jenkins, welches auch in die Jenkins-Software direkt integriert ist, kann eine Vielzahl an Drittanbieterplugins installiert werden (vgl. [Sma11, Kap. 4.2]). Die Plugins fallen, wie Jenkins, unter die MIT-Lizenz. Zu den für das Entwicklungskonzept relevanten Plugins gehören:

Das Mercurial Plugin [JK15a] für Jenkins fügt zu diesem die Unterstützung für das Versionsverwaltungssystem Mercurial (vgl. Abschnitt 5.1.2.1) hinzu. Dadurch ist Jenkins in der Lage, zu Beginn eines Builds den Quellcode des Builds aus einem Mercurial-Repository auszuchecken.

Das MSBuild Plugin [JK15b] ermöglicht es Jenkins, .NET-Quellcode mit MSBuild (vgl. Abschnitt 5.1.4.2) zu kompilieren. MSBuild wird dabei über die Kommandozeile aufgerufen, die erzeugten Ausgaben werden in den von Jenkins gespeicherten *Console Logs* gespeichert. So können Fehler, die während des Buildvorgangs auftreten, verfolgt werden.

Das MSTest Plugin [Jen15a] ist eine Erweiterung für Jenkins, die es ermöglicht, Tests über MSTest (vgl. Abschnitt 5.1.5.1) auszuführen. Die Testsausführung findet im Anschluss an einen Build, wie er mit MSBuild durchgeführt werden könnte, statt. Die Testergebnisse werden ebenfalls im *Console Log* von Jenkins gespeichert. Zusätzlich erfolgt die Erstellung von detaillierten Testergebnissen durch MSTest.

Das VsTestRunnerPlugin Plugin [JY15] ist eine Erweiterung für Jenkins, mit der Tests über VSTest (vgl. Abschnitt 5.1.5.2) ausgeführt werden können. Die Funktionsweise entspricht größtenteils der des MSTest Plugins, mit der Ausnahme, dass VSTest statt MSTest zur Durchführung der Tests verwendet wird.

5.1.4.2 Microsoft Build Engine

Die Microsoft Build Engine (MSBuild) [Mic15i] ist eine Buildplattform von Microsoft, die unter der MIT-Lizenz veröffentlicht wurde. Verwendet wird MSBuild unter anderem zum Kompilieren von Projekten in Visual Studio. MSBuild verwendet Projektdateien im XML-Format zur Spezifizierung des Buildprozesses (vgl. [Mic15f]).

MSBuild kann mit eigenen Aufgaben, sogenannten MSBuild Tasks, erweitert werden. Das Projekt MSBuild Community Tasks [Lor15] bietet eine Vielzahl an fertiger Tasks an, die in MSBuild-Projekten verwendet werden können. Hervorzuheben sind vielseitig verwendbare Aufgaben wie die Interaktion mit Versionsverwaltungssystemen, die Erstellung oder Veränderung von Dateien und die Ersetzung von Text in Dateien mittels regulärer Ausdrücke.

5.1.4.3 Windows Installer XML Toolset

Das Windows Installer XML (WiX) Toolset [OR15] ist eine Werkzeugsammlung, mit der Windows Installer-Pakete [Mic15b] erzeugt werden können. Es ist unter der Microsoft Reciprocal License (Ms-RL) [Mic07b] lizenziert. Die Entwicklung des WiX Toolsets als Open Source Software hat bei Microsoft begonnen. Die Definition eines zu erstellenden Windows Installer-Pakets findet im XML-Format statt. Die erzeugten Pakete sind MSI-Dateien, die relationale Datenbanken darstellen (vgl. [Ram12, Kap. 1.4]). Es besteht die Möglichkeit, die MSI-Dateien direkt auszuführen und die Installation über einen grafischen Dialog durchzuführen, oder die Software über Kommandozeilenbefehle und ohne Eingreifen des Benutzers zu installieren.

Über eine Visual Studio-Erweiterung integriert sich WiX direkt in die IDE. In Kombination mit MSBuild integriert sich WiX in den Buildprozess. Die zum Toolset gehörenden Werkzeuge werden dabei in der vordefinierten Reihenfolge aufgerufen.

5.1.4.4 Sandcastle Help File Builder

Das Sandcastle Help File Builder (SHFB) Projekt [Eri15] stellt Werkzeuge zur Erstellung von Dokumentationen zu Software bereit. Es basiert auf der ab 2006 bei Microsoft entwickelten Software Sandcastle und erstellt Dokumentationen im Stil des Microsoft Developer Network (MSDN). Microsoft entwickelt Sandcastle seit 2010 nicht mehr weiter, stattdessen wird es seitdem von Eric Woodruff in einer Abspaltung des ehemaligen Projekts weitergeführt. SHFB steht, wie auch das ursprüngliche Sandcastle, unter der Open Source-Lizenz Microsoft Public License (Ms-PL) [Mic07a]. Sandcastle und SHFB erstellen Dokumentationsdateien im HTML-Format, die typischerweise in eine einzige, komprimierte Datei im CHM-Format umgewandelt werden.

Von SHFB werden zwei Arten von Themen in den Dokumentationen unterstützt: API Referenzen und konzeptionelle Themen. Die API Referenzen werden unter Verwendung von XML-Kommentaren im Quellcode der zu dokumentierenden Software sowie der Syntax und Struktur der durch Code repräsentierten Typen erstellt (vgl. [Mic15a]). Die konzeptionellen Themen können in der XML-basierten Markupsprache Microsoft Assistance Markup Language (MAML) verfasst werden und können für die Erstellung einer Benutzerdokumentation mit Nutzungshinweisen, Anleitungen und Ähnlichem dienen.

Die Definition eines Dokumentationsprojektes mit SHFB erfolgt in einer MSBuild-Projektdatei, die von MSBuild wie jede andere Projektdatei als Build-Anweisung interpretiert und ausgeführt

werden kann. Somit gestaltet sich die Integration in eine kontinuierliche Build-Umgebung mit MSBuild einfach. SHFB bringt für Visual Studio ab Version 2013 eine Erweiterung mit sich, die zur Verwaltung von SHFB-Projekten direkt in Visual Studio dient.

Eine Alternative zu SHFB stellt Doxygen dar. Doxygen [van15] ist ein Dokumentationswerkzeug, welches unter der Open Source Lizenz GPL Version 2 [Fre91] verfügbar ist. Es wird besonders bei Projekten in der Programmiersprache C++ häufig eingesetzt, beschränkt sich jedoch nicht nur auf diese Sprache (vgl. [Lis13, Kap. 27]). Stattdessen unterstützt es auch Sprachen wie Java, Python und C#. Als Ausgabeformate für die Dokumentation lassen sich unter anderem HTML, LaTeX, Extensible Markup Language (XML) oder CHM-Dateien verwenden.

Zur Benutzung von Doxygen existieren von Haus aus zwei Methoden: Es kann über die Kommandozeile aufgerufen werden oder über eine grafische Benutzeroberfläche. In beiden Fällen nutzt Doxygen zur Erstellung der Dokumentation eine Konfigurationsdatei, die für jedes Projekt erstellt werden muss. Die in der Konfigurationsdatei angegebenen Quellcode-Dateien durchsucht Doxygen nach Kommentaren, die es interpretieren kann. So wie Doxygen verschiedene Eingabedateiformate unterstützt, kann es auch Kommentare in unterschiedlichen Formaten lesen. Unter den Kommentarformaten ist auch das XML-Format von Microsoft zu finden (vgl. [Mic15a]).

5.1.5 Test

Viele integrierte Entwicklungsumgebungen besitzen bereits die Fähigkeit, Tests ausführen zu können. Während dies für Entwickler sehr nützlich oder sogar notwendig ist, kann diese Funktionalität zumeist nicht direkt in einem automatisierten Testprozess eingesetzt werden. Um Tests automatisiert durchzuführen, werden dafür konzipierte Softwarewerkzeuge eingesetzt. MSTest und VSTest.console sind zwei Werkzeuge, mit denen Tests ausgeführt werden können, die mit Visual Studios Unittest Framework erstellt wurden.

5.1.5.1 MSTest

MSTest [Mic15c] ist ein Kommandozeilenwerkzeug zur Ausführung von automatisierten Tests aus einer kompilierten Testdatei, die mit Visual Studio erstellt wurde. Unter den unterstützten Tests sind beispielsweise Unittests zu finden, Coded UI Tests hingegen werden nicht unterstützt.

Um MSTest ausführen zu können, müssen entweder Microsoft Visual Studio oder die Test Agents for Visual Studio installiert sein.

5.1.5.2 VSTest.console

VSTest.console [Mic15d] ist, wie MSTest (vgl. Abschnitt 5.1.5.1), ein Kommandozeilenwerkzeug, mit dem sich Testfälle ausführen lassen, welche in Visual Studio definiert wurden. Es unterstützt dabei Unittests und Coded UI Tests, welche dem Testen von grafischen Benutzeroberflächen dienen. VSTest wird in Visual Studio ab Version 2012 und an Stelle von MSTest eingesetzt. Wie MSTest wird auch VSTest.console von Visual Studio und den Test Agents for Visual Studio ausgeliefert.

5.1.6 Softwareverteilung und Feedback mit VersionInfo

Als Werkzeug, welches sich sowohl um die Verteilung und Versionsverwaltung von entwickelter Software kümmert als auch die Meldung von Feedback ermöglicht, wird das in der Abteilung entwickelte VersionInfo vorgestellt.

VersionInfo ist ein in der Abteilung AE-SAS entwickeltes Werkzeug zur Versionsverwaltung und Verteilung von Software und Dateien auf abteilungsinternen Computersystemen. Es ist in der Sprache C# geschrieben und findet nur innerhalb der Abteilung AE-SAS Anwendung. Die Einträge der Software umfassen jedoch nicht nur Produkte, die in der Abteilung entstanden sind, sondern auch eingesetzte Drittanbieterprodukte. Es wurde im Rahmen dieser Bachelorarbeit in seinem Funktionsumfang grundlegend erweitert. Die Funktionalität dieser Software zu Beginn der Arbeit ist in Abschnitt 3.1 beschrieben, die erfolgten Entwicklungen werden in Kapitel 6 erläutert.

5.2 Umsetzung

Das in Kapitel 4 beschriebene Konzept des Entwicklungsprozesses soll seinen Einsatz bei jeder Softwareentwicklung der Abteilung finden. Dabei erfolgt die Umsetzung für alle Entwicklungsprozesse unter Zuhilfenahme der Software, die in Abschnitt 5.1 vorgestellt wurde. In den folgenden Abschnitten wird die Umsetzung der einzelnen Schritte des Konzeptes und die Auswahl der jeweiligen Softwarewerkzeuge beschrieben.

Evaluiert wurde die Umsetzung, indem sie konkret auf die Entwicklung einer Software angewendet wurde. Um eine Entwicklung mit möglichst wenig Abhängigkeiten zu anderen Anwendungen für den Prozess auszuwählen, fiel die Wahl auf die Software VersionInfo (vgl. Abschnitt 5.1.6).

5.2.1 Aufgabenverwaltung

Als zentrales Kollaborationswerkzeug für den Entwicklungsprozess wurde Phabricator (vgl. Abschnitt 5.1.1) ausgewählt. Phabricator erfüllt alle an das Kollaborationswerkzeug gestellten Anforderungen und befindet sich im Institut bereits zur Evaluation im Einsatz. Aus diesem Grund läuft Phabricator auf einem separaten Linux-System, welches für die weiteren Betrachtungen in dieser Bachelorarbeit nicht relevant ist. Positiv zu sehen ist bei Phabricator, dass es sich unter aktiver Weiterentwicklung befindet. Außerdem kann Phabricator über die offene API leicht erweitert werden (vgl. Abschnitt 5.1.1).

Über die definierbaren Regeln in Phabricators Herald wurden Aktionen eingerichtet, die bei der Erstellung neuer Aufgaben die Entwickler eines Projektes über diese Aufgaben benachrichtigen. Die Entwickler können die Aufgaben anschließend auf dem Kanban-Board des Projektes organisieren und sie untereinander verknüpfen. Alle verfügbaren Informationen über zugehörige Commits, Builds oder Benutzer sind direkt in den Aufgaben ersichtlich.

5.2.2 Entwicklung

Die Wahl der Entwicklungsumgebung fiel auf Microsoft Visual Studio (vgl. Abschnitt 5.1.3.1), da sie zu den am weitesten entwickelten Entwicklungsumgebungen für die Programmiersprache C# gehört und von dem Unternehmen hinter der Sprache C# und des .NET-Frameworks, Microsoft, bereitgestellt wird. Außerdem wird sie in der Abteilung bereits seit einigen Jahren erfolgreich zur Entwicklung von .NET-Software verwendet. Um auf dem aktuellen Stand der Technik zu sein, wurde Visual Studio in der Version 2015 ausgewählt. Damit der volle Funktionsumfang zur Evaluation zur Verfügung steht, wurde die Enterprise-Variante eingesetzt.

Als Unterstützung für die Entwickler ist die Erweiterung ReSharper (vgl. Abschnitt 5.1.3.2) in Visual Studio integriert worden. Mit ihr soll die Qualität des Codes und die Produktivität der Entwickler erhöht werden, da auf häufige Fehlerquellen hingewiesen wird und umfangreiche Werkzeuge zur Codeerzeugung und zum Refactoring bereitgestellt werden.

Als Quellcodeverwaltung wurde das verteilte Versionsverwaltungssystem Mercurial (vgl. Abschnitt 5.1.2.1) ausgewählt. Die Eignung für den Prozess ist gegeben, da jede gewünschte Funktionalität unterstützt wird (vgl. Abschnitt 4.3.1). Es wurde Git (vgl. Abschnitt 5.1.2.2) vorgezogen, weil die Akzeptanz von Mercurial im Institut AE durch seine weite Verbreitung dort groß ist. Nennenswerte Unterschiede für den Entwicklungsprozess konnten zwischen dem, außerhalb des Instituts populäreren, Git und Mercurial nicht gefunden werden. Eine Unterstützung für Mercurial bieten alle verwendeten Softwarewerkzeuge, bei denen diese notwendig ist.

Die Entwicklung zur Erledigung einer Aufgabe erfolgt zu Beginn lokal. Die Entwickler haben hier bereits die Möglichkeit, die implementierten Testfälle mit Visual Studio auszuführen. Sobald die Entwicklung abgeschlossen ist, kann die Übertragung auf das Versionsverwaltungssystem stattfinden. Um mit Visual Studio direkt auf Mercurial-Repositories zugreifen zu können, findet die Erweiterung HgSccPackage ihren Einsatz. Damit können aus dem entwickelten Quellcode Commits erstellt werden, in denen sich direkt auf eine Aufgabe bezogen werden sollte. Möglich ist dies durch die Verwendung von Schlüsselwörtern in den Commit-Nachrichten, die von Phabricator interpretiert werden. So kann über die Worte „Fixes T123, T124“ angegeben werden, dass die darauf folgenden Phabricator-Tasks (hier: Aufgaben T123 und T124) mit dem zugehörigen Commit gelöst werden. Phabricator führt die entstehenden Aktionen aus, sobald der Commit mit den Repository per *Push* synchronisiert wurde. Tabelle 2 listet einige mögliche, vorangestellte Schlüsselwörter auf, die in Phabricator verwendet werden können.

Aktion	**Mögliche Schlüsselwörter**
Schließen einer Aufgabe als „Erledigt“	Closes, Fixes
Schließen einer Aufgabe als „Ungültig“	Invalidates
Schließen einer Aufgabe als „Wird nicht behoben“	Wontfixes
Anhängen des Commits an eine Aufgabe	Ref, References

Tabelle 2: Ausgewählte, von Phabricator interpretierte Schlüsselwörter in Commit-Nachrichten (vgl. [Pri14])

Die Entwickler werden nicht standardmäßig von Phabricator dazu verpflichtet, Verknüpfungen zwischen Commits und Tasks für jeden Commit herzustellen. Die Möglichkeit hierzu ist über die Blockierung von Commits ohne Verweis auf Phabricator-Aufgaben in der Nachricht

gegeben, die mit einer Herald-Regel umsetzbar ist. Dabei überprüft ein regulärer Ausdruck die Commit-Nachrichten auf Angaben von Aufgabennummern. Um keine Merge Commits, die beim Zusammenführen zweier Versionsstände entstehen, zu blockieren, kann für diese eine Ausnahmeregel erstellt werden. Eine derartige Blockierung ist für den Prozess sinnvoll, damit garantiert wird, dass für jede Entwicklungsaufgabe auch eine Phabricator-Aufgabe erstellt wird. Als nachteilig könnte es sich herausstellen, dass der systemseitige Zwang zu restriktiv ist und auf Ablehnung unter den Entwicklern stößt. In diesem Fall ist auf die Herald-Regel zu verzichten. Falls auf diese systemseitige Überprüfung verzichtet wird, sollten Entwickler in den Richtlinien auf die Benutzung von Verknüpfungen zwischen Commits und Tasks hingewiesen werden.

In einzelnen Fällen kann die Entwicklung größerer Aufgaben auf separaten Zweigen des Repositories, sogenannten *Branches*, erfolgen. In diesem Fall wird für eine zu erledigende Aufgabe ein passender Branch auf dem Repository angelegt, woraufhin die zugehörigen Commits auf diesen Branch übertragen werden. Solche Branches, die der Entwicklung einzelner Funktionalitäten dienen, werden häufig als *Feature Branches* bezeichnet. Problematisch kann sich das Zusammenspiel zwischen *Feature Branches* und einem System zur kontinuierlichen Integration gestalten (vgl. [Wol15]). Da nur für den *default-Branch* ein vollständiges System zur kontinuierlichen Integration umgesetzt wurde, ist es unbedingt notwendig, dass die Entwickler die *Feature Branches* direkt nach Abschluss der Entwicklung in den *default-Branch* integrieren (vgl. Abschnitt 5.2.4). Ansonsten finden die Qualitätssicherungsmaßnahmen des Prozesses für die Entwicklungen dieser Branches keine Anwendung. Im Allgemeinen sollte, passend zum Prinzip „Build Quality In“ der kontinuierlichen Auslieferung nach [HF11], die Integration von Quellcode in den *default-Branch* möglichst häufig stattfinden (vgl. Abschnitt 2.4).

5.2.3 Code Review

Die Möglichkeit zur Durchführung von Code Reviews wurde ebenfalls mit Phabricator umgesetzt. Da nach dem Konzept des Entwicklungsprozesses Code Reviews sowohl vor als auch nach einem Push auf das Repository möglich sein sollen, wurden beide Möglichkeiten umgesetzt und evaluiert. Abbildung 10 stellt die Abläufe von Pre- und Post-Push Code Reviews gegenüber. Bei Pre-Push Reviews (grün) folgt der Schritt des Code Reviews (rot) auf das Hochladen der Änderungen (Diffs). In diesem Fall ist das Code Review blockierend für den Buildprozess des Commits. Ein Push kann erst nach erfolgtem Code Review ohne weitere Änderungswünsche stattfinden. Bei Post-Push Reviews (blau) kann ein Commit direkt

per Push hochgeladen werden, woraufhin der Buildprozess beginnt. Ein Review des Commits kann zu einem beliebigen Zeitpunkt stattfinden, wobei auftretende Änderungswünsche in einer weiteren Entwicklung resultieren.

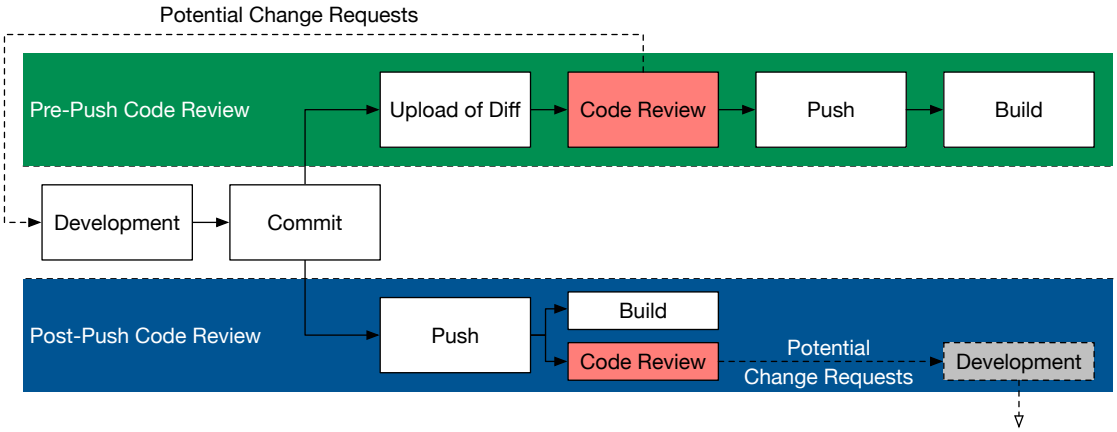


Abbildung 10: Vergleich der Abläufe von Pre- und Post-Push Code Reviews.

Um Code Reviews vor einem Push durchzuführen, müssen sogenannte Commit Hooks eingerichtet werden. Sie sorgen in diesem Fall dafür, dass Commits vom Repository abgewiesen werden, wenn kein Code Review zu dem betreffenden Commit durchgeführt wurde. Um dieses Code Review vorher durchzuführen, erstellen die Entwickler lokal einen Commit. Anschließend schicken sie dessen Diff mit dem Kommandozeilenwerkzeug Arcanist an Differential (vgl. Abschnitt 5.1.1). Zu dem Commit wird eine Aufgabe angegeben und ein oder mehrere Benutzer, die das Code Review durchführen sollen. Diese Benutzer, die Reviewer, haben die Möglichkeit, den Quellcode zu kommentieren und können ihn nach dem Review entweder akzeptieren oder Nachbesserung fordern. Wird ein Code Review akzeptiert, kann der Entwickler des Quellcodes mit einem Push seinen Commit in das Repository laden. Dadurch wird der Build-Schritt angestoßen (vgl. Abschnitt 5.2.4). Falls keine Pre-Push Code Reviews durchgeführt werden, kann ein Commit einfach, ohne vorheriges Anlegen eines Differential-Diffs, auf das Repository geladen werden. Die Möglichkeit eines nachfolgenden Post-Push Code Reviews besteht weiterhin zu jedem Zeitpunkt.

Die Durchführung von Post-Push Code Reviews erfordert keine Einrichtung von Commit Hooks. Nach Wunsch kann jedoch eine Herald-Regel erstellt werden, mit der neue Commits einem Reviewer zugewiesen werden. Post-Push Code Reviews werden über die Phabricator-Anwendung

Audit durchgeführt und dort als Audits bezeichnet (vgl. Abschnitt 5.1.1). Durchgeführt werden können sie zu jeder Zeit für jeden Commit, der in Phabricator registriert ist.

5.2.4 Build

Die nachfolgende Beschreibung zur Umsetzung des Buildprozesses bezieht sich konkret auf die Ausführung auf dem Buildsystem. Allerdings sind sämtliche, zur Erstellung der Anwendung notwendige Schritte auch lokal von den Entwicklern ausführbar. Dies ist für manuelle Tests genauso notwendig wie für die Entwicklung ohne Verfügbarkeit des Buildsystem. In diesem Fall entfallen Phabricator und Jenkins aus den Prozess, der Build wird nun von Visual Studio gestartet.

Der automatisierte Buildprozess, der auf einem separaten Microsoft Windows 7-System ausgeführt wird, wird von Phabricator initiiert. Dies geschieht über eine Herald-Regel. Sie besagt, dass für jeden akzeptierten Commit, der auf den *default-Branch* des angegebenen Repositories gepusht wird, nach dem Push ein Harbormaster-Build Plan ausgelöst. Der *default-Branch* ist in diesem Fall der Hauptzweig des Mercurial-Repositories. Phabricator überwacht das vom Entwicklungsprojekt verwendete Repository, um die Regel im Fall eines Commits auszulösen.

Zur Steuerung, ob auf einen Push ein Build mit Verteilung folgen soll, wurden verschiedene Möglichkeiten umgesetzt. Während der Erstellung eines Commits ist es möglich, die Commit-Nachricht mit einem Schlüsselwort zu versehen, damit auf deren Push kein Build folgt. Dafür wurde das Schlüsselwort „#noRelease“ angelegt. Sobald dieses in der Commit-Nachricht auftaucht, wird kein Build Plan ausgeführt. Dies bedeutet, dass die im Commit enthaltenen Änderungen auch nicht getestet werden und keine zugehörige Softwareversion verteilt wird. Hier ist ein Widerspruch zu den in Abschnitt 2.4 genannten Prinzipien der kontinuierlichen Auslieferung zu sehen: Das Prinzip „Done Means Released“ kann verletzt werden, wenn die mit dem Schlüsselwort versehenen Commits zur Erledigung von Aufgaben dienen. Dieses Schlüsselwort sollte deshalb nicht bei regulären Entwicklungsaufgaben eingesetzt werden, sondern nur in besonderen Fällen, in denen dies notwendig ist. Beispielsweise kann durch das Schlüsselwort verhindert werden, dass der gleichzeitige Push von vielen Commits auf einmal zu genauso vielen Buildvorgängen wie Commits führt. Dies kann konkret auftreten, wenn während einer auswärtigen Messkampagne ohne Verfügbarkeit des Buildsystems viele Commits

erzeugt werden, die nach der Rückkehr auf das von Phabricator überwachte Repository geladen werden.

Sollte der Commit nicht mit dem Schlüsselwort „#noRelease“ versehen worden sein, werden Build und Tests in jedem Fall durchgeführt. Danach gibt es zwei weitere, mögliche Vorgehensweisen für die Art der Verteilung. Die erstellte Softwareversion könnte als aktuellste Version an die Nutzer verteilt werden, so dass diese eine Meldung darüber erhalten. Alternativ ist es möglich, die Version für Anwender verfügbar zu machen, jedoch die Kennzeichnung als aktuellste Version manuell vorzunehmen. Die Diskussion dieser Funktionalität erfolgt in Abschnitt 5.2.6.

Der für das Entwicklungsprojekt angelegte Build Plan besteht aus nur einem einzigen Schritt: Der Durchführung einer Hypertext Transfer Protocol (HTTP)-Anfrage an den Jenkins-Server auf dem Buildsystem. In dieser Anfrage wird Jenkins mitgeteilt, dass der Build des Projekts gestartet werden soll. Zusätzlich erhält Jenkins einige Informationen vom Phabricator:

- Einen Identifikator (ID) für den Commit.
- Einen Phabricator Identifikator (PHID) für den Build. Er wird benötigt, um nach Abschluss des Builds Rückmeldungen von Jenkins an Phabricator zu senden.
- Eine Build ID. Auch sie stammt, wie die PHID des Builds, aus Phabricator. Im Unterschied zu der PHID enthält sie jedoch keine Buchstaben oder Sonderzeichen. Stattdessen ist sie eine Nummer, die von Phabricator mit jedem Build erhöht wird.

Mit den erhaltenen Informationen kann Jenkins zum tatsächlichen Build-Prozess übergehen. Dafür lädt Jenkins den aktuellen Stand des Repositories herunter und setzt es auf den Commit, dessen Identifikator in der Anfrage mitgeteilt wurde. Nun ist das lokale Repository von Jenkins auf dem Stand, wie es zum Zeitpunkt des Commits bei dem Entwicklers war. Falls ein Backup des Repositories gewünscht ist, kann dieses nun, nach der Synchronisation, durchgeführt werden.

Im nächsten Schritt wird MSBuild (vgl. Abschnitt 5.1.4.2) ausgeführt. Die den Build-Prozess beschreibende MSBuild-Projektdatei befindet sich im Repository, so dass sie bei Bedarf von den Entwicklern angepasst werden kann. MSBuild wird parametrisiert mit dieser Projektdatei und der von Phabricator übergebenen Build ID gestartet. Die MSBuild-Projektdatei verweist auf die Projektmappe des Entwicklungsprojekts. In der Projektmappe ist angegeben, welche Projekte kompiliert werden müssen und welche Abhängigkeiten zwischen den zu kompilierenden

Projekten existieren. Die Auflösung der Abhängigkeiten zu einer Abfolge an Build-Aktionen übernimmt MSBuild, welches die Build-Aktionen nachfolgend auch direkt ausführt.

Unter diesen Build-Aktionen sind nicht nur Kompilierungen von Projekten, sondern auch weitere notwendige Aufgaben zur Erstellung eines Softwareprodukts. Eine dieser Aufgaben ist etwa die Änderung von Quellcode-Dateien, in denen die Versionsnummern der Ausgabedateien angegeben sind. Mit der als Parameter erhaltenen Build ID werden die Versionsnummern durch die MSBuild Community Tasks aktualisiert.

Außerdem ist es vor Erstellung der Dokumentation notwendig, dass die Änderungsprotokolle aktualisiert werden. Als Werkzeug zur Erstellung der Dokumentation wurde SHFB ausgewählt, da es sich einfacher in den MSBuild-basierten Prozess integrieren lässt als Doxygen (vgl. Abschnitt 5.1.4.4). Die Aktualisierung der Änderungsprotokolle erfolgt über die Software VersionInfo auf Grundlage von Logdateien des Repositories. Eine detaillierte Beschreibung dieser Funktionalität findet in Abschnitt 6.2 statt.

Sobald alle Anwendungsteile und die Dokumentation erstellt wurden, folgt die Erzeugung der Installationsdateien. Das WiX Toolset übernimmt diese Aufgabe. Durch den automatisierten Prozess wird ein standardisiertes Verfahren zur Erstellung und eine gleichbleibende Gestaltung der Installationsroutine sichergestellt. Die Gestaltung wurde passend zu dem Corporate Design des DLRs erstellt.

Falls der Build fehlschlägt, meldet Jenkins dies über einen Aufruf des Kommandozeilenwerkzeugs Arcanist an die Conduit-API von Phabricator zurück. Hierbei kommt die zuvor übertragene Build PHID zum Einsatz, die nun mit dem Buildresultat an Phabricator zurückgesendet wird und die Verknüpfung zum gestarteten Build ermöglicht. Sobald Phabricator die Informationen zu dem Build erhalten hat, kann der Buildstatus auf der Weboberfläche eingesehen werden. Bei einem Fehlschlag wird außerdem der Entwickler des Commits benachrichtigt.

Bei erfolgreichen Builds folgt auf den Buildprozess der Testprozess (siehe Abschnitt 5.2.5).

5.2.5 Test

Wie bereits in Abschnitt 5.2.2 erwähnt, haben die Entwickler bereits vor dem Commit die Möglichkeit zur Durchführung der implementierten Testfälle. Diese lokalen Tests vor dem Laden des Quellcodes auf das Repository sollten, besonders bei großen Änderungen, durchgeführt werden, um den Aufwand zur Behebung von Fehlern gering zu halten (vgl.

Kapitel 4.6). Es ist besonders zu beachten, dass manuelle und nicht automatisierbare Tests auf dem Buildsystem nicht ausgeführt werden. Diese müssen entweder von den Entwicklern vor dem Push oder von beliebigen Anwender nach der Softwareverteilung durchgeführt werden.

Die Dokumentation der verfügbaren und verwendeten Testdaten ist, wie in Abschnitt 4.6.5 beschrieben, für alle Testfälle in einer Testdatenbank vorzunehmen. Die Testdatenbank wurde mit der Software Microsoft Access [Mic15h] umgesetzt, die DLR-weit ohne zusätzliche Kosten verfügbar ist. Da die konkrete Umsetzung der Testdatenbank mit einem Softwarewerkzeug für diese Bachelorarbeit nicht relevant ist, wird an dieser Stelle nicht weiter auf Microsoft Access eingegangen.

Der automatisierte Test auf dem Buildsystem wurde mit VSTest.console und der Jenkins-Erweiterung VsTestRunnerPlugin umgesetzt (vgl. Abschnitte 5.1.5.2 und 5.1.4.1). VsTestRunnerPlugin startet dafür VSTest.console mit der entsprechend angegebenen Bibliotheksdatei, die die auszuführenden Testfälle enthält. Dabei ist auch die Verwendung mehrerer Programmbibliotheken, die Testfälle beinhalten, möglich. Eine optionale Einstellungsdatei erlaubt weitere Anpassungen der automatisierten Testausführung. Die während der Ausführung der Tests erzeugten Logdateien und Testergebnisse werden abgespeichert und können archiviert werden. Mit ihrer Hilfe ist es zum Beispiel möglich, fehlgeschlagene Tests zu debuggen.

Durch den Einsatz von Visual Studio Enterprise konnte auch die Möglichkeit zur Durchführung von automatisierten GUI Tests evaluiert werden (vgl. Abschnitt 5.1.3.1). Als Coded UI Tests bezeichnete GUI Tests lassen sich in Visual Studio entwerfen und ausführen. Außerdem können die Coded UI Tests, dank VSTest.console, auch auf dem Buildsystem automatisiert ausgeführt werden. Dafür wird, wie bei allen automatisierten Testfällen, eine Programmbibliothek erzeugt, die alle Testmethoden enthält. Somit können Coded UI Tests zur automatisierten Ausführung von System- und Akzeptanztests verwendet werden.

Sollte die automatische Ausführung einiger System- oder Akzeptanztests nicht möglich sein, müssen die entsprechenden Testfälle mittels manueller Tests abgearbeitet werden. In diesem Fall sind Checklisten zu verwenden, um alle notwendigen Schritte der Tests zu dokumentieren. Außerdem tragen die Checklisten zur Übersichtlichkeit während der Ausführung bei, da erledigte Aktionen als solche gekennzeichnet werden können und die Checklisten damit wie Testprotokolle verwendet werden können (vgl. [Hof13, S. 324 ff.]). Die Durchführung der manuellen Tests findet nach dem Verteilungsprozess statt, wobei die Benutzer in diesem Fall keine Benachrichtigung für die erstellte Softwareversion erhalten. Eine detaillierte Beschreibung dieses Ablaufs erfolgt in Abschnitt 5.2.6. Zur Dokumentation der Testdurchführung ist es

sinnvoll, die ausgefüllten Checklisten zusammen mit eventuell entstandenen Anmerkungen zu archivieren.

Beim Auftreten eines Fehlers in einem der Tests markiert Jenkins den Build ebenfalls als fehlerhaft. Die Mitteilung dazu wird, wie auch bei einem Fehler im Buildprozess, an Phabricator übermittelt. In diesem Fall wird ebenfalls der Entwickler des Commits benachrichtigt.

Sollten alle Tests erfolgreich abgeschlossen worden sein, wird Phabricator darüber informiert und die Softwareverteilung beginnt (vgl. Abschnitt 5.2.6).

5.2.6 Softwareverteilung

Auf den Build und die Tests der entwickelten Softwareversion folgt deren Verteilung. Dabei kommt das in dieser Bachelorarbeit weiterentwickelte VersionInfo zum Einsatz. Ausgelöst wird der Verteilungsprozess von Jenkins. Jenkins tätigt einen Kommandozeilenaufruf an die für die Verteilung von Software zuständige Komponente von VersionInfo, den VersionInfoManager. Bei dem Aufruf wird eine Konfigurationsdatei angegeben. In ihr sind die von VersionInfo zur Verteilung benötigten Informationen enthalten. Anhang C.1 zeigt eine solche Konfigurationsdatei, die für den umgesetzten Verteilungsvorgang verwendet wird. Angegeben sind neben Informationen zum Zielserver, auf dem die Version abgelegt werden soll, einige Pfade zur Erstellung der Änderungsprotokolle.

Durch die Ausführung der Verteilungskomponente von VersionInfo wird die neue Softwareversion auf den in der Konfigurationsdatei angegebenen Pfad auf dem Dateiserver der Abteilung kopiert und in die dortige VersionInfo-Versionsdatenbank eingetragen. Mit Hilfe des neuen Eintrags in die, als XML-Datei umgesetzte, Versionsdatenbank wird die Softwareversion allen Nutzern von VersionInfo zugänglich gemacht.

In der Konfigurationsdatei des Verteilungsprozesses kann festgelegt werden, ob neue Softwareversionen als aktuellste Versionen der Software markiert werden sollen. Je nach Bedeutung und Schutzbedarfskategorie der Anwendung sollte für jedes Entwicklungsprojekt einzeln entschieden werden, ob ein automatisiertes Setzen jeder neu erstellten Version als aktuellste Softwareversion gewünscht ist oder nicht. Für kritische Anwendungen ist eher davon abzusehen als bei unkritischen. Sollten beispielsweise noch manuelle Tests der Softwareversion anstehen, ist ebenfalls davon Abstand zu nehmen. Damit obliegt dieser letzte Überprüfungsschritt einem Entwickler, testenden Anwendern oder dem Administrator von VersionInfo.

Sobald in der Versionsdatenbank eingetragen wird, dass eine neue Softwareversionen als aktuellste Versionen gesetzt wurde, informiert VersionInfo die Benutzer der Software darüber. Abbildung 11 zeigt, wie die Benachrichtigung über dem VersionInfo-Symbol in der Taskleiste von Windows 7 angezeigt wird. Notwendig für diese Benachrichtigung ist, dass die Benutzer VersionInfo während der Eintragung ausführen. Sollten die Benutzer VersionInfo erst nach erfolgter Eintragung starten, wird ihren die Verfügbarkeit einer neuen Version über eine farbige Hervorhebung der entsprechenden Software signalisiert (vgl. Abbildung 15 in Abschnitt 6.1).

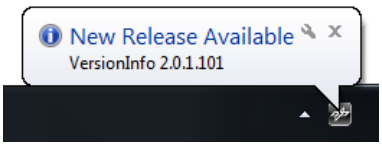


Abbildung 11: Benachrichtigung von VersionInfo über eine neue Softwareversion.

Aufgrund von DLR-internen Einschränkungen ist eine automatisierte Installation neuer Softwareversionen auf den Systemen der Abteilung AE-SAS ohne Eingreifen der Benutzer aktuell nicht umsetzbar. Grund dafür ist die restriktive Konfiguration der sogenannten Benutzerkontensteuerung von Windows 7, die DLR-weit vorgeschrieben ist. Ungeachtet dessen erlauben sowohl VersionInfo als auch die erstellten Installationsdateien eine Nutzung der automatisierten Installation, so dass diese Funktionalität bei einer eventuellen Änderung der Einschränkungen zum Einsatz kommen könnte. Bis dahin kann die Installation neuer Softwareversion von den Benutzern direkt aus VersionInfo heraus aufgerufen werden. Falls es zu einem beliebigen Zeitpunkt, beispielsweise aus Kompatibilitätsgründen, notwendig sein sollte, eine ältere Softwareversion zu nutzen, ist auch dies mit VersionInfo möglich. Dafür kann in VersionInfo jede, in der Versionsdatenbank eingetragene Version ausgewählt und installiert werden.

5.2.7 Feedback an die Entwickler

Benutzer der entwickelten Software haben die Möglichkeit, Feedback zu der Software über VersionInfo an die Entwickler zu senden. Dafür wird für jedes gesendete Feedback eine Aufgabe in Phabricator erstellt, die mit dem betreffenden Phabricator-Projekt verbunden wird. Durch die Verbindung mit dem Phabricator-Projekt erhalten die Entwickler, die dem Projekt zugeordnet sind, eine Benachrichtigung über neu hinzugefügte Aufgaben. Dafür ist die zusätzliche Einrichtung einer Herald-Regel notwendig, die bei neu erstellten Aufgaben

ausgeführt wird (vgl. Abschnitt 5.2.1). Neben den Entwicklern werden auch die Nutzer, die Feedback melden, über die Aufgabe auf dem aktuellen Stand gehalten. Sie erhalten beispielsweise eine Benachrichtigung, wenn die Aufgabe einem Entwickler zugewiesen oder sie als erledigt markiert wird.

Die Benachrichtigungen von Phabricator können, je nach Benutzerwunsch, in Form einer E-Mail versendet oder ausschließlich in Phabricator angezeigt werden. Durch E-Mail-Benachrichtigungen kann sichergestellt werden, dass Nutzer über den aktuellen Stand der Entwicklung informiert belieben, auch wenn sie sich nicht regelmäßig im Phabricator informieren.

Die Anmeldung bei Phabricator und die Verknüpfung einer Aufgabe mit dem meldenden Benutzer erfolgt über ein Conduit Authentifizierungstoken (vgl. Abschnitt 5.1.1). Das Authentifizierungstoken ist zwingend notwendig, um die Feedback-Meldung über Phabricator zu Nutzen. Benutzer müssen sich, um ein solches Token zu erhalten, einmalig mit ihren DLR-internen LDAP-Zugangsdaten beim Phabricator einloggen und es dort generieren. Das dauerhaft gültige Authentifizierungstoken hinterlegt der Nutzer anschließend in den Einstellungen von VersionInfo, die benutzerspezifisch auf dem Computer gespeichert werden.

Für die Meldung von Feedback stellt VersionInfo mehrere Oberflächen bereit. Abbildung 12 stellt die Oberfläche dar, welche zur Meldung von Fehlern verwendet wird. Der Nutzer kann einen Titel, die Version, eine Priorität und eine Beschreibung des Fehlers angeben. Die Versionsnummer wird automatisch anhand der installierten Versionen ausgewählt. Die Gestaltung und Funktionalität der Feature Request-Oberfläche unterscheidet sich nicht wesentlich von der in Abbildung 12 dargestellten Bug Report-Oberfläche.

An Feedback, das mit VersionInfo erstellt wird, können Nutzer Anhänge anfügen. Abbildung 13 stellt die, für diesen Zweck implementierte, Oberfläche dar. Entwickler haben die Möglichkeit, in der Versionsdatenbank empfohlene Dateien zu definieren, die den Nutzern als Anhänge zu dem Feedback vorgeschlagen werden. Die vorgeschlagenen Dateien können für Feature Requests und Bug Reports unterschiedlich sein. Die Benutzer dürfen entscheiden, ob sie diese empfohlenen Anhänge oder beliebige andere Dateien mitsenden wollen. Sämtliche, angehängte Dateien werden zu der Phabricator-Aufgabe hinzugefügt und können von allen Nutzern in der Aufgabenbeschreibung angesehen und heruntergeladen werden.

Häufig kann es Entwicklern bei der Erledigung von Aufgaben und der Behebung von Fehlern helfen, wenn ihnen passende Bildschirmfotos bereitgestellt werden. Die Aufnahme von Bild-

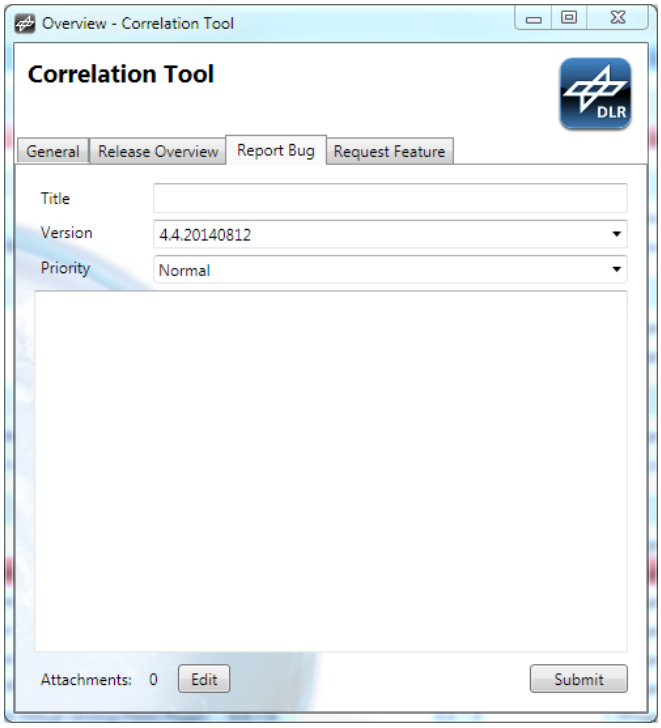


Abbildung 12: Oberfläche zur Meldung von Fehlern in VersionInfo.

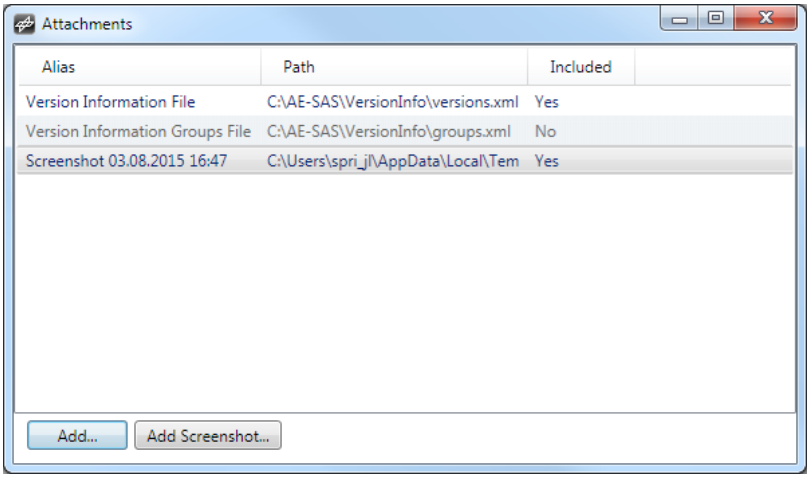


Abbildung 13: Oberfläche zum Anfügen von Anhängen an Feedback-Meldungen in VersionInfo.

schirmfotos wird ebenfalls von VersionInfo unterstützt. In der Oberfläche zur Übersicht über die Anhänge ist für diesen Zweck ein Button zur Aufnahme eines Bildschirmfotos vorhanden (vgl. Abbildung 13, Button „Add Screenshot...“). Bei einem Klick auf den Button werden sämtliche Fenster von VersionInfo minimiert und der Benutzer kann den Bildschirmbereich auswählen, der auf dem Bildschirmfoto zu sehen sein soll. Abbildung 14 zeigt beispielhaft, wie die Aufnahme der Bildschirmfotos für den Nutzer dargestellt wird.

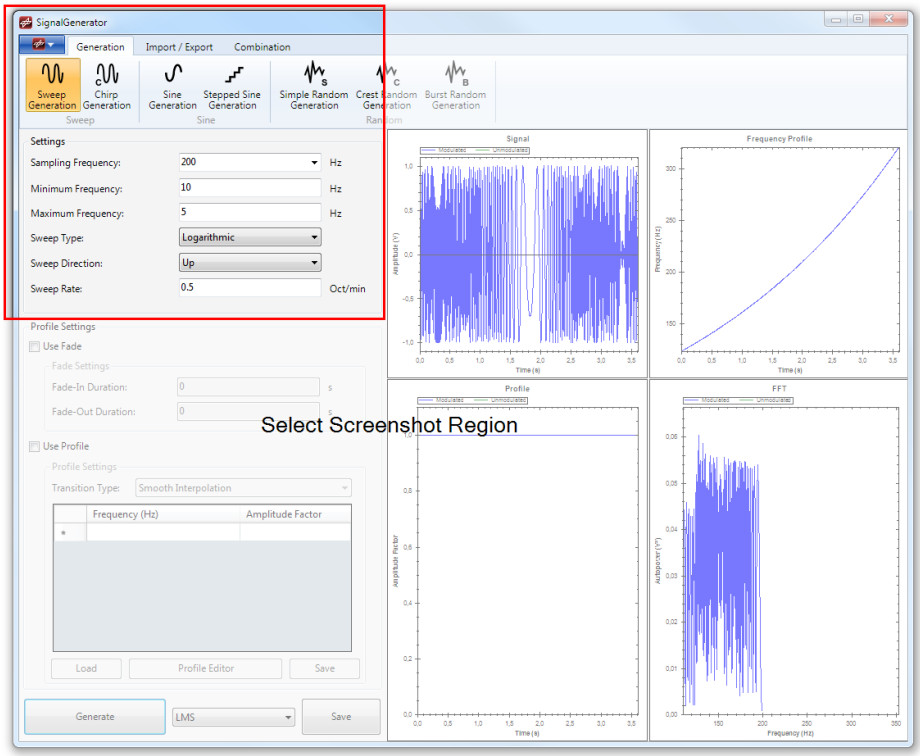


Abbildung 14: Aufnahme von Bildschirmfotos mittels VersionInfo.

Sämtliche, für VersionInfo entworfene und implementierte Oberflächen zur Meldung von Feedback, können direkt in andere Software eingebunden werden. Somit ist es nicht notwendig, für jedes Feedback VersionInfo aufzurufen. Stattdessen kann das Feedback direkt aus der betreffenden Software als Aufgabe an Phabricator übermittelt werden. Die Meldung über VersionInfo ist in jedem Fall weiterhin möglich, damit Nutzer auch bei schwerwiegenden Softwareproblemen, die eine Meldung aus der entwickelten Software heraus verhindern, ihre Probleme an die Entwickler übermitteln können.

6 Entwicklungen an der Software VersionInfo

Die Anwendung VersionInfo dient in der Abteilung AE-SAS zur Verwaltung und Verteilung von Software und Dateien. Um den Anforderungen, die im Rahmen dieser Bachelorarbeit hinzugekommen sind, entsprechen zu können, wurde VersionInfo grundlegend überarbeitet.

In der Umsetzung des Entwicklungsprozesses ist die Software VersionInfo für die Verteilung neuer Softwareversionen vorgesehen (vgl. Abschnitt 5.2.6). Die Entwicklung der dafür notwendigen Funktionalität wird in Abschnitt 6.2 erläutert. Des Weiteren ist es Teil des Prozesses, dass die Änderungsprotokolle der entwickelten Software durch Informationen zu den erledigten Aufgaben ergänzt werden. Außerdem müssen sie in ein Format gebracht werden, welches sich für die weitere Verwendung eignet. Die Implementierungen hierfür werden in Abschnitt 6.3 beschrieben. Um die Entwickler bei Verfügbarkeit einer neuen Softwareversion informieren zu können, ist ein entsprechendes Benachrichtigungssystem in der Software VersionInfo notwendig, welches in Abschnitt 6.4 vorgestellt wird. Außerdem ist es eine Anforderung des Entwicklungsprozesses, dass Benutzer Feedback zur Software geben können (vgl. Abschnitt 5.2.7). Auch hierfür wird eine Umsetzung in VersionInfo benötigt, welche in Abschnitt 6.5 beschrieben wird.

Zusätzlich zu den Funktionserweiterungen wurden tiefgreifende Überarbeitungen an der Oberfläche vorgenommen, wodurch Änderungen an der Softwarearchitektur notwendig wurden. Ihre Beschreibung erfolgt in Abschnitt 6.1.

6.1 Überarbeitung der Oberfläche

Die Oberfläche von VersionInfo basierte ursprünglich auf der Grafikbibliothek Windows Forms [Mic15k], welche Teil des .NET Frameworks ist. Um VersionInfo mit den restlichen C#-Anwendungen der Abteilung anzugleichen, wurde die Windows Forms-Oberfläche durch eine Oberfläche auf Basis des Windows Presentation Foundation (WPF) Frameworks [Mic15e] ersetzt. WPF stellt eine moderne Alternative zu Windows Forms dar und wurde mit dem .NET Framework in Version 3.0 eingeführt.

Passend zur WPF-Oberfläche findet mit der Weiterentwicklung das Entwurfsmuster Model View ViewModel (MVVM) [Smi09] in VersionInfo seinen Einsatz. WPF verwendet die Mar-

kupsprache Extensible Application Markup Language (XAML) zur Definition des Designs von Oberflächen, welche die Views des MVVM-Entwurfsmusters darstellen. In den Views ist definiert, was den Nutzern der Software angezeigt wird. Dazu gehören das Aussehen und die Datenverbindungen der Oberflächen. Das Model enthält die Daten, auf welche die Software zugreift. Das ViewModel stellt die Verbindungsschicht zwischen View und Model dar und stellt die Datenverbindungen zum Model an den View bereit.

Die neue Oberfläche zur Ansicht aller Einträge der Versionsdatenbank ist in Abbildung 15 zu sehen. Zusätzlich wurde für die Einträge ein Übersichtsfenster erstellt, welches in Abbildung 16 zu sehen ist. Das Übersichtsfenster enthält neben allgemeinen Informationen über einen Eintrag sämtliche verfügbare Versionen des Eintrags mit dem jeweiligen Änderungsprotokoll. Außerdem besteht dort die Möglichkeit, Feature Requests und Bug Reports zu erstellen (vgl. Abbildung 12 in Abschnitt 5.2.7).

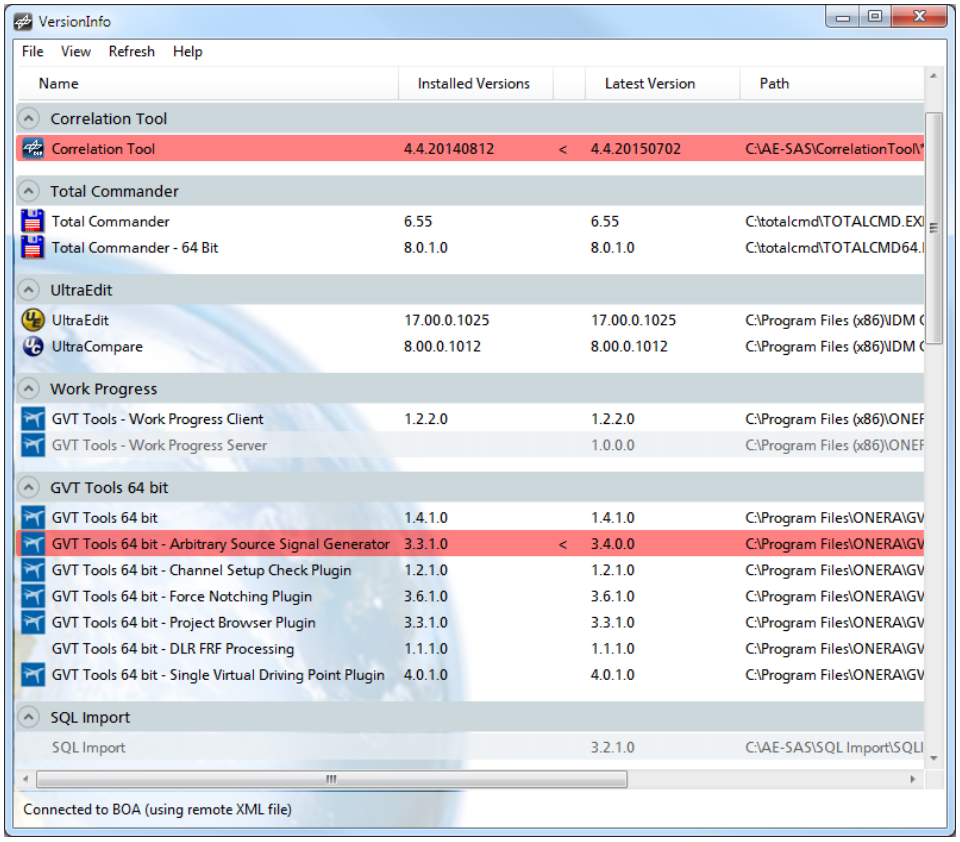


Abbildung 15: Die Oberfläche von VersionInfo nach Abschluss der Überarbeitungen.

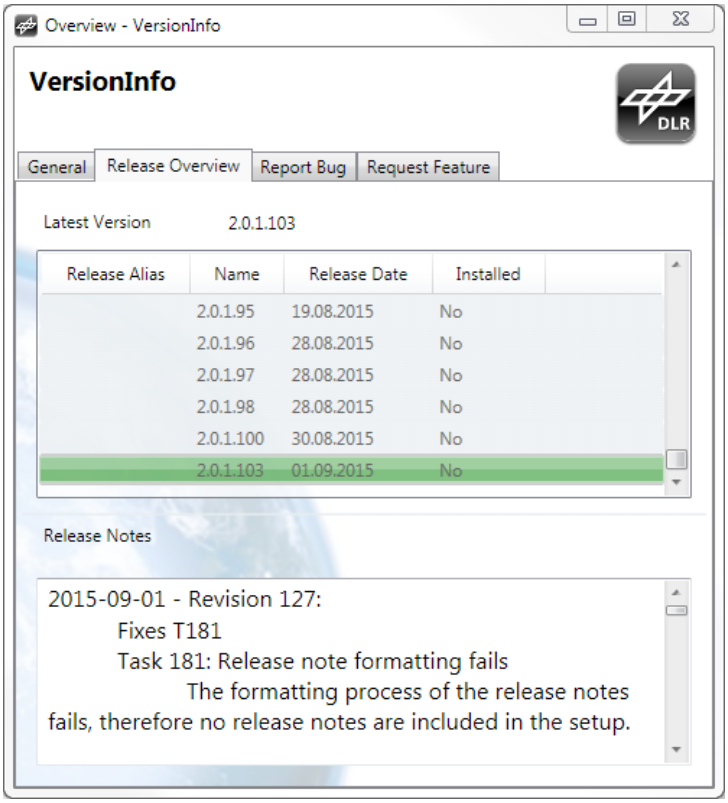


Abbildung 16: Das Übersichtsfenster von VersionInfo.

6.2 Softwareverteilung

Die Verteilung von neu erstellten Softwareversionen erfolgt über eine gesonderte Komponente von VersionInfo, den VersionInfoManager. Er dient der Interaktion mit der Versionsdatenbank auf Kommandozeilenebene. Der VersionInfoManager wurde im Rahmen der Arbeit für den konzipierten Entwicklungsprozess entwickelt.

Um die Verteilung einer neuen Softwareversion anzustoßen, wird der VersionInfoManager mit einer Konfigurationsdatei als Parameter gestartet. Anhang C.1 zeigt beispielhaft eine solche Konfigurationsdatei (vgl. Abschnitt 5.2.6). Ein Kommandozeilenaufruf wie

```
VersionInfoManager.exe add-release "dotNET\DeploymentConfig.xml"
```

führt zur Verteilung einer Softwareversion.

Die Versionsnummer, welche die erstellte Version tragen soll, wird über die Dateiinformationen der Software automatisch ermittelt. Für diese Versionsnummer wird nun ein Eintrag in der XML-Versionsdatenbank erstellt, die in der Konfigurationsdatei angegeben ist. Anhang C.2 zeigt an einem Beispiel, wie die Einträge in der Versionsdatenbank aufgebaut sind. Während der Verteilung werden die für die Installation der Version benötigten Dateien an ihren vorgesehenen Speicherort kopiert und die neuen Pfade in den Eintrag mit aufgenommen. Außerdem wird ein Hash der ausführbaren Datei der Software erstellt und hinzugefügt. Änderungsprotokolle, die vor der Verteilung erstellt werden müssen, werden ebenso ergänzt wie ein Veröffentlichungsdatum.

In der Konfigurationsdatei kann auch angegeben werden, ob die veröffentlichte Version als aktuellste Version gekennzeichnet werden soll. In diesem Fall werden die Benutzer der Software über die Aktualisierung hingewiesen. Weitere Informationen hierzu sind in Abschnitt 6.4 beschrieben.

6.3 Erstellung der Änderungsprotokolle

Vor der Verteilung neuer Softwareversionen müssen die Änderungsprotokolle der Software aufbereitet werden. Auch dafür lässt sich der VersionInfoManager einsetzen. Als Eingabe sind die Änderungsprotokolle eines Mercurial-Repositories in Textform vorgesehen, welche die vorangegangenen Commits mit ihren zugehörigen Nachrichten enthalten. Grundsätzlich kann jedoch auch jedes andere textbasierte Format verwendet werden. Der Aufruf des VersionInfoManagers erfolgt mit Angabe einer Konfigurationsdatei, welche die Pfade der aufzuarbeitenden Änderungsprotokolle und der Vorlagen für die Ausgabeformate enthält:

```
VersionInfoManager.exe prepare-release-notes "dotNET\DeploymentConfig.xml".
```

Der VersionInfoManager durchsucht die kompletten Änderungsprotokolle nach referenzierten Aufgaben aus Phabricator. Diese werden mit Hilfe eines regulären Ausdrucks im Text gefunden. Als Suchtext für Phabricator-Aufgabennummern kommt der reguläre Ausdruck `[\s]+[Tt][\s]*([0-9]+)` zum Einsatz. Er findet Angaben von Aufgabennummern aus Commit-Nachrichten der Form „Fixes T123, T 124 t125“und akzeptiert dabei verschiedene Schreibweisen für die Aufgabennummern. Schlüsselworte, wie das „Fixes“ in genanntem Beispiel, sind für die Erkennung nicht notwendig. Anschließend werden nachfolgend auf die Textzeilen, in denen die Aufgabennummern enthalten sind, detaillierte Beschreibungen zu den Aufgaben aus Phabricator angefügt. Hierfür stellt VersionInfoManager eine Verbindung zu Phabricators

Conduit-API her und fragt die Titel und Beschreibungen der angegebenen Aufgaben ab (vgl. Abschnitt 5.1.1).

Nachdem die Anreicherung der Änderungsprotokolle stattgefunden hat, werden sie für mehrere Ausgabeformate passend formatiert. Implementiert wurde die Ausgabe von Änderungsprotokollen im Rich Text Format (RTF)- und im MAML-Format. Die Änderungsprotokolle im RTF-Format werden in zwei Ausführungen erstellt: Eine Variante zur Anzeige innerhalb von VersionInfo, eine weitere Variante als druckfähiges Dokument im Corporate Design. Anhang C.3 zeigt ein Beispiel solcher, angereicherter Änderungsprotokolle im Corporate Design.

6.4 Benachrichtigungssystem

Um den Benutzern von VersionInfo Benachrichtigungen über neue Softwareversionen bieten zu können, überwacht VersionInfo, in der nun überarbeiteten Version, in regelmäßigen Zeitabständen die Versionsdatenbank auf Änderungen. Das verwendete Intervall kann vom Benutzer in den Einstellungen von VersionInfo verändert werden. Es sollte nicht zu klein gewählt werden, da die Versionsdatenbank zumeist auf einem zentralen Server abgelegt ist und die Netzwerklast durch eine häufige Aktualisierung gesteigert wird.

In jedem Intervall wird überprüft, ob Änderungen an der Versionsdatenbank eine Benachrichtigung erfordern. Dafür wird zu Beginn eines Intervalls überprüft, von welcher Software nicht die aktuellste, sondern eine veraltete Version installiert ist. Am Ende des Intervalls wird die Versionsdatenbank erneut eingelesen und die gleiche Abfrage zur Ermittlung veralteter Software erneut ausgeführt. Nun werden die beiden Ergebnisse der Abfragen verglichen. Falls bei der zweiten Abfrage eine veraltete Software hinzugekommen ist, wurde seit der ersten Abfrage eine neue Softwareversion verteilt. In diesem Fall wird dem Benutzer eine Benachrichtigung in der Taskleiste angezeigt, welche auf die neue Softwareversion hinweist. Abbildung 11 in Abschnitt 5.2.6 zeigt, wie eine derartige Benachrichtigung aussieht.

6.5 Feedback an die Entwickler

In Abschnitt 5.2.7 wird beschrieben, wie die Erstellung von Feedback über VersionInfo in den Entwicklungsprozess integriert wird. Möglich ist die Erstellung des Feedbacks durch eine Verbindung zwischen der in VersionInfo eingetragenen Software und den Phabricator-Projekten. Zu einem Eintrag in der Versionsdatenbank kann ein, als *Project-PHID* bezeichneter, Identifikator eines Phabricator-Projekts hinzugefügt werden. Sobald Benutzer neue Aufgaben erstellen, stellt Phabricator über diese Project-PHID eine Verbindung zu dem Projekt her. Die Erstellung neuer Aufgaben erfolgt, wie auch die Anreicherung der Änderungsprotokolle, über HTTP-Anfragen an die Conduit-API.

Zusätzlich zu einer Aufgabenbeschreibung in Textform können Benutzer Dateien an ihr Feedback anhängen. Das Hochladen dieser Dateien ist über weitere Anfragen an die Conduit-API umgesetzt, bei denen VersionInfo die Daten Base64-kodiert an Phabricator überträgt. Nachdem alle gewünschten Anhänge hochgeladen wurden, fügt VersionInfo einen Kommentar zu der erstellten Phabricator-Aufgabe hinzu, der eine Verknüpfung zu den Dateien enthält. Dadurch werden Bilder direkt in dem Kommentar zur Aufgabe angezeigt. Anhänge anderen Typs können von dort heruntergeladen werden. Bilder können beispielsweise Bildschirmfotos sein, die, wie in Abschnitt 5.2.7 beschrieben, zum Feedback hinzugefügt werden können.

In den Einträgen der Versionsdatenbank können, spezifisch für eine Feedbackart, Vorschläge für anzuhängende Dateien eingefügt werden. Wenn die Benutzer Feedback erstellen und die Oberfläche zum Anfügen von Dateien öffnen, werden ihnen diese angegebenen Dateien als mögliche Anhänge vorgeschlagen. Sollte es von den Benutzern gewünscht sein, hängt VersionInfo sie, wie zuvor beschrieben, an die Aufgaben an (vgl. Abschnitt 5.2.7).

Für Software, die bisher noch nicht mit Hilfe von Phabricator entwickelt wird, existiert eine weitere Möglichkeit zur Meldung von Feedback. Diese besteht in der Versendung von E-Mails mit den Informationen des Feedbackformulars. Zu diesem Zweck wird in der Versionsdatenbank eine E-Mail-Adresse der Entwickler angegeben, an die das Feedback versendet werden soll. Nach Eingabe der benötigten Daten wird die zur Absendung bereitstehende E-Mail im Standardmailprogramm des Benutzers geöffnet.

7 Entwicklung von Richtlinien zur Softwareentwicklung

Die meisten Maßnahmen, die im konzipierten Entwicklungsprozess enthalten sind, zeigen nur Wirkung, wenn die Entwickler sie akzeptieren und tatsächlich umsetzen. Kontinuierliche Integration hat beispielsweise keinen Nutzen, wenn die Entwickler nie oder nur selten ihren Quellcode in das Repository laden (vgl. Abschnitt 2.3). Code Reviews, die ohne Betrachtung des Codes akzeptiert werden, verfehlen ihre gewünschte Wirkung (vgl. Abschnitt 4.4). Feedback, welches nicht beachtet wird, bringt keine Verbesserungen der Software mit sich (vgl. Abschnitt 4.8). Um die Umsetzung des Entwicklungsprozesses zu unterstützen, sollen Richtlinien zum Prozess einen weiteren Beitrag zum Software-Qualitätskonzept dieser Bachelorarbeit liefern.

Richtlinien stellen Anweisungen für bestimmte Situationen und Tätigkeiten dar (vgl. [Dud15]). In dieser Arbeit sind sie dafür vorgesehen, die Einhaltung des Entwicklungsprozesses und die damit einhergehenden Qualitätssicherungsmaßnahmen zu garantieren. Außerdem sind grundlegende Anweisungen zur Softwareentwicklung in der Abteilung AE-SAS sowie technologiespezifische Vorgaben in den Richtlinien unterzubringen. Sie gehören damit zur konstruktiven Qualitätssicherung (vgl. [Hof13, S. 20 ff.]). Zur Unterstützung der Entwickler bei der Umsetzung des Qualitätskonzepts wurden folgende Inhalte definiert:

- Vorgaben zum Ablauf des Prozesses
- Vorgaben an die zu verwendende Software
- Anweisungen zur Erstellung der Dokumentation
- Programmierrichtlinien

Um die Vorgaben, die für die Umsetzung des konzipierten Entwicklungsprozesses relevant sind, an die Entwickler bereitzustellen, wurde im Laufe der Bachelorarbeit bereits ein Entwurf der Richtlinien [Deu15c] erstellt. Die Gliederung der Richtlinien erfolgt dabei nach den Schritten des Entwicklungsprozesses. Die in die Richtlinien zu integrierenden Anweisungen an die Entwickler sollen den jeweiligen Schritten zugeordnet werden. So soll erreicht werden, dass die Entwickler alle, für ihren aktuellen Arbeitsschritt relevanten, Informationen nah beieinander verfügbar haben. Der Entwurf der Richtlinien [Deu15c] kann dem beiliegenden, digitalen Anhang dieser Bachelorarbeit entnommen werden.

Die vollständige Ausarbeitung der Richtlinien zur Softwareentwicklung erfolgt im Nachgang dieser Bachelorarbeit, sobald der konzipierte Entwicklungsprozess für mehrere Projekte angewendet wird. Es sollen dabei alle Entwickler und weitere Beteiligte in die Erstellung einbezogen werden, um eine möglichst hohe Akzeptanz zu erreichen. Sobald eine vollständige Version der Richtlinien vorliegt, ist eine Einbindung der Richtlinien in das Qualitätsmanagement-Handbuch des Instituts AE vorgesehen.

Wichtig für die korrekte Anwendbarkeit der Richtlinien ist, dass sie immer auf dem aktuellen Stand sind. Aus diesem Grund sind die Richtlinien kontinuierlich an neue Begebenheiten anzupassen. So ist es notwendig, dass die Richtlinien angepasst werden, falls die Anwender der Richtlinien auf Lücken oder Fehler stoßen. Es muss außerdem ein Ziel sein, Unklarheiten in dem Dokument zu beseitigen, um die Akzeptanz der Entwickler für die Richtlinien zu bewahren. Für den Fall, dass am Entwicklungsprozess Veränderungen vorgenommen werden, sind diese ebenfalls in die Richtlinien zu übernehmen. Hiermit wird auf das Prinzip „Continuous Improvement“ der kontinuierlichen Auslieferung nach [HF11] eingegangen (vgl. Abschnitt 2.4).

8 Fazit und Ausblick

Die Erstellung eines Qualitätskonzepts für die Softwareentwicklungsprozesse der Abteilung AE-SAS konnte erfolgreich abgeschlossen werden. Ein verbesserter Prozess wurde konzipiert und umgesetzt, so dass er bereits bei der Entwicklung von Software innerhalb der Abteilung AE-SAS zum Einsatz kommt.

Der Entwicklungsprozess stellt einen wesentlichen Bestandteil des erstellten Qualitätskonzepts dar. Er beinhaltet Maßnahmen und Softwarewerkzeuge, die eine hohe Qualität der entwickelten Software garantieren sollen. Abbildung 17 stellt den auf die Meldung einer Aufgabe folgenden Ablauf dar, wie er mit dem Entwicklungsprozess dieser Bachelorarbeit verlaufen sollte.

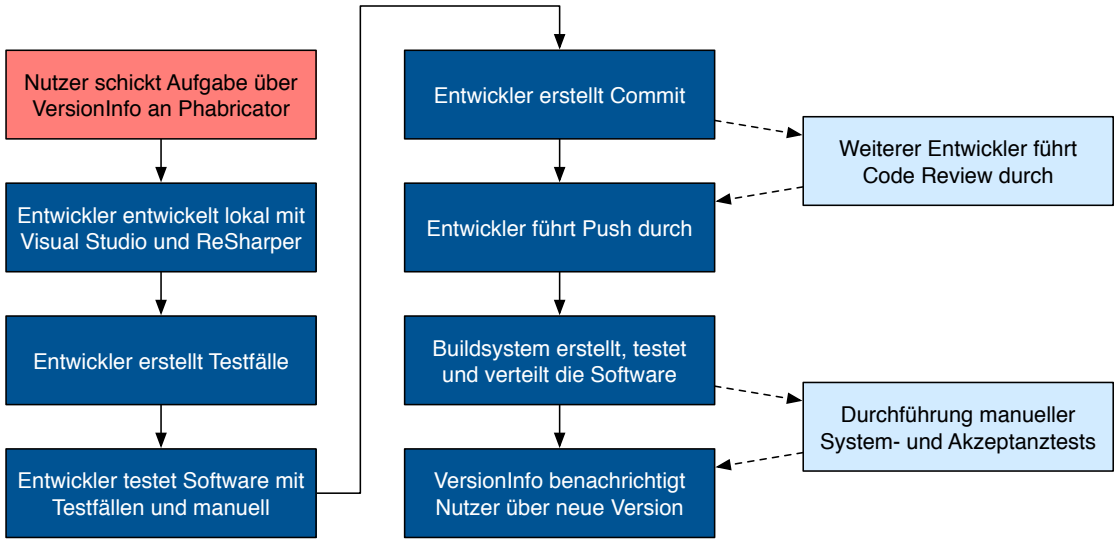


Abbildung 17: Typischer Ablauf der Softwareentwicklung unter Einsatz des entwickelten Konzepts.

Die Umsetzung des Entwicklungsprozesses verwendet zu einem großen Teil Software, die kostenfrei erhältlich ist. Nur die Entwicklungswerkzeuge Microsoft Visual Studio und ReSharper benötigen kostenpflichtige Lizenzen. Da jedoch nur Lizenzen für vier Entwickler benötigt werden, sind diese Kosten, verglichen mit der anvisierten Qualitätssteigerung, überschaubar.

Ergänzt wird der Prozess durch die Richtlinien zur Softwareentwicklung, die zur Einhaltung der vorgegebenen Abläufe dienen. Ein Entwurf dieser Richtlinien wurde im Laufe der Bachelorarbeit

erstellt und muss während des zukünftigen Einsatzes evaluiert und gegebenenfalls ergänzt werden. Außerdem müssen die Richtlinien in Zukunft angepasst werden, falls Änderungen am Entwicklungsprozess stattfinden. Nach Abschluss dieser Bachelorarbeit ist es vorgesehen, die Richtlinien in das Qualitätsmanagement-Handbuch des Instituts AE zu übernehmen. Dadurch kann bei zukünftigen Zertifizierungen des Qualitätsmanagements auch die Softwareentwicklung miteinbezogen werden.

Eine weitere Maßnahme im Nachgang dieser Bachelorarbeit ist es, den Entwicklungsprozess auf sämtliche Softwareentwicklungen der Abteilung AE-SAS anzuwenden. Zusammen mit den Richtlinien wird damit das aufgestellte Qualitätskonzept, mit allen Beteiligten gemeinsam, umgesetzt. Außerdem kann überlegt werden, ob das Buildsystem in Zukunft auf ein dediziertes System verlagert wird. Dieses sollte für den Dauerbetrieb ausgelegt sein, wodurch eine ständige Verfügbarkeit sichergestellt wird.

Literaturverzeichnis

[ABL89] Ackerman, A. F. ; Buchwald, L. S. ; Lewski, F. H.: Software inspections: An effective verification process. In: *IEEE Software* 6 (1989), Nr. 3, S. 31–36. http://dx.doi.org/10.1109/52.28121. – DOI 10.1109/52.28121. – ISSN 0740–7459

[AG15] Android ; Google ; Statista (Hrsg.): *Anzahl der verfügbaren Apps im Google Play Store von Dezember 2009 bis Februar 2015*. http://de.statista.com/statistik/daten/studie/74368/umfrage/anzahl-der-verfuegbaren-apps-im-google-play-store/. Version: 2015. – Zugriff: 24.08.2015

[AS14] Aichele, Christian ; Schönberger, Marius: *IT-Projektmanagement: Effiziente Einführung in das Management von Projekten*. Wiesbaden : Springer Fachmedien Wiesbaden, 2014 (essentials). – ISBN 978–3–658–08388–5

[BA04] Beck, Kent ; Andres, Cynthia: *Extreme programming explained: Embrace change*. 2nd ed. Boston, MA : Addison-Wesley, 2004. – ISBN 9780321278654

[Bal09] Balzert, Helmut: *Lehrbuch der Software-Technik: Basiskonzepte und Requirements Engineering*. 3. Aufl. Heidelberg [u.a.] : Spektrum, Akad. Verl., 2009 (Lehrbücher der Informatik). – ISBN 9783827417053

[Bal11] Balzert, Helmut: *Lehrbücher der Informatik*. Bd. [2]: *Lehrbuch der Software-Technik: Entwurf, Implementierung, Installation und Betrieb*. 3. Aufl. Heidelberg [u.a.] : Spektrum, Akad. Verl., 2011. – ISBN 9783827417060

[BBv+01] Beck, Kent ; Beedle, Mike ; van Bennekum, Arie ; Cockburn, Alistair ; Cunningham, Ward ; Fowler, Martin ; Grenning, James ; Highsmith, Jim ; Hunt, Andrew ; Jeffries, Ron ; Kern, Jon ; Marick, Brian ; Martin, Robert C. ; Mellor, Steve ; Schwaber, Ken ; Sutherland, Jeff ; Thomas, Dave: *Manifesto for Agile Software Development*. http://agilemanifesto.org/. Version: 2001. – Zugriff: 20.08.2015

[Bec99] Beck, Kent: *Extreme programming eXplained: Embrace change*. Reading, MA : Addison-Wesley, 1999 (XP series). – ISBN 9780201616415

[BK13] Broy, Manfred ; Kuhrmann, Marco: *Projektorganisation und Management im Software Engineering*. Berlin, Heidelberg : Imprint: Springer Vieweg, 2013

(Xpert.press). – ISBN 978–3–642–29289–7

[Bla15] Black Duck Software, Inc: *Statistik zur Nutzungshäufigkeit von Versionsverwaltungssystemen unter Open Source-Projekten, die bei OpenHUB registriert sind*. https://www.openhub.net/repositories/compare. Version: 2015. – Zugriff: 18.08.2015

[Bun08] Bundesamt für Sicherheit in der Informationstechnik: *IT-Grundschutz Vorgehensweise*. https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzStandards/ITGrundschutzStandards_node.html. Version: 2.0, 2008. – Zugriff: 10.08.2015

[BW05] Bennett, Ted L. ; Wennberg, Paul W.: Eliminating Embedded Software Defects Prior to Integration Test. In: U.S. Air Force Software Technology Support Center (Hrsg.): *CrossTalk: The Journal of Defense Software Engineering. Volume 18, Number 12, December 2005* Bd. 12. Ft. Belvoir : Defense Technical Information Center, 2005 (18), S. 215

[Cla01] Clauss, Stephan ; Handelsblatt GmbH (Hrsg.): *Software-Fehler kosten Geld*. http://www.handelsblatt.com/archiv/fruehzeitiges-testen-wird-nur-unzureichend-genutzt-software-fehler-kosten-geld/2103754.html. Version: 2001. – Zugriff: 24.08.2015

[Coh10] Cohn, Mike: *Succeeding with agile: Software development using Scrum*. Upper Saddle River, NJ : Addison-Wesley, 2010 (The Addison-Wesley signature series). – ISBN 0321579364

[CS14] Chacon, Scott ; Straub, Ben: *Pro Git, Second Edition*. Springer Science and Business Media, 2014. – ISBN 1484200772

[Deu08a] Deutsches Zentrum für Luft- und Raumfahrt e. V.: *Qualitätsmanagement-Handbuch des QM-Rahmensystems Teil 1*. 22.01.2008

[Deu08b] Deutsches Zentrum für Luft- und Raumfahrt e. V.: *Rahmenrichtlinie Software-Engineering: Verfahrensanweisungen - QMH-DLR-04-V03*. 22.01.2008

[Deu15a] Deutsches Zentrum für Luft- und Raumfahrt e. V.: *DLR Software-Katalog: Interne Webanwendung zur Erfassung von im DLR entwickelten Softwareprogrammen*. 2015. – Zugriff: 03.09.2015

[Deu15b] Deutsches Zentrum für Luft- und Raumfahrt e. V.: *DLR-Software-Standards, generiert im DLR Software-Katalog für die Software der Abteilung*

AE-SAS. 2015

[Deu15c] Deutsches Zentrum für Luft- und Raumfahrt e. V.: *Richtlinien für die Softwareentwicklung: Bestandteil des Software-Qualitätskonzepts der Abteilung AE-SAS*. 2015

[DHJ14] Domnick, André ; Hamdy, Safuat ; Jendrian, Kai: Gut erprobt: Wie man die Sicherheit von Anwendungen systematisch überprüft. In: *iX - für Professionelle Informationstechnik 12/2014* 2014 (2014), Nr. 12, S. 64–71

[DIN01] DIN Deutsches Institut für Normung e.V.: *Software-Engineering - Qualität von Software-Produkten - Teil 1: Qualitätsmodell (ISO/IEC 9126:2001)*. Berlin, 06.2001

[DIN08] DIN Deutsches Institut für Normung e.V.: *Qualitätsmanagementsysteme – Anforderungen (ISO 9001:2008)*. Berlin, 12.2008

[DIN11] DIN Deutsches Institut für Normung e.V.: *Informationstechnik – IT-Sicherheitsverfahren – Informationssicherheits-Managementsysteme – Überblick und Terminologie (ISO/IEC 27000:2009)*. Berlin, 07.2011

[DIN14] DIN Deutsches Institut für Normung e.V.: *Qualitätsmanagementsysteme – Grundlagen und Begriffe (ISO/DIS 9000:2014)*. Berlin, 08.2014

[DIN15] DIN Deutsches Institut für Normung e.V.: *Webpräsenz des Deutschen Instituts für Normung e.V. - DIN - kurz erklärt*. http://www.din.de/de/ueber-normen-und-standards/basiswissen. Version: 2015. – Zugriff: 03.09.2015

[Dud15] Duden ; Bibliographisches Institut GmbH (Hrsg.): *Wörterbucheintrag zum Wort "Richtlinie"*. http://www.duden.de/rechtschreibung/Richtlinie. Version: 2015. – Zugriff: 06.09.2015

[EEG14] Eckkrammer, Tobias ; Eckkrammer, Florian ; Gollner, Helmut: Agiles IT-Projektmanagement im Überblick. In: Bauer, Nikolai (Hrsg.) ; Tiemeyer, Ernst (Hrsg.): *Handbuch IT-Projektmanagement*. München : Hanser, Carl, 2014. – ISBN 3446441212, S. 75–118

[Epp11] Epping, Thomas: *Kanban für die Softwareentwicklung*. Berlin, Heidelberg : Springer-Verlag Berlin Heidelberg, 2011 (Informatik im Fokus). – ISBN 3642225942

[Eri15] Eric Woodruff: *Webpräzens des Sandcastle Help File Builder auf Github*. https://github.com/EWSoftware/SHFB. Version: 2015. – Zugriff: 12.08.2015

[Fag76] Fagan, M. E.: Design and code inspections to reduce errors in program development. In: *IBM Systems Journal* 15 (1976), Nr. 3, S. 182–211. http://dx.doi.org/10.1147/sj.153.0182. – DOI 10.1147/sj.153.0182. – ISSN 0018–8670

[Fre91] Free Software Foundation, Inc.: *GNU General Public License Version 2*. http://www.gnu.org/licenses/gpl-2.0.html. Version: 1991. – Zugriff: 21.07.2015

[Fri15] Friese, Ulrich ; Frankfurter Allgemeine Zeitung GmbH (Hrsg.): *A400M stürzte wohl wegen Software-Problemen ab*. http://www.faz.net/aktuell/wirtschaft/militaer-airbus-a400m-stuerzte-wegen-software-problemen-ab-13600602.html. Version: 2015. – Zugriff: 24.08.2015

[Gas14] Gasior, Łukasz: *ReSharper Essentials: Make your Microsoft Visual studio work smarter with ReSharper*. Birmingham, UK : Packt Pub, 2014 (Community experience distilled). – ISBN 9781849698702

[GL12] Guckenheimer, Sam ; Loje, Neno: *Visual Studio Team Foundation Server 2012: Adopting agile software practices : from backlog to continuous feedback*. 3rd ed. Upper Saddle River, NJ : Addison-Wesley, 2012. – ISBN 9780321864871

[Gra15] Graf, Christian A.: *Skript zur Vorlesung Software-Qualitätssicherung an der DHBW Mannheim*. 09.06.2015

[Har15] Harry, Brian ; Microsoft Corporation (Hrsg.): *Ankündigung zur Verfügbarkeit von Git in Visual Studio*. http://blogs.msdn.com/b/bharry/archive/2013/01/30/git-init-vs.aspx. Version: 2015. – Zugriff: 24.08.2015

[HF11] Humble, Jez ; Farley, David: *Continuous delivery*. Upper Saddle River, NJ : Addison-Wesley, 2011 (The Addison-Wesley signature series). – ISBN 0321601912

[Hof13] Hoffmann, Dirk W.: *Software-Qualität*. 2., aktualisierte und korrigierte Aufl. Berlin : Springer Vieweg, 2013 (eXamen.press). – ISBN 978–3–642–35699–5

[IDA15] IDATE ; Statista (Hrsg.): *Umsatz im Markt für Software und IT-Services weltweit von 2005 bis 2018*. http://de.statista.com/statistik/daten/studie/159325/umfrage/weltweiter-umsatz-mit-software-und-it-services-seit-2005/. Version: 2015. – Zugriff: 24.08.2015

[IfD14] IfD Allensbach ; Statista (Hrsg.): *Anzahl der Computernutzer (private und/oder berufliche Nutzung) in Deutschland von 2013 bis 2014*. http://de.statista.com/statistik/daten/studie/168951/umfrage/anzahl-der-computernutzer-in-deutschland/. Version: 2014. – Zugriff: 24.08.2015

[Int11] International Software Testing Qualifications Board: *ISTQB Certified Tester Foundation Level Syllabus*. 2011

[Jen15a] Jenkins CI: *MSTest Plugin im Jenkins Plugin-Verzeichnis*. https://wiki.jenkins-ci.org/display/JENKINS/MSTest+Plugin. Version: 2015. – Zugriff: 23.07.2015

[Jen15b] Jenkins CI: *Webpräsenz von Jenkins CI*. http://jenkins-ci.org/. Version: 2015. – Zugriff: 16.07.2015

[Jet15] JetBrains s.r.o.: *Webpräsenz von ReSharper*. https://www.jetbrains.com/resharper/. Version: 2015. – Zugriff: 18.08.2015

[JK15a] Jenkins CI ; Kohsuke Kawaguchi: *Mercurial Plugin im Jenkins Plugin-Verzeichnis*. https://wiki.jenkins-ci.org/display/JENKINS/Mercurial+Plugin. Version: 2015. – Zugriff: 22.07.2015

[JK15b] Jenkins CI ; Kyle Sweeney: *MSBuild Plugin im Jenkins Plugin-Verzeichnis*. https://wiki.jenkins-ci.org/display/JENKINS/MSBuild+Plugin. Version: 2015. – Zugriff: 22.07.2015

[JY15] Jenkins CI ; Yasuyuki Saito: *VsTestRunner Plugin im Jenkins Plugin-Verzeichnis*. https://wiki.jenkins-ci.org/display/JENKINS/VsTestRunner+Plugin. Version: 2015. – Zugriff: 22.07.2015

[Kan03] Kan, Stephen H.: *Metrics and models in software quality engineering*. 2nd ed. Boston : Addison-Wesley, 2003. – ISBN 9780201729153

[Klo12] Kloss, Michael ; Heise Online (Hrsg.): *Die Rollen von Scrum in einer Person vereinen*. http://heise.de/-1748170. Version: 2012. – Zugriff: 05.09.2015

[KP09] Kemerer, C. F. ; Paulk, M. C.: The Impact of Design and Code Reviews on Software Quality: An Empirical Study Based on PSP Data. In: *IEEE Transactions on Software Engineering* 35 (2009), Nr. 4, S. 534–550. http://dx.doi.org/10.1109/TSE.2009.27. – DOI 10.1109/TSE.2009.27. – ISSN 0098–5589

[KRSW08] Kersten, Heinrich ; Reuter, Jürgen ; Schröder, Klaus-Werner ; Wolfenstetter, Klaus-Dieter: *IT-Sicherheitsmanagement nach ISO 27001 und Grundschutz: Der Weg zur Zertifizierung*. 1. Aufl. Wiesbaden : Friedr. Vieweg & Sohn Verlag, 2008 (Edition Kes). – ISBN 978–3–8348–0178–4

[KWV14] Kraus, Georg ; Westermann, Reinhold ; Vetter, Ulrike M.: *Projektmanagement mit system: Organisation, methoden, steuerung*. 5., Auflage. Wiesbaden, Germany : Springer Gabler, 2014. – ISBN 3834945900

[Lis13] Lischner, Ray: *Exploring C++ 11*. 2. ed. Berkeley, CA : Apress, 2013 (Expert's voice in C++). – ISBN 1430261943

[Lor15] LoreSoft: *Webpräsenz des MSBuild Community Tasks Projekts auf GitHub*. https://github.com/loresoft/msbuildtasks. Version: 2015. – Zugriff: 17.08.2015

[Mac15] Mackall, Matt: *Webpräsenz von Mercurial*. https://mercurial.selenic.com/. Version: 2015. – Zugriff: 16.07.2015

[Mar09] Martin, Robert C.: *Clean code: A handbook of agile software craftsmanship*. Upper Saddle River, NJ : Prentice Hall, 2009 (Robert C. Martin series). – ISBN 9780132350884

[Mas88] Massachusetts Institute of Technology: *MIT Lizenz*. http://opensource.org/licenses/MIT. Version: 1988

[MB14] Marquart, Maria ; Braun, Katja: *Umfrage zur Lebenszufriedenheit: Das stille Glück der Generation Mitte*. http://www.spiegel.de/wirtschaft/soziales/umfrage-zu-digitalisierung-und-demografie-angst-der-generation-mitte-a-1000811.html. Version: 2014. – Zugriff: 24.08.2015

[McC01] McCann, Robert T.: How Much Code Inspection Is Enough? In: U.S. Air Force Software Technology Support Center (Hrsg.): *CrossTalk: The Journal of Defense Software Engineering. Volume 14, Number 7, July 2001* Bd. 7. Ft. Belvoir : Defense Technical Information Center, 2001 (14), S. 9–12

[Mic07a] Microsoft Corporation: *Microsoft Public License (Ms-PL)*. http://opensource.org/licenses/ms-pl. Version: 2007. – Zugriff: 12.08.2015

[Mic07b] Microsoft Corporation: *Microsoft Reciprocal License (Ms-RL)*. http://opensource.org/licenses/ms-rl. Version: 2007. – Zugriff: 23.07.2015

[Mic15a] Microsoft Corporation: *Beschreibung der XML Kommentare zur Dokumentation von Quellcode*. https://msdn.microsoft.com/en-us/library/b2s063f7.aspx. Version: 2015. – Zugriff: 17.08.2015

[Mic15b] Microsoft Corporation: *Beschreibung zu Windows Installer*. https://msdn.microsoft.com/en-us/library/cc185688.aspx. Version: 2015. – Zugriff: 07.08.2015

[Mic15c] Microsoft Corporation: *Dokumentation zu MSTest*. https://msdn.microsoft.com/en-us/library/jj155804.aspx. Version: 2015. – Zugriff: 17.08.2015

[Mic15d] Microsoft Corporation: *Dokumentation zu VSTest.console*. https://msdn.microsoft.com/en-us/library/jj155800.aspx. Version: 2015. – Zugriff: 17.08.2015

[Mic15e] Microsoft Corporation: *Einführung zu WPF im Microsoft Developer Network*. https://msdn.microsoft.com/en-us/library/ms742119(v=vs.100).aspx. Version: 2015. – Zugriff: 01.09.2015

[Mic15f] Microsoft Corporation: *MSBuild Dokumentation*. https://msdn.microsoft.com/en-us/library/dd393574.aspx. Version: 2015. – Zugriff: 28.07.2015

[Mic15g] Microsoft Corporation: *Vergleich der Varianten von Visual Studio 2015*. https://www.visualstudio.com/products/compare-visual-studio-2015-products-vs. Version: 2015. – Zugriff: 19.07.2015

[Mic15h] Microsoft Corporation: *Webpräsenz von Microsoft Access*. http://office.microsoft.com/access. Version: 2015. – Zugriff: 08.09.2015

[Mic15i] Microsoft Corporation: *Webpräsenz von MSBuild auf GitHub*. https://github.com/microsoft/msbuild. Version: 2015. – Zugriff: 20.07.2015

[Mic15j] Microsoft Corporation: *Webpräsenz von Visual Studio*. https://www.visualstudio.com/. Version: 2015. – Zugriff: 16.07.2015

[Mic15k] Microsoft Corporation: *Windows Forms im Microsoft Developer Network*. https://msdn.microsoft.com/en-us/library/dd30h2yb(v=vs.100).aspx. Version: 2015. – Zugriff: 01.09.2015

[.NE15] .NET Foundation: *Webpräsenz von NuGet*. https://www.nuget.org/. Version: 2015. – Zugriff: 21.07.2015

[OR15] Outercurve Foundation ; Rob Mensching: *Webpräsenz des WiX Toolset*. http://wixtoolset.org/. Version: 2015. – Zugriff: 22.07.2015

[O'S09] O'Sullivan, Bryan: *Mercurial: The definitive guide*. 1st ed. Sebastopol, Calif. : O'Reilly Media, 2009. – ISBN 0596551290

[Pha15a] Phacility, Inc.: *Dokumentation zu Arcanist*. https://secure.phabricator.com/book/phabricator/article/arcanist/. Version: 2015. – Zugriff: 20.08.2015

[Pha15b] Phacility, Inc.: *Vergleich zwischen Reviews und Audits in Phabricator*. https://secure.phabricator.com/book/phabricator/article/reviews_vs_audit/. Version: 2015. – Zugriff: 03.08.2015

[Pha15c] Phacility, Inc.: *Webpräsenz von Phabricator*. http://phabricator.org/. Version: 2015. – Zugriff: 16.07.2015

[Pha15d] Phacility, Inc.: *Webpräsenz von Phacility*. http://phacility.com/. Version: 2015. – Zugriff: 21.07.2015

[PJR09] Page, Alan ; Johnston, Ken ; Rollison, Bj: *How we test software at Microsoft*. Redmond, Wash. : Microsoft, 2009 (Best practices). – ISBN 0735624259

[PP03] Poppendieck, Mary ; Poppendieck, Thomas D.: *Lean software development: An agile toolkit*. Boston, Mass. : Addison-Wesley, 2003 (The agile software development series). – ISBN 9780321150783

[Pri14] Priestley, Evan ; Phacility, Inc. (Hrsg.): *Auflistung der von Phabricator interpretierten Schlüsselwörter in dem Kollaborationstool von Phacility, Inc.* https://secure.phabricator.com/T5132. Version: 2014. – Zugriff: 27.08.2015

[Ram12] Ramirez, Nick: *WiX 3.6: A Developer's Guide to Windows Installer XML*. Birmingham : Packt Pub, 2012. – ISBN 1782160434

[Sco15] Scott Chacon: *Webpräsenz von Git*. https://git-scm.com/. Version: 2015. – Zugriff: 18.08.2015

[Ser15] Sergey Antonov: *Webpräsenz von HgSccPackage auf Bitbucket*. https://bitbucket.org/zzsergant/hgsccpackage. Version: 2015. – Zugriff: 21.07.2015

[SG15] Stellman, Andrew ; Greene, Jennifer: *Learning agile: Understanding Scrum, XP, Lean, and Kanban*. First edition. Beijing : O'Reilly, 2015. – ISBN 9781449363826

[Sma11] Smart, John F.: *Jenkins: The definitive guide*. 1st ed. Sebastopol, CA : O'Reilly Media, 2011. – ISBN 144931306X

[Smi09] Smith, Josh ; Microsoft Corporation (Hrsg.): *Patterns - WPF Apps With The Model-View-ViewModel Design Pattern*. https://msdn.microsoft.com/en-us/magazine/dd419663.aspx. Version: 2009. – Zugriff: 01.09.2015

[Sof15] Software in the Public Interest, Inc: *Statistik zur Anzahl an Installationen von Git und Mercurial auf Debian-Systemen*. https://qa.debian.org/popcon-graph.php?packages=git+mercurial. Version: 2015. – Zugriff: 18.08.2015

[The04] The Apache Software Foundation: *Apache License Version 2.0*. http://www.apache.org/licenses/LICENSE-2.0.html. Version: 2004. – Zugriff: 21.07.2015

[Van13] Vance, Stephen: *Quality code: Software testing principles, practices, and patterns*. Upper Saddle River, NJ : Addison-Wesley, 2013. – ISBN 9780321832986

[van15] van Heesch, Dimitri: *Webpräsenz von Doxygen*. http://www.stack.nl/~dimitri/doxygen/. Version: 2015. – Zugriff: 17.08.2015

[Ver06] Verein Deutscher Ingenieure: *Technische Dokumentation - Begriffsdefinitionen und rechtliche Grundlagen (VDI 4500 Blatt 1)*. Juni 2006. Berlin, 06.2006

[Wol15] Wolff, Eberhard ; Heise Online (Hrsg.): *Continuous Integration widerspricht Feature Branches!* http://heise.de/-2736487. Version: 2015. – Zugriff: 10.09.2015

Anhang

A Werte der agilen Softwareentwicklung aus dem agilen Manifest

Aus [BBv+01], Erklärungen nach [EEG14, S. 75]

Individuals and interactions over processes and tools

Kommunikation zwischen Individuen ist wichtiger als der Einsatz von Prozessen und Werkzeugen.

Working software over comprehensive documentation

Eine funktionsfähige Software ist wichtiger als eine umfangreiche Dokumentation.

Customer collaboration over contract negotiation

Zusammenarbeit mit dem Auftraggebern ist wichtiger als die Aushandlung von Verträgen.

Responding to change over following a plan

Auf Veränderungen zu reagieren ist wichtiger als die Einhaltung eines Plans.

B Werte der schlanken Softwareentwicklung

Aus [PP03], Erklärungen nach [SG15, Kap. 8.1] und [Epp11, S. 40 ff.]

Eliminate Waste

Entferne Ballast aus dem Projekt, der Fortschritt behindert.

Amplify Learning

Lerne aus Fehlern und Rückmeldungen.

Decide as Late as Possible

Treffe Entscheidungen so spät wie möglich, damit viele Informationen gesammelt werden können.

Deliver as Fast as Possible

Arbeite schnell, um zuverlässiges Feedback und verzögerte Entscheidungen möglich zu machen.

Empower the Team

Bilde ein Team aus aktiven Personen, anstatt einzelnen Personen die Steuerung zu überlassen.

Build Integrity In

Erlange die Akzeptanz der Anwender, indem die Software in sich stimmig und intuitiv benutzbar ist.

See the Whole

Lege den Fokus auf die Gesamtheit der Software, der sich die einzelnen Teile unterordnen.

C Ein- und Ausgabeformate von VersionInfo

C.1 Beispiel einer XML-Konfigurationsdatei für den Softwareverteilungsvorgang mit dem VersionInfoManager

```
<?xml version="1.0" encoding="utf-8"?>
<VersionInfoDeploymentConfig xmlns="http://dlr.de/VersionInfoDeploymentConfig.xsd">
  <ApplicationName>VersionInfo</ApplicationName>
  <DeployAsLatestVersion>false</DeployAsLatestVersion>
  <ExecutableDirectory>dotNET\VersionInfoWPF\bin\Release</ExecutableDirectory>
  <RelativeExectuableFile>VersionInfo.exe</RelativeExectuableFile>

  <SetupDirectory>dotNET\VersionInfoWiXSetup\bin\Release</SetupDirectory>
  <RelativeSetupFile>VersionInfo.msi</RelativeSetupFile>

  <DestinationXMLFile>\\boa.ae.go.dlr.de\Programme\VersionInfo\versions.xml</
      DestinationXMLFile>
  <DestinationVIDirectory>\\boa.ae.go.dlr.de</DestinationVIDirectory>
  <DestinationSubDirectory>%VIREMOTEDIR%\Programme\VersionInfo\Archiv</
      DestinationSubDirectory>

  <ReleaseNotesFile>%SetupDirectory%\ReleaseNotes.txt</ReleaseNotesFile>
  <RTFReleaseNotesTemplate>dotNET\ReleaseNotesTemplate.rtf</RTFReleaseNotesTemplate>
  <AMLReleaseNotesTemplate>dotNET\ReleaseNotesTemplate.aml</AMLReleaseNotesTemplate>
  <RTFReleaseNotesFileFormatted>%SetupDirectory%\ReleaseNotes.rtf</
      RTFReleaseNotesFileFormatted>
  <RTFReleaseNotesFileTemplated>%SetupDirectory%\ReleaseNotes_Templated.rtf</
      RTFReleaseNotesFileTemplated>
  <AMLReleaseNotesFileTemplated>dotNET\VersionInfoDocumentation\Content\VersionHistory\
      ReleaseNotes.aml</AMLReleaseNotesFileTemplated>

  <PhabricatorHost>http://phabricator.ae.go.dlr.de</PhabricatorHost>
  <PhabricatorAuthentificationToken>api-grxutlyv2gl4rd6blefl6fz3dlyc</
      PhabricatorAuthentificationToken>
</VersionInfoDeploymentConfig>
```

C.2 Ausschnitt aus einer XML-Versionsdatenbank von VersionInfo

```
<?xml version="1.0" encoding="utf-8"?>
<Versions>
  <Application name="VersionInfo" quietUpdate="True" company="Deutsches Zentrum fuer
      Luft- und Raumfahrt e. V. (DLR)" path="C:\AE-SAS\VersionInfo\VersionInfo.exe"
      iconpath="%VIREMOTEDIR%\Programme\VersionInfo\Resources\Icons\DLR-Icon_000-schwarz.
      ico" groupName="VersionInfo" latestVersion="2.0.1.103">
    <Release name="2.0.1.103" releaseDate="01-09-2015" hash="6
        e9c84702ce69ab22260437a68d4975a" setupFile="%VIREMOTEDIR%\Programme\VersionInfo\
        Archiv\VersionInfo\2.0.1.103\VersionInfo.msi" wholeDirectory="True">
      <ReleaseNotes>
        <RTFReleaseNote path="%VIREMOTEDIR%\Programme\VersionInfo\Archiv\VersionInfo
            \2.0.1.103\ReleaseNotes.rtf" />
      </ReleaseNotes>
    </Release>
    <ContactMethods>
      <PhabricatorBugReportingMethod phid="PHID-PROJ-el72u7z6ao4awjuhypix">
        <Attachment alias="Version Information File" path="C:\AE-SAS\VersionInfo\versions.
            xml" />
        <Attachment alias="Version Information Groups File" path="C:\AE-SAS\VersionInfo\
            groups.xml" />
      </PhabricatorBugReportingMethod>
      <PhabricatorFeatureRequestingMethod phid="PHID-PROJ-el72u7z6ao4awjuhypix" />
    </ContactMethods>
  </Application>
  <File name="Microphones" quietUpdate="False" company="Deutsches Zentrum fuer Luft- und
      Raumfahrt e. V. (DLR)" path="C:\AE-SAS\Hardware\Datenbank\DLR_Microphones.xls"
      groupName="Transducers" latestVersion="2015.01.08">
    <Release name="2015.01.08" hash="7e870f02a1c2dbdaf1e99560281bbc54" setupFile="%
        VIREMOTEDIR%\SMT\SMT-Hardware\Sensoren+Geraete\Mikrofone\Sensorwerte\2015.01.08\
        DLR_Microphones.xls" wholeDirectory="True" />
  </File>
</Versions>
```

C.3 Mit dem VersionInfoManager angereicherte und formatierte Änderungsprotokolle im Corporate Design

Release Notes
VersionInfo 2.0.1.103

2015-09-01 - Revision 128:
Minor comment fix

2015-09-01 - Revision 127:
Fixes T181
Task 181: Release note formatting fails
The formatting process of the release notes fails, therefore no release notes are included in the setup.

2015-08-30 - Revision 126:
Test: Deploy as latest version

2015-08-30 - Revision 125:
Fixes T180
Task 180: [FEATURE] Allow to automatically set a new release as latest version
When deploying via the VersionInfoManager, the deployed release will only be added to the application's VersionItem. The deployment config file should contain a flag which tells VersionInfoManager to also set the release as the latest version on deployment
Reported for version 2.0.1.94 of VersionInfo

2015-08-30 - Revision 124:
Fixes T180
Task 180: [FEATURE] Allow to automatically set a new release as latest version
When deploying via the VersionInfoManager, the deployed release will only be added to the application's VersionItem. The deployment config file should contain a flag which tells VersionInfoManager to also set the release as the latest version on deployment
Reported for version 2.0.1.94 of VersionInfo

2015-08-28 - Revision 123:
Removed whitespace

2015-08-28 - Revision 122:
Added whitespace

2015-08-28 - Revision 121:
Another test of CodedUiTests

2015-08-19 - Revision 120:
Fixes T165 now, as the last commit has not done so
Task 165: [BUG] An invalid release notes file is written to the version.xml file during deployment via VersionInfoManager

2015-08-19 - Revision 119:
Fixes T165
Task 165: [BUG] An invalid release notes file is written to the version.xml file during deployment via VersionInfoManager
Minor documentation changes

2015-08-18 - Revision 118:
Fixes T162
Task 162: Link to specific topic / chapter in the .chm documentation file

Deutsches Zentrum
für Luft- und Raumfahrt
German Aerospace Center

Institut für Aeroelastik
Bunsenstr. 10
37073 Göttingen