ulm university universität
uulm

Universität Ulm

Fakultät für Mathematik und Wirtschaftswissenschaften

Institut für angewandte Informationsverarbeitung

Begleit-Seminar zur Vorlesung

Entwicklung und Betrieb von Informationssystemen

Sommersemester 2014

Titel der Ausarbeitung:

Dokumenten- /Konfigurationsmanagement in verteilten Software-Projekten

Sara Steisslinger	Matrikelnummer: 823825	Studienrichtung: WiWi BSc
Jasmin Klose	Matrikelnummer: 757415	Studienrichtung: WiWi BSc
Manuel Buchert	Matrikelnummer: 758373	Studienrichtung: WiWi BSc
Florian Rotter	Matrikelnummer: 761110	Studienrichtung: WiWi BSc
Daniel Glunz	Matrikelnummer: 702033	Studienrichtung: WiWi MSc
Manuel Ott	Matrikelnummer: 770969	Studienrichtung: WiWi BSc

Ehrenwörtliche Erklärung

Wir versichern, dass wir die Seminararbeit gemeinsam verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und uns auch sonst keiner unerlaubten Hilfe bedient haben.

Ulm, den _____ _____

Unterschriften

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Mit immer fortschreitender Globalisierung entwickelt sich auch die Struktur der Unternehmen weiter. Viele Unternehmen entwickeln sich hin zu multinationalen Konzernen. Hierbei müssen sich alle Bereiche den neuen Anforderungen anpassen. Dies betrifft auch die Softwareentwicklung innerhalb der Unternehmen. Um eine konzernweit einsetzbare Software zu entwickeln, setzten Unternehmen immer mehr auf eine Kooperation von Entwicklungsteams. Der gesamte Software-Entwicklungsprozess wird zerlegt und von verschiedenen Mitarbeitern oder Teams vollzogen. Was früher noch in einer Abteilung geschah, Passiert heute um den ganzen Globus verteilt. So ist es bei vielen Unternehmen Realität, dass ein Teil der Software aus Deutschland, ein anderer aus China und der dritte Teil aus Indien stammt. Hierbei kommt es schnell zu neuen Problemen, die durch die Zergliederung auftreten. Kommunikation wird als wichtige Voraussetzung zur Verwaltung der Ressource „Wissen“ angesehen. „Softwareentwicklung ist von Natur aus teambasiert“ [GDHHIS07]. In verteilten Projekten kommt der Kommunikation eine noch größere Bedeutung zu. Es muss Wissensaustausch über Landes- oder sogar Kontinentgrenzen hinweg betrieben werden, was oft eine schwere Aufgabe darstellt. So hat z. B. eine Studie aufgezeigt, dass persönliche Treffen nicht durch Kommunikationsmedien ersetzt werden können. Trotz Videokonferenzen und Life-Chats, ist die Face-to-Face Kommunikation durch Reisen unabdingbar. Durch die verteilte Arbeit mehrerer Personen, rückt aber auch die Verwaltung der Versionen von Programmen und Dokumenten in den Vordergrund. Es müssen Regeln definiert werden, wer etwas ändern kann, wann dies geschehen darf oder in welcher Reihenfolge.

1.2 Gliederung

Diese Arbeit greift zuerst das Thema Software Configuration Management auf und behandelt seine einzelnen Aspekte. Zuerst wird die Konfigurationsidentifikation behandelt, anschließend folgt das Build Management, das Versionsmanagement, ebenso wie das Release und schließlich das Change Management.

2 Software Configuration Management (SCM)

Software Configuration Management bedeutet, die Evolution einer Software zu jeder Zeitpunkt des Produktlebenszyklus zu begleiten und zu kontrollieren. SCM als Softwareentwicklungsdisziplin, die die Werkzeuge und Techniken bzw. Methoden beinhaltet, die zum Erzeugen, Verwalten, Erweitern und Warten von Softwarebestandteilen benutzt werden, ist eine etwas formalere Beschreibung. Die Notwendigkeit der Kontrolle des Prozesses der Softwareentwicklung wird von den meisten Unternehmen erkannt, doch scheitert es meist an der optimalen Umsetzung. So findet das Thema SCM in der Literatur und auch der Forschung eine eher geringe Aufmerksamkeit. Die zu lösenden Probleme sind bei den meisten jungen Akademikern nicht präsent und finden deshalb auch auf dem Markt wenig Nachfrage. SCM zeichnet gute Softwareentwicklung aus. Es verbessert die Qualität und Ausfallsicherheit von Software durch Strukturen zur Identifizierung und Kontrolle von Dokumentation, Code, Interfaces, und Datenspeicher. Auch die Unterstützung von Methoden, angepasst an die Anforderungen, Standards, Organisationsstrukturen und Management-Philosophie ist wichtig. Ein anderer Punkt, der zur Verbesserung beiträgt, sind Statusmeldungen über Baseline, Change Control, Test, Release, sowie Auditing, die für Produktmanagement und zur Information über das Produkt wichtig sind.

2.1 Herkunft von SCM

Software ist sehr schnell und einfach zu zu erstellen und zu ändern. Aber es ist von großer Bedeutung, nachvollziehen zu können, welche Version was beinhaltet und welche Komponenten zusammen ein funktionsfähiges System ergeben. Dies wird besonders dann ersichtlich, wenn man das Konfigurationsmanagement (KM) statt dem SCM betrachtet. Das SCM ist von diesem, in den 70er Jahren, abgeleitet und auf Software übertragen worden. KM wird von der ISO folgendermaßen definiert: „Konfigurationsmanagement ist eine Managementtätigkeit, die die technische und administrative Leitung des gesamten Produktlebenszyklus, der Konfigurationseinheiten des Produktes und der produktkonfigurationsbezogenen Angaben übernimmt“ (ISO 10007: 2003 [Ham12]). Der amerikanische Industriestandard EIA-649 definiert KM etwas verständlicher: „Verfahren zur Herbei-

führung und ständiger Sicherstellung der Übereinstimmung der Leistungs-, Funktions- und physischen Charakteristiken eines Produkts mit den zugehörigen Anforderungen, den Ausführungen, den Ausführungsunterlagen und den für den Betrieb erforderlichen Informationen während des gesamten Lebenszyklus des Produkts.“ [Gbt]

2.2 Vergleich mit KM

Nimmt man, statt Software, die Hardware als Beispiel wird dies gleich viel klarer. So kann jeder leicht nachvollziehen, dass nicht jedes Teil mit jedem in einem Computer beliebig kombinierbar ist, sondern bestimmte Konfigurationen andere ausschließen. Man stelle sich Bestandteile eines Computers vor, z.B.: Mainboard, Prozessor, einen Speicher, Maus, Monitor, usw. Hat man jetzt einen Computer, ohne VGA-Anschluss, so kann man den Monitor, der nur einen solchen Anschluss besitzt, nicht anschließen und folglich auch nicht verwenden, da die beiden Systeme nicht kompatibel sind. Hieraus entsteht also kein funktionsfähiges System. So gibt es bei einem Computer viele einzelne Bestandteile, die in der passenden Konfiguration vorliegen müssen, um am Ende ein arbeitsfähiges System zu ergeben. Dieser Sachverhalt lässt sich auch auf Software übertragen. So muss auch zwischen den verschiedenen Teilen, passende Verbindungen, sog. Interfaces, geschaffen werden, sodass diese zusammenführbar werden. Auch das SCM beschäftigt sich mit denselben Problemen wie das KM der Hardware und noch vielen mehr. Ein Softwarebestandteil hat ein Interface. Diese Softwareteile werden auch Subsystem, Modul oder Komponente genannt und haben jeweils ein Interface. Viele Subsysteme zusammen und ergeben ein Softwaresystem. Jedes dieser Subsysteme muss eindeutig identifizierbar sein und eine Versionsnummer aufweisen. Schlussendlich benötigt man eine Stückliste über die Versionen aller Bestandteile, die das Gesamtsystem bilden. Allerding ist es viel schwerer, SCM „richtig“ anzuwenden , da hier die physikalischen Grenzen fehlen, die es beim KM gibt. Zudem ist Software viel einfacher und schneller änderbar. Schon ein paar Klicks und eine neue Version ist geschaffen. Im Gegensatz zur Hardwareproduktion, ist die Softwareentwicklung sehr viel schneller und kann an einem Tag von mehreren Teammitgliedern 100fach betrieben werden. So entsteht schnell eine riesige Flut an Versionen, die es zu verwalten gilt. Am Ende des Prozesses gilt es, alle Teile am selben Ort zur selben Zeit zusammenzuführen, sodass ein System entsteht, dass wie gefordert funktioniert. Die Bedeutung von SCM wird meist erkannt

und verstanden, doch scheitert es meist an der Umsetzung. SCM ist sehr komplex und es gibt keine genauen Richtlinien und Grundsätze für ein gutes SCM-System.

2.3 Aufgaben des SCM

Die Aufgaben des SCM sind vielseitig und bestehen aus der Verwaltung aller Komponenten eines Softwaresystems meist in einem Repository. Es soll keine redundanten Kopien oder Versionen geben und diese sicher gespeichert sein. Die Unterschiede zwischen den Versionen sollen hier sichtbar gemacht werden. Ddamit sollen Änderungen leichter ersichtlich gemacht werden, sodass diese diskutiert und dann verworfen oder angenommen werden können. Und es werden zusätzlich zu jeder Änderung Metadaten gespeichert. Diese beinhalten Daten über den Autor der Änderung, den Zeitpunkt, warum etwas geändert wurde und wo diese Änderung erfolgt. Diese Daten sind wichtig, um die Versionen verständlich und auch später noch nachvollziehbar zu machen. Es macht durch die persistente Speicherung auch den Zugriff auf Vorgängerversionen gut möglich. Und erlaubt auch die Definition von Referenzversionen, z. B. Releases, Milestones oder wichtige Zwischenstände wie bspw. fehlerfrei getestete Versionen.
Es gehören eine ganze Menge Aufgaben dazu, wie das Konfigurationsidentifikation, Build, Release und das Change Management, welche in den folgenden Kapiteln näher erläutert werden. [DK11]

2.4 Relevanz

Es ist sehr wichtig für ein gutes SCM die individuellen Anforderungen zu identifizieren und das Konzept an diese anzupassen. Die Zielvorstellung, sowie das erwähnte Konzept, dürfen nicht vernachlässigt werden und sollten immer präsent sein, um geeignete Werkzeuge unterstützend einsetzten zu können. Diese sollen vor allem die Effektivität und die Effizienz des gesamten Softwareentwicklungsprozess steigern und somit zu einer höheren Kundenzufriedenheit führen.

[HB09], [Edu04]

3 Konfigurationsidentifikation

3.1 Konfigurationsidentifikation / Requirements

Am Anfang eines jeden Software-Projektes sollte die zielgerichtete Festsetzung von Anforderungen (engl.: Requirements) stehen. Auch wenn dieser Aspekt viel zu häufig bedeutend geringer behandelt wird, als es eigentlich notwendig wäre, hat sich daraus inzwischen eine eigene wissenschaftliche Teildisziplin entwickelt. Betrachtet man große Projekte, die über einen langen Zeitraum entwickelt werden, ist eine Orientierung anhand der ursprünglichen Anforderungen von höchster Priorität, da sonst oftmals die Gefahr besteht, den eigentlichen Zweck der Software aus dem Fokus zu verlieren. Außerdem muss man passend auf Anpassungen in den Anforderungen reagieren können. Durch Arbeitsteilung und der daraus resultierenden Spezialisierung innerhalb von Projektgruppen, kann es schnell zu Verständnisproblemen und Kommunikationsschwierigkeiten kommen, die wiederum eine unpassende Problembehandlung nach sich ziehen können. Es ist somit zwingend erforderlich, gleich zu Anfang der Software-Entwicklung die entsprechenden Konfigurationsaspekte zu identifizieren und diese zwischen den beteiligten Personen zu synchronisieren, damit sich während der Arbeitsprozesse keine Eigenschaften aus einem Selbstzweck entwickeln, die den eigentlichen Vorgaben nicht dienlich sind.

3.2 Requirements Engineering / Requirements Management

Wird hinsichtlich der Anforderungen bereits sauber gearbeitet, verringern sich Fehlerursachen und ein mögliches Scheitern des Software-Projekts erheblich. Wichtig ist hierbei vor allem zu verstehen, dass zunächst nur die Identifizierung und Handhabe von reinen Anforderungen relevant ist und noch keinerlei Bezug auf eine mögliche Umsetzung genommen werden sollte. Somit zeichnen sich gute Anforderungen durch einige Qualitätskriterien aus, die als grundlegende Voraussetzungen für ihre Verwertbarkeit angesehen werden können.
Qualitätskriterien einzelner Anforderungen:

- Eindeutig
- Korrekt (besser: „adäquat“ oder „valide“)
- Klassifizierbar (bezüglich juristischer Verbindlichkeit)
- Konsistent (in sich und mit externen Vorgaben)
- Testbar (Erfüllung nachprüfbar)
- Aktuell gültig
- Verstehbar (für alle Stakeholder)
- Realisierbar
- Notwendig
- Bewertbar (hinsichtlich Wichtigkeit, Kritikalität oder Priorität)

[PP14]

3.3 Funktionale Anforderungen

Funktionale Anforderungen legen ausschließlich fest, was das System tun soll (funktional). Es wird jedoch nicht spezifiert, wie die Umsetzung zu erfolgen hat oder welche Eigenschaften damit verbunden sind. Typische funktionale Anforderungen sind:

- Eingaben und deren Einschränkungen (Daten, Ereignisse, Stimuli, ...)
- Bereitgestellte Dienste („Funktionen“), die das System ausführen können soll

  Umformung von Daten („funktionales Verhalten“)

  Verhaltensweisen, abhängig von Ereignissen/Stimuli („reaktives Verhalten“)
- Ausgabe (Daten, Fehlermeldungen, Reaktionen, ...)
- Manchmal auch

  Relevante Systemzustände („Betriebsmodi“)

  Zeitliches Verhalten des Systems

[PP14]

3.4 Nicht-funktionale Anforderungen

Anders als funktionale Anforderungen, beschreiben die nicht-funktionalen Anforderungen, wie gut ein System etwas tun soll (qualitativ). Abhängig des zu entwickelnden Systems können diese Anforderungen von unterschiedlichster Art sein. In diesem Zusammenhang werden über den ISO Standard 9126 hauptsächlich folgende sogenannte Qualitätsattribute für nicht-funktionale Anforderungen beschrieben, die für die meisten Projekte zutreffen:

- Performanz
- Funktionalität
- Usability
- Portabilität
- Sicherheit

[Fe07b]

3.5 Dokumentation (Anforderungsspezifikation)

Von entscheidender Relevanz für einen strukturierten und zielgerichteten Entwicklungsprozess ist eine angemessene Dokumentation der Anforderungen. Dadurch bleiben die wesentlichen Konfigurationsdetails im Blickfeld und es wird vermieden, dass sich Aspekte in die Systementwicklung einschleichen, die mit den ursprünglichen Anforderungen in keinem Zusammenhang stehen. Es hat sich als förderlich erwiesen, eine separate Dokumentation der Anforderungen von Kunden und Entwicklern umzusetzen, damit die ursprünglichen Konfigurationsdetails nicht durch die Konzepte der technischen Umsetzung verfälscht werden und alle beteiligten Parteien über Dokumente verfügen, die ihre fachlichen Konventionen und deren Rolle innerhalb des Projekts adäquat abbilden. Als Resultat entstehen verschiedene Anforderungsdokumente, die dann als Basis für die Entwicklung dienen und zumeist auch die Grundlage der Kommunikation zwischen Stakeholdern bilden. Die wichtigsten Anforderungsdokumente, das Lasten- und Pflichtenheft, werden in den folgenden Kapiteln nochmals detailliert beschrieben. Im weiteren Fortgang des Projekts werden die Anforderungen aus diesen Dokumenten dann oft über spezielle Anforderungsmanagement-Software verwaltet und umgesetzt. Dadurch lassen

sich Interdependenzen zwischen verschiedenen Teilsystemen einfacher abbilden und zurückverfolgen und man hat eine einheitliche Datenbasis für alle Beteiligten. Ferner lassen sich hierdurch außerdem Änderungen an den Konfigurationen einfacher pflegen und in das komplette System einfügen, ohne Fehler zu provozieren, die durch die komplexen Zusammenhänge innerhalb der Anforderungen leicht entstehen können, wenn nachträgliche Anpassungen erfolgen. [Fe07a]

3.6 Lastenheft

Das Lastenheft beschreibt den Soll-Zustand der zu entwickelnden Software und enthält alle ermittelten und an das System verbindlich gestellten Anforderungen. Auch hier unterscheidet man zwischen funktionalen und nicht-funktionalen Anforderungen, die auch als solche gekennzeichnet sind. Es dient oft auch als Grundlage für Ausschreibungen bei der Projektvergabe und ist fester Vertragsbestandteil zwischen Auftraggeber und Auftragnehmer. Die darin enthaltenen Spezifikationen legen die Rahmenbedingungen bezüglich der Entwicklung beim Auftragnehmer fest und werden von diesem dann in das Pflichtenheft überführt. Auf Grundlage des Lastenhefts wird also die gesamte Entwicklung des geforderten Systems begründet. Analog zur Konfigurationsidentifikation enthält dieses Dokument noch keine Aspekte bezüglich der expliziten Entwicklung und Umsetzung des Software-Projekts, sondern lediglich Anforderungen an das spätere „Enderzeugnis“ . Das Lastenheft sollte möglichst so gestaltet sein, dass Veränderungen an den Spezifikationen nachträglich ergänzt und verändert werden können und eine Rückverfolgung (traceability) dieser Prozesse möglich ist. Denn zumeist wird es vor Projektbeginn und auch während der Entwicklung an Veränderungen in den Anforderungen angepasst. Der Grund hierfür ist einerseits die Synchronisation der Anforderungen zwischen Stakeholdern und Auftragnehmern vor Projektbeginn und andererseits die Berücksichtigung von Aspekten, die erst in der tatsächlichen Entwicklungsphase auftreten. [Deu14]

3.7 Pflichtenheft

Das Pendant zum Lastenheft stellt das Pflichtenheft dar. In diesem Dokument werden die Gesamtspezifikationen des Projekts seitens des Auftragnehmers konkretisiert und hinsichtlich der tatsächlichen technischen Umsetzung festgelegt. Anhand der An-

forderungen des Lastenhefts wird in diesem Rahmen eine erste Grobarchitektur des Systems entwickelt und geeignet beschrieben. Ebenfalls werden neben der Entwicklung des eigentlichen Systems auch zu entwickelnde Untersysteme identifiziert und den jeweiligen Anforderungen zugeordnet, was auch eine erste Arbeitsteilung zwischen den Entwicklern nach sich zieht. Darüber hinaus werden auch vertragliche Aspekte einbezogen, die den Lieferumfang des fertigen Gesamtsystems und die damit verbundenen Abnahmekriterien umfassen. Um sicher zu stellen, dass letztlich alle Anforderungen im Pflichtenheft berücksichtigt und für eine konkrete Umsetzung eingeplant sind , wird eine Anforderungsverfolgung, sowohl bezüglich des Lastenhefts als auch in Richtung des Systems und den entsprechenden Untersystemen durchgeführt. Die Ausarbeitung des Pflichtenhefts erfolgt innerhalb der Projektumgebung beim Auftragnehmer zwischen den Experten, die für die spätere Erstellung der Systemkomponenten (evtl. auch Untersysteme) zuständig sind , sowie den Projektverantwortlichen, die ein Hauptaugenmerk auf den Gesamtentwurf des Systems haben und diesbezüglich auch in engem Kontakt mit dem Auftraggeber stehen. Dieser wiederum prüft, ob die Gesamtspezifikationen des Pflichtenhefts seinen Vorgaben entsprechen. Beide Dokumente entstehen somit in sehr enger Synchronisation zwischen allen Beteiligten des Projekts und unterliegen während des Entwicklungsprozesses auch dem Anpassungsprozess von Anforderungen und Umsetzungsentwürfen, was eine Veränderlichkeit und Nachverfolgbarkeit beider Dokumente voraussetzt.

[Deu14]

4 Versionsmanagement

Moderne Softwareprogrammierung wird immer komplexer, sie muss meist von mehreren Programmierern gemeinsam erledigt werden, dies erfordert einen hohen Organisationsaufwand, da jeder Programmierer nur noch einen kleinen Teil des Quellcodes erstellt und diese Teile zusammengeführt werden müssen. Ferner besteht ein System sowohl aus verschiedenen Versionen als auch Varianten, dies muss ebenfalls gemanaged werden insbesondere um „Bugs“ zu verhindern.
Das Erstellen einer neuen Version bedingt durch einen Ausschneiden, Einfügen oder Löschen von bereits vorhandenem Code wird dabei „history step“ ([Ben95], S. 41) genannt. Die Schrittweise Entwicklung wird durch die Versionskontrolle nachvollziehbar gemacht, da sowohl das Original als auch die Vorgängerversion für jeden „history step“ erhalten bleiben. So ist die Entwicklung auch später noch nachvollziehbar und im Gegensatz zu einem herkömmlichen Back-Up wurde mit der Prämisse gearbeitet so viel Speicherplatz wie möglich zu sparen. Dies wird durch Kompressionsverfahren ermöglicht. [BSI05]
Gemäß dem ITIL-Standard umfasst Versionsmanagement „alle Verfahren von der Anforderungsbearbeitung über die Planung der Umsetzung, Tests und Abnahme von Soft- und Hardwareversionen bis hin zur organisatorischen und technischen Vorbereitung der Einführung einer Komponente“. [BSI05], S. 20

4.1 Anfänge der Versionierung

SCCS (Source Code Control System)
SCCS wurde 1972 in den Bell Labs entwickelt und wurde später in C neu geschrieben um in UNIX Systemen eingesetzt werden zu können, diese Möglichkeit wurde auch genutzt weswegen SCCS in vielen UNIX-Distributionen zur Verfügung stand. Es war bis zum Erscheinen von RCS (siehe Unten) das am Meisten genutzte System zur Versionierung.
Die Idee von SCCS war, dass eine Software „linear“ entwickelt wird, also eine neuere Version seine jeweilige Vorgängerversion ablöst, wobei die Versionen dabei nach dem Schema „X.Y“ durchnummeriert werden. Bei einer größeren Änderung wird „X“ erhöht, bei einer kleineren Änderung das „Y“. Für jeden „history step“ wird dabei um Speicher-

platz zu sparen nur das „Delta “ , also die Änderung im Vergleich zur Vorgängerversion, gespeichert. Die nachfolgende Abbildung verdeutlicht dies, die Quader stehen dabei für Versionen, die Dreiecke für die jeweiligen Deltas.

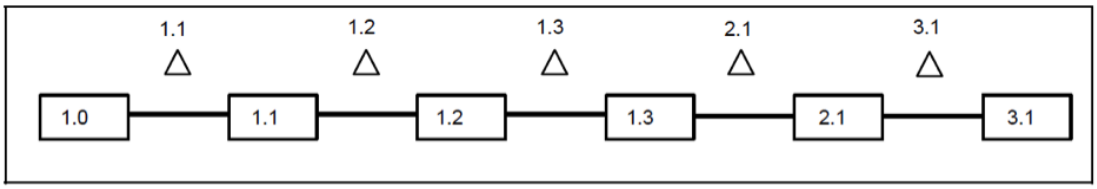

Abbildung 4.1: SCCS nach [Ben95], S. 43

Wichtig dabei ist, dass es sich um „forward deltas “ ([Ben95], S. 43), also vorwärts gerichtete Deltas, handelt. Dies heißt, dass SCCS immer die Ursprungsversion speichert und für diese dann alle weiteren Änderungen. Die gesamte Speicherung erfolgt dabei in einer einzigen Datei und es ist auch möglich, dass eine Version zwei Nachfolger hat, jedoch nicht mehr als zwei. [BSI05]

RCS (Revision Control System)

RCS wurde in den 80ern als Alternative zu dem bis dahin sehr populären SCCS von Walter F. Tichy als freie Alternative zu diesem Veröffentlicht. Mit RCS wurden viele der Funktionen wie z.B. Speicherung, Logging oder das Zusammenführen von Codeteilen automatisiert und somit vereinfacht. RCS arbeitete damit hauptsächlich mit dem UNIX *diff*-Kommando, das die Unterschiede zwischen zwei Dateien ermitteln kann.

Wie auch SCCS unterstützt RCS jedoch nur eine einzelne Datei zur Speicherung des Codes und es fehlt auch die Unterstützung für komplette Projekte bzw. Ordnerstrukturen und es war Entwicklern auch nicht möglich gemeinsam an einem großen Projekt zu arbeiten.

RCS war von Anfang an darauf ausgelegt, dass die Entwicklung einer Baumstruktur gleicht, also eine Version auch mehrere Nachfolgerversionen haben kann. Deswegen wird dies im Gegensatz zu SCCS auch besser unterstützt. Des Weiteren wurde in RCS auch das automatische Zusammenführen verschiedener Versionen implementiert. Dabei können Versionen von Verschiedenen Verästelungen automatisch zu einer neuen Version zusammengefasst werden und mögliche Differenzen in beiden Versionen werden Markiert und für den Programmierer sichtbar gemacht wodurch dieser manuell entscheiden kann welchen Teil des Codes er übernehmen will. Im Gegensatz zu SCCS

wird in RCS auch nur die aktuellste Version gespeichert und alle anderen älteren bzw. möglicherweise parallel erstellten Versionen sind über Deltas wiederherstellbar. Diese sind für denselben Ast rückwärts gerichtet und für eine Verästelung vorwärtsgerichtet. Ein Schema von RCS ist auf der nachfolgenden Abbildung ersichtlich. [BSI05]

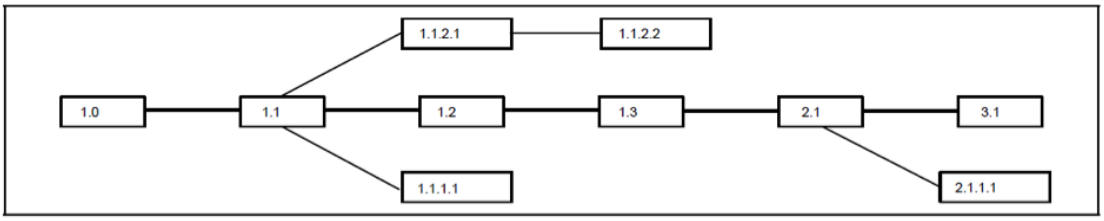

Abbildung 4.2: RCS nach [Ben95], S.45

Andere Versionierungssysteme

Natürlich sind die beiden erwähnten System (SCCS und RCS) nicht die einzigen verfügbaren Systeme. Es gibt weitere Möglichkeiten der Versionsverwaltung wie z.B. CVS oder VOODOO, auf diese soll jedoch nicht näher eingegangen werden. [BSI05]

4.2 Anwendungsbeispiel - Git

Ein verfügbares freies Programm für die Versionierung, welches heutzutage häufig genutzt wird, ist Git. Git wurde von Linus Torvalds ursprünglich für die koordinierte Entwicklung des Linux-Betriebssystems geschrieben. Im Gegensatz zu vielen anderen Tools arbeitet Git mit „software revisions “ ([Spi12], S. 1) und nicht mit Versionen die für viele Programmierer insbesondere bei Freizeitprojekten nicht von Belang sind. Stattdessen wollen diese nachvollziehen können welche Änderungen übernommen worden sind oder wie ein von ihnen geschriebener Code-Teil in das Programm eingebettet wurde.
In Git werden die Dateien vergleichbar mit einem kleinen „filesytem“ ([Cha09], S.5) gespeichert, für jede Änderung wird ein Abbild der aktuellen Datenstruktur erzeugt, dies ist auf der Abbildung 4.3 ersichtlich.
Git ermöglicht es Programmierern das Komplette „Software-Repository“ zu kopieren und ihre eigenen privaten Zweige zu erstellen und zu löschen, kleine unvollständige Änderungen einzureichen oder die komplette Code-Revision zu vertagen. Ferner ermöglicht es Git den Programmierern genau festzustellen wo eine Änderung durchgeführt wurde und es behält einen kompletten Graph zur Übersicht über alle Änderungen, so müssen

Sie sich nicht mit kleinen Dateien befassen in denen möglicherweise Änderungen erfolgt sind, sondern haben die Übersicht über das komplette Programm.

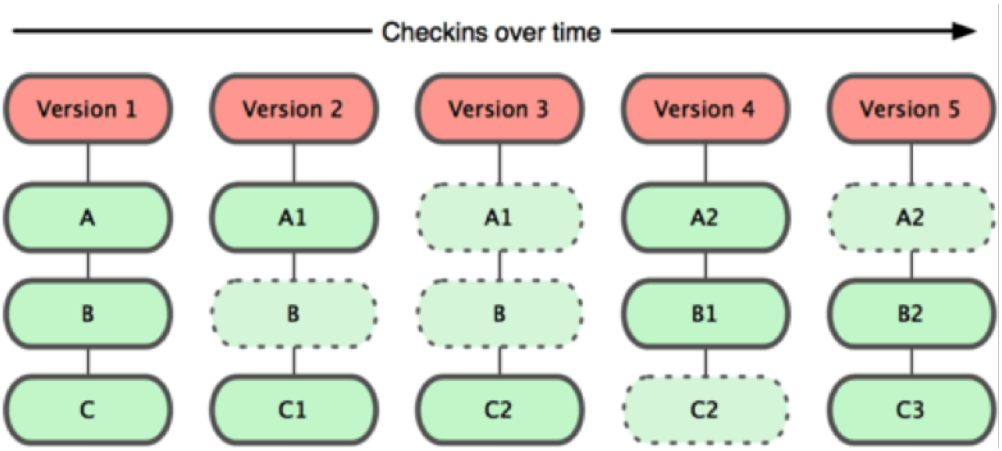

Abbildung 4.3: Git nach [Cha09], S.5

Die private Kopie des Repositorys ermöglicht auch neue Arbeitsabläufe. So können Programmierer z.B. gegenseitig den Code von ihren Repositorys laden oder ein Integrations-Manager kann von jedem seiner Programmierer frei und ohne diese zu beeinflussen Code holen und diesen in das „master repository“ ([Spi12], S2) einbetten. Ferner ermöglicht es Programmierer zu entlasten, da das Coding mehrstufig aufgebaut werden kann. Dies heißt, dass vergleichbar mit der Linux-Kernel-Entwicklung zunächst die Programmierer den Code schreiben, in der nächsten Stufe wird dieser Code in Teilprojekte integriert welche dann wiederum in das Gesamt-Projekt integriert werden können.
Ein weiterer Vorteil der lokalen Speicherung besteht darin, dass ein Programmierer nicht zwangsläufig Zugang zum Internet benötigt. Stattdessen kann er seine Änderungen „offline“ implementieren und zu einem Zeitpunkt seiner Wahl zum „master repository“ hinzufügen. Des Weiteren sind die Zugriffszeiten bei dieser Art der Speicherung geringer und die Produktivität sollte steigen.
Git ermöglicht es zudem ein Repository für die Öffentlichkeit zur Verfügung zu stellen. Mittlerweile kann dieses auch von einem Drittanbieter wie etwa GitHub gehostet werden. Wobei der Anbieter das „repository management“ ([Spi12], S. 2) über eine Browser-Schnittstelle gewährleistet.
[Cha09], [Spi12]

5 Build Management/ System Building

Hat man nun den Quellcode der einzelnen Software Komponenten in dem Software-Projekt erstellt, dann möchte man diesen ausführbar machen. Hierum kümmert sich der Build Prozess. Dieser Prozess ist zuständig für das Kompilieren und Linking der einzelnen Software Komponenten in ein ausführbares System. Da die Abhängigkeiten von den Quellcodes, Bildern und weiteren Artefakten bei großen Projekten sehr komplex und unübersichtlich sind, empfiehlt es sich, diese Abhängigkeiten nicht alle manuell zu kompilieren. Hinzu kann natürlich noch kommen, dass für unterschiedliche Systeme unterschiedliche Kombinationsmöglichkeiten von Einzelkomponenten für den Build Prozess benötigt werden. Man erkennt also, dass das einfache Kompilieren von Quellcode für ein Software Projekt nicht ausreicht, sondern man weitreichende Überlegungen dazu anstellen muss.
Für diesen Build Schritt in der Softwareentwicklung gibt es inzwischen Unterstützung von automatischen Tools. Diese Tools enthalten meistens eine Abhängigkeitsspezifikationssprache und einen Interpreter für diese Sprache. Ebenso ist meist noch die Möglichkeit zur verteilten Kompilation, die Auswahl von Tools und Support für die Instanziierung des Softwareprojektes gegeben.
Das bekannteste Tool für UNIX ist make für die C/C++ Entwicklung. Make ist dafür zuständig die Abhängigkeiten zwischen Sourcecode und Objektcode zu beobachten und re-kompiliert automatisch die Sourcen welche nach dem Erstellungsdatum des zugehörigen Objektcodes verändert wurden. Die Abhängigkeiten werden in Makefiles abgebildet, wie auch im folgenden Code ersichtlich wird:

Listing 5.1: Makefile für Speicherverwaltung aus Systemnahe Software 1-WS 2013/14

```
1 Sources := $(wildcard *.c)
2 Objects := $(patsubst %.c,%.o,$(Sources))
3 Target := testit
4 CC := gcc
5 CFLAGS := -Wall -std=gnu11
6 $(Target): $(Objects)
7 .PHONY: clean depend realclean
8 clean:
9 rm -f $(Objects)
```

```
10 realclean:   clean
11             rm -f $(Target)
12 depend:
13             gcc-makedepend $(CFLAGS) $(Sources)
14 # DO NOT DELETE
15  my_alloc.o: my_alloc.c my_alloc.h my_system.h
16  my_system.o: my_system.c my_system.h
17  testit.o: testit.c my_alloc.h my_system.h
```

In Listing 5.1 werden zuerst die Quellen, die Objekte (die Abhängigkeiten) und die Zielquelle definiert, ebenso der benötigte Kompiler mit Option. Anschließend werden die einzelnen Befehle beschrieben, also was passiert bei dem Aufruf von *make clean, make realclean* und *make depend*. In den Zeilen 15-17 findet man dann die einzelnen Abhängigkeiten, die mit make depend gefunden wurden.
Weitere bekannte Tools sind Apache Ant, das wohl häufigste verwendete Tool für Java, das analog zu make in einer Datei build.xml die Abhängigkeiten beschreibt, Maven (ebenfalls Java), Jam, MS Build (.NET) und scons. Doch auch an solche automatischen Tools gibt es eine Anforderungsliste. [Som00]

- Enthalten die Build Anweisungen alle benötigten Komponenten ?
- Ist für jede Einzelkomponenten die richtige Version spezifiert?
- Sind alle benötigten Dateien verfügbar ?
- Sind die Referenzen innerhalb der Komponenten richtig, also ruft die eine Komponente eine andere mit den richtigen Parametern und Namen auf?
- Wird die Software auch für das richtige Betriebssystem erstellt?
- Mit welcher Kompilerversion und weiteren benötigten Software-Tools wird dieser Build-Prozess durchgeführt?

5.1 Continous Build

Schnell stellt sich jedoch die Frage wie oft denn ein solcher Build Prozess durchgeführt werden soll? Soll er nach jeder Änderung, jeder fertigen Teilkomponente durchgeführt werden, um die Funktionen zu testen, oder soll erst am Ende vom Projekt ein kompletter Durchlauf des Build-Prozesses durchlaufen?

Hierfür gibt es keine richtige und falsche Antwort. Die sollte je nach Komplexität des Softwareprojekts individuell entschieden werden. Denn ein solcher Build Prozess kann je nach Komplexität der zu erstellenden Software sehr rechenintensiv und lange sein. Zum einen wird ein Build aufwendiger, je seltener dieser durchgeführt wird, da die Fehlerwahrscheinlichkeit steigt und die Einzelkomponenten der Software auseinander laufen können.
Anderseits wiederum ist das ein Nachteil wenn man den Build zu oft durchführt, da bei großen Software Projekten viele Ressourcen benutzt werden, die der Build zu berücksichtigen hat und somit immer rechenintensiver wird.
Die optimale Strategie muss für jedes Softwareprojekt individuell gefunden werden. Wenn man nach jeder Änderung ein Build durchführen möchte, dann bezeichnet man dies als Continuous Integration. Durch Verwendung von Tools kann hierbei eine Überwachung der Ressourcen und der Aufruf des Build-Tools automatisiert werden. Ebenso kann die Automatisierung des Test nach dem erfolgten Build erfolgen. [Kas07], [Che09]
Im der folgenden Abbildung 5.1 sieht man die Abhängigkeiten, die in einem Software-Projekt vorliegen, darunter auch den Build Prozess.

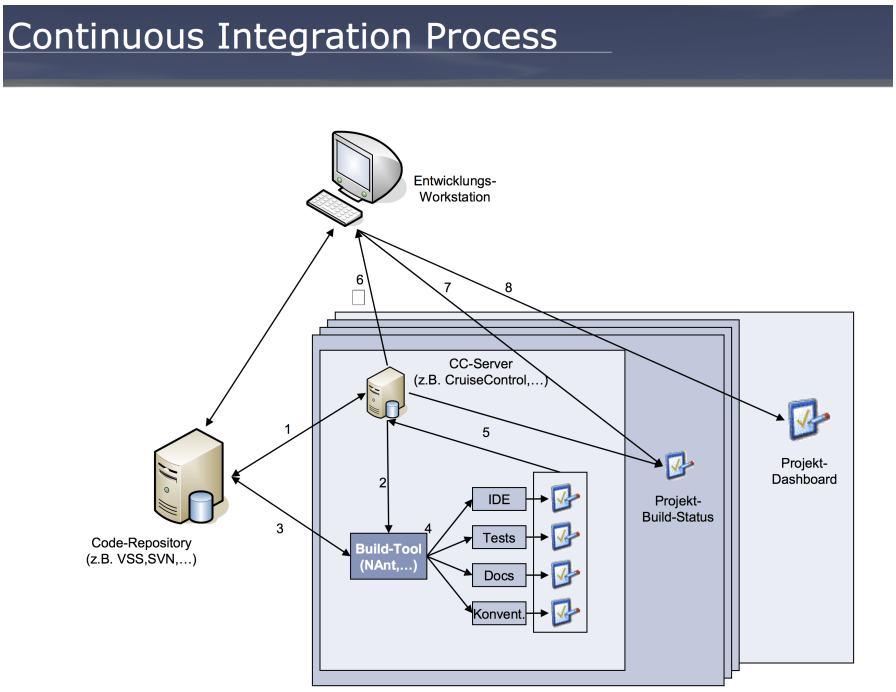

Abbildung 5.1: Continuous Build Prozess eines Softwareprojekts nach [PDEBD11]

5.2 Die Kosten eines fehlgeschlagenen Builds

Was passiert wenn der Build Prozess fehlschlägt, weil aus welchen Gründen auch immer die Abhängigkeiten nicht korrekt definiert wurden?
Man unterscheidet hierbei bei den Fehlerentstehung zwischen einem *Loud failure*, einem *Silent failure* und einem *Intentional failure*. Der Loud failure ist ist der Zustand, wenn der Kompiler eine Datei nicht übersetzt, oder das Linking nicht korrekt ist und den Build zum Abbruch bringt. Der Silent failure ist im Regelfall ein schwer zu findender Fehler, denn der Build war hier erfolgreich, aber der gewünschte Output ist nicht korrekt, weil z. B. ein geänderter Code nicht re-kompiliert oder eine alte Bibliothek verwendet wurde.
Der dritte Fehler, der Intentional failure, der vermutlich am schwersten zu finden ist, entsteht, wenn der Quellcode Backdoors, etc. enthält. [Che09]

5.3 Effektiver Build

Wie kann man diesen Build Prozess denn nun verbessern, wenn oft etwas schief geht oder der Build Prozess sehr lange dauert?
Folgende simple Antworten sind hierbei zu finden, [Che09]:

- Verbesserung der Qualität der Software
- Reduzierung von verschwendeter Zeit
- Verhinderung von Sicherheitsrisiken
- Compliance Vorgaben einhalten

6 Release Management

6.1 Aufgaben und funktionale Einordnung

Das Release Management ist für die effektive, sichere und nachvollziehbare Durchführung von Änderungen der IT-Infrastruktur verantwortlich. Aufgaben des Release Managements sind die Planung, Überwachung und Durchführung von Rollouts und Rollins. Dies erfolgt in Abstimmung mit dem Change Management. Ferner hat das Release Management die Aufgabe die Gesamtheit der Änderungen der IT zeitlich, technisch und inhaltlich zu bündeln und aufeinander abzustimmen. [Wik13b], [Pil10]
Die Funktion des Release Management ist dem Service Support zugeordnet. Dieser hat die operativen Prozesse zur Behandlung von Service-Unterbrechungen und Durchführung zur Aufgabe und garantiert somit die Aufrechterhaltung der IT-Services. Der Service Support ist wiederum dem IT Service Management zugeordnet. [Wik13c]

6.2 Einteilung von Releases

Emergency Release:
Behebung von Störungen oder signifikanten Problemen in der IT. Ähnelt dem Minor Release, benötigt jedoch i. d. R. viel weniger Zeit.
Minor Release:
Dieser Release enthält kleinere Erweiterungen und Fixes. Werden häufiger als Major Releases durchgeführt, benötigen jedoch weniger Zeit und Planung. Ersetzen vorangegangene Emergency Releases.
Major Release:
Beinhaltet signifikante neue Funktionalitäten, Upgrades, oder neue Services in der IT. Ersetzen alle vorangegangen Minor und Emergency Releases, welche bezüglich eines Problems in der IT durchgeführt wurden und macht diese überflüssig. Dieser Release wird selten durchgeführt, benötigt jedoch mehr Zeit und Planung als Minor und Emergency Releases. [How12]

6.3 Teilprozesse

Die Teilprozesse des Release Management nach ITIL sind in der folgenden Abbildung 6.1 dargestellt.

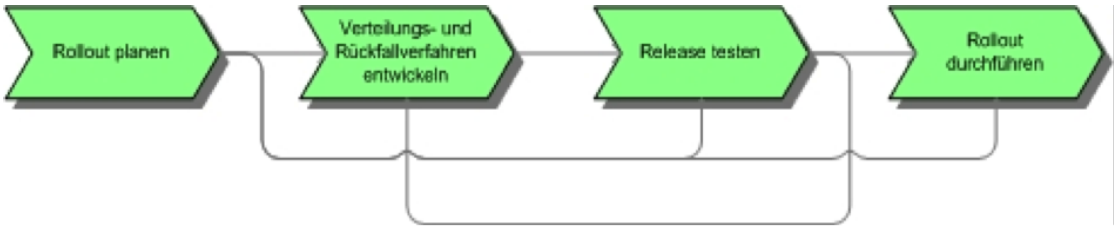

Abbildung 6.1: Teilprozesse des Release Management nach ITIL [Wik13b]

Rollout planen:
Für die Planung eines Release wird eine Vielzahl von Informationen benötigt. Diese sind in einem Release-Plan und einem Release-Steckbrief gebündelt. [SS08]
Release-Plan:

- Abstimmung über den Inhalt des Releases
- Absprache über zeitliche Reihenfolge, Standorte und Organisationsbereiche
- Klärung der eingesetzten Hard- und Software
- Klärung der erforderlichen Mittel für Hardware, Software und Dienstleistungen
- Abstimmung der Verantwortlichen
- Dienstleistungen, welche von Dritten benötigt und über das Supplier Management koordiniert werden
- Erstellen von Back-Out-Plänen
- Aufwandsabschätzung

Release-Steckbrief:

- Name des Release
- Version des Release
- Beschreibung des Release
- Dokumentationsablage

- Historie

Verteilungs- und Rückfallverfahren entwickeln:
In diesem Teilprozess werden die technischen Voraussetzungen zur Installation, bzw. der Verteilung der neuen Komponenten des Release geschaffen. Des Weiteren werden hier Vorkehrungen für das Zurückfahren des Rollouts für den Fall von unvorhergesehenen Problemen getroffen. [Wik13b]

Release testen:
Der Test des Release erfolgt in der Regel durch das Betriebspersonal (=> Realitätsnähe). Besonderer Beachtung sollten hierbei der Funktionsweise, den technischen Betriebsaspekten, dem Leistungsverhalten und der Integration in die vorhandene Infrastruktur geschenkt werden. [SS08]
Im Rahmen des Release Prozesses sind drei Arten von Tests vorgesehen. Dies sind der Unit-, Integration- und der Abnahme-Test. [Pil10]
Für den Unit-Test ist der Applikationsexperte verantwortlich. Hierbei wird geprüft, ob jedes Element des Release so funktioniert, wie es spezifiziert wurde. Es werden Testprogramme und Testskripte geschrieben. Hierbei werden einfache Testfälle benutzt. Die Verantwortung für den Integration Test trägt der IT-Projektleiter. Dieser überprüft, ob die definierten Funktionen und Schnittstellen funktionieren und im Verbund funktionieren. Hierbei sollen möglichst alle Benutzer-Szenarien berücksichtigt werden. Für den Abnahme-Test ist der Fachprojektleiter verantwortlich. Dieser Test unterteilt sich in einen User Acceptance Test und einen Regression Test. Im User Acceptance Test wird geprüft, ob das System alle geforderten Business-Funktionalitäten abdeckt und den korrekten Output generiert. Im Regression Test wird sichergestellt, dass die Systemveränderungen nicht einen vorher funktionierenden Systemteil beeinflusst haben.

Rollout durchführen:
Dieser Prozess hat einerseits zum Ziel, die Release-Komponenten in die IT-Umgebung auszurollen. D.h. die neuen Funktionalitäten werden auf alle Zielobjekte ausgebreitet und installiert. Die Mitarbeiter, welche von den Änderungen der Releases betroffen sind, werden geschult. Die Dokumentation über entsprechende Konfigurationselemente wird aktualisiert. [Pil10]
Andererseits findet eine Erfolgskontrolle statt. [Wik13b]

7 Change Management

Verändern sich die Rahmenbedingungen eines Unternehmens, so muss sich das Unternehmen an diese dementsprechend anpassen. Wird beispielsweise eine neue gesetzliche Regelung eingeführt, die die Erhöhung des Mehrwertsteuersatzes vorschreibt, muss diese sowohl in die Geschäftsprozesse als auch in die komplette IT-Landschaft integriert werden. Damit diese Änderung durchgeführt werden kann, ohne das laufende IT-System negativ zu beeinflussen, muss dieser Änderungsprozess möglichst effizient gesteuert werden. Hierfür werden standardisierte Verfahren definiert, mithilfe deren jede Veränderung hinsichtlich Notwendigkeit und Risiko umfassend analysiert wird. Ist eine Änderung erforderlich und kann diese ohne weitere Störungen durchgeführt werden, wird diese genehmigt. Ziel des Change Managements ist es also, alle Veränderungen, von der Registrierung bis hin zur Implementierung, zu kontrollieren, um das Risiko einer Instabilität möglichst minimal zu halten.
Eine Änderung kann durch Störungen im laufenden Betrieb, bei der Benutzung oder auch durch Verbesserungsvorschläge, Erweiterungen, Kundenanfragen bzw. Rückfragen oder durch geänderte Anforderungen bzw. Geschäftsbedingungen hervorgerufen werden. Trifft eine Anfrage für eine Veränderung ein, wird der Change Management Prozess ausgelöst. Diese Veränderung betrifft in der Regel das Ändern, Hinzufügen oder das Entfernen von Komponenten des Konfigurationsmanagements wie beispielsweise Software, Hardware, Anwendungen oder Netzwerkkomponenten. Sowohl Kunden als auch Prozessbeteiligte und das Change Management selbst können eine Veränderung veranlassen.

7.1 Der Change Management Prozess

Jeder Change muss den Prozess durchlaufen, unabhängig von der Schwere seiner Auswirkung oder seiner Wichtigkeit. Dabei ist der Prozess so aufgebaut, dass ein Change immer für den nächstfolgenden Schritt freigegeben werden muss. Ziel dabei ist, alle Change-Aktivitäten zunächst umfassend auf potenzielle Gefahren zu untersuchen, bevor der Change in der Liveumgebung implementiert wird. So können unkontrollierte Störungen und Unterbrechungen in der IT-Umgebung vermieden werden. Zudem sind alle Aktivitäten, die einen Change betreffen, zu dokumentieren.

Soll eine Veränderung an der IT-Infrastruktur umgesetzt werden, wird ein Request for Change durch den Antragsteller ausgelöst. Diese Anfrage wird anschließend an das Change Management weitergeleitet. Der Verantwortliche im Change Management Prozess prüft nun die Anfrage. Entspricht dieser nicht allen Qualitätsanforderungen oder lässt sich der Change nicht durchführen, wird der Request for Change abgelehnt. Ist er jedoch notwendig, um einen betrieblichen Ablauf gewährleisten zu können, wird der Request for Change registriert und für den nächsten Schritt freigegeben.
Bevor der Auftrag, eine Änderung an einem bestimmten Konfigurationselement vorzunehmen, genehmigt wird, muss dieser hinsichtlich seiner Dringlichkeit und Auswirkung auf die IT-Umgebung analysiert werden. Auf Basis dieser Analyse wird das weitere Vorgehen bestimmt.
Hierbei wird dem Change zunächst eine Priorität zugewiesen. Die Priorität ergibt sich aus der Dringlichkeit der Durchführung des Changes und dem Ausmaß des Problems, welches er beheben soll. Muss die Durchführung eines solchen sofort erfolgen, hat dieser höchste Priorität. Dies ist dann der Fall, wenn ein Problem auftritt, dass die Funktionsweise des IT-Services enorm beeinträchtigt und somit schon einen gewissen Schaden verursacht hat. Es ist dementsprechend absolut notwendig, um einen ordnungsgemäßen Betrieb wiederherzustellen, dass der Change sofort in der Liveumgebung implementiert wird und nicht erst darauf gewartet wird, dass dieser nach einem langwierigen durchlaufen des Prozesses genehmigt wird. Hierbei müssen sofort entsprechende Maßnahmen eingeleitet werden. Dazu kann die Durchführung, weniger wichtiger Changes, zeitlich verzögert werden, um gewährleisten zu können, dass für die Durchführung dieses Changes genügend Ressourcen zur Verfügung stehen. Je nach Dringlichkeit, kann auf das Testen der Implementierung verzichtet werden. Changes mit einer niedrigen Priorität sind zwar wünschenswert aber nicht notwendig. Diese können gegebenenfalls verschoben und zu einem späteren Zeitpunkt durchgeführt werden.
Anschließend wird dem Change eine Kategorie zugeteilt. Die jeweilige Kategorie gibt das Ausmaß an, inwieweit das laufende System durch den Veränderungsprozess belastet und gegebenenfalls in seiner Funktionalität beeinträchtigt wird. Je höher das Risiko der Anpassung eines Konfigurationselements an die geänderten Anforderungen, umso höher ist die zugeordnete Kategorie. Handelt es ich bei den Changes um Veränderungen, die regelmäßig durchgeführt werden, beispielsweise das Ändern eines Passworts, so wäre es unvorteilhaft, wenn diese Veränderung jedes Mal erneut genehmigt werden

müsste. Solche Changes stellen für das Unternehmen kein großes Risiko dar. Aufgrund dessen werden solche Changes in die niedrigste Kategorie eingeteilt. Alle Changes, die in diese Kategorie fallen, müssen nicht mehr ausdrücklich vom Change Verantwortlichen genehmigt werden, sondern gelten schon als vorab autorisiert. Damit wird ein schneller Durchlauf des Veränderungsprozesses garantiert. Die Kosten dieses Changes können gut eingeschätzt werden. Zudem muss die Implementierung der Changes nicht immer von neu erfolgen.
Changes, die riskieren andere Komponenten des IT-Systems negativ zu beeinträchtigen, werden einer höheren Kategorie zugeordnet. Um jedoch beurteilen zu können, wie sich die Änderung eines Konfigurationselements auf die restliche IT-Infrastruktur auswirkt, muss zunächst festgestellt werden, in welcher Beziehung das jeweilige Konfigurationselement mit anderen Konfigurationselementen steht. An diesem Punkt kommt das Konfigurationsmanagement zum Einsatz. Das Konfigurationsmanagement liefert Informationen darüber, welche Wechselwirkungen zwischen den einzelnen Konfigurationselementen bestehen. Beeinflusst der Change die Konfigurationselemente nur geringfügig, handelt es sich um einen Change mit geringem Risiko. Hier müssen noch keine Maßnahmen zur Absicherung des IT-Systems definiert werden. Zudem lässt sich die Veränderung ohne großen Aufwand umsetzen. Dieser Change fällt in die Kategorie 1. Alle Veränderungen dieser Kategorie werden vom Change Manager umfassend beurteilt und anschließend genehmigt.

- *Der Change Manager* leitet den kompletten Change Management Prozess. Es ist für einen ordnungsgemäßen Ablauf des Change Management Prozesses verantwortlich und überwacht alle Änderungsaktivitäten. Er hat bei geringfügigen Changes die alleinige Befugnis diese für die nächste Phase freizugeben. Dazu führt er im Vorfeld für jeden Change eine umfassende Risikoanalyse durch und bewertet dessen Auswirkung auf den laufenden Betrieb.

Schwerwiegende Changes können nur mit großem Aufwand umgesetzt werden. Gerade deswegen, da im Problemfall mit Unterbrechungen im IT-Service gerechnet werden muss. Die Wahrscheinlichkeit, dass es zu Störungen im ordnungsmäßigen Ablauf der Geschäftsprozesse kommt, ist enorm. Hier müssen die Risiken ausreichend abgeschätzt werden und Maßnahmen entwickelt werden, um einen negativen Einfluss auf die IT-Services weitgehend zu vermeiden. Um sicherzustellen, dass dieser Change

hinreichend beraten wird, wird dieser Change an das Change Advisitory Board (CAB) weitergeleitet.

- *Das CAB* unterstützt den Change Manager bei risikoreichen Changes hinsichtlich der Entscheidung über die Annahme oder Ablehnung des Changes. Dazu setzt sich das CAB aus erfahrenen Personen zusammen, die sowohl die Business-Seite als auch den technischen Standpunkt vertreten. Mitglieder des CAB sind in der Regel Kunden, Anwender, IT-Experten, Entwickler und Tester, evtl. Subunternehmen oder Hersteller, sowie der Change Manager selbst. Die Zusammensetzung des CAB hängt dabei von dem zu diskutierenden Change ab.

Je nachdem welche Kategorie und Priorität dem Change zugewiesen wurde, wird dieser durch die dafür zuständige Autorisierungsebene genehmigt. Dabei sind jedoch nicht nur die Auswirkung und Dringlichkeit von Bedeutung, sondern auch die Kosten und die Zeit der Umsetzung, als auch die zur Verfügung stehenden Kapazitäten und Ressourcen.
Wird ein Change genehmigt, beginnt die Planung des Changes durch den Change Manager. Dazu wird ein genauer Zeitplan erstellt, indem alle weiteren Aktivitäten klar definiert und zeitlich festgelegt sind. Dieser enthält unter anderem das vorgesehene Implementierungsdatum. Alle Aufgaben müssen klar verteilt sein. Zur optimalen Kontrolle, müssen alle durchgeführten Aktivitäten in dem Zeitplan dokumentiert werden.
Anschließend wird der Change an die Entwicklungsabteilung weitergereicht, die für den Entwurf, die Implementierung und das Testen zuständig ist. Der Change Manager hat dabei nur noch eine Koordinationsfunktion, da nun andere Personen für die Umsetzung des Changes zuständig sind. Er überwacht alle Aktivitäten und stellt sicher, dass diese im vorgesehenen zeitlichen Rahmen ausgeführt werden.
In der Entwurfsphase werden die Anforderungen an die durchzuführende Veränderung in einer technischen Lösung umgesetzt. Grundlegend ist die Strukturierung der Software. Dabei wird festgelegt, in welcher Beziehung die einzelnen Konfigurationskomponenten zueinander stehen und welches Verhalten diese haben. Der Entwurf dient als Vorlage, nach dem programmiert wird.
Die Implementierung liegt in den Händen von Technikern, die nun für die Durchführung des Changes verantwortlich sind. Das Change Management selbst arbeitet hier nur im Hintergrund und sorgt dafür, dass die Implementierung im dem festgelegten zeitlichen Rahmen erfolgt. Wurde die Implementierung durchgeführt, muss diese getestet werden. Inwieweit der Change getestet wird, hängt von der Kategorie ab. Getestet werden hierbei

die Funktionen der Konfigurationskomponenten sowie der Programmcode. Zudem wird die gesamte Implementierungsphase genau analysiert. Wichtigstes Kriterium hierbei ist, ob mit dem Change die erwartete Veränderung an der IT-Infrastruktur erreicht wurde. Weitere Kriterien sind:

- Traten bei der Implementierung und im gesamten Change-Prozess Störungen auf?
- Waren die zur Verfügung stehenden Ressourcen und das Budget ausreichend?
- Wie reagiert die IT-Infrastruktur auf den Change bzw. treten Qualitätsverluste auf?

Ist der Test nicht erfolgreich, muss das sogenannte Fall-Back Szenario eingeleitet werden. Dazu wurde im Vorfeld ein Fall-Back Plan erstellt. Dieser Plan erlaubt es, den vorherigen Zustand wiederherzustellen, falls die Implementierung fehlerhaft ist und nicht zum versprochenen Erfolg führt. Ist der Test erfolgreich, wird der Change durchgeführt und der Change-Management Prozess ist beendet.
Zum Abschluss muss die Konfigurationsmanagementdatenbank hinsichtlich der Veränderung an den Konfigurationskomponenten aktualisiert werden, da möglicherweise neue Komponenten hinzugekommen sind, die in die Datenbank mitaufgenommen werden müssen. Das Konfigurationsmanagement muss immer über die aktuelle, gegebenenfalls geänderte, IT-Infrastruktur Bescheid wissen.
Für ein Unternehmen ist es vorteilhaft, Methoden zu definieren, mithilfe derer ein Change an einem oder mehreren Konfigurationskomponenten, kontrolliert und verwaltet werden kann. Dementsprechend können die Risiken leichter analysiert und eingeschränkt werden. Zugleich können Unterbrechungen und Qualitätsverluste im laufenden Betrieb gemindert werden. Geänderte Anforderungen können somit im Unternehmen schneller und effizienter zu geringen Kosten durchführt werden.

[OUO14], [MLS13], [Wik13a] [Wi10], [ISM13]

A Anhang

Im Sinne dieses Seminar wurde die Ausarbeitung mit LaTex in Git erstellt. Das Projekt ist unter folgender Adresse zu finden.
https://github.com/dgbass/Seminar_EBIS

Abkürzungsverzeichnis

bzw.	beziehungsweise
etc.	et cetera
i. d. R.	in der Regel
ISO	Internationale Organisation für Normierung
RCS	Revision Control System
SCCS	Source Code Control System
SCM	Software Configuration Management
z. B.	zum Beispiel

Abbildungsverzeichnis

Literaturverzeichnis

[Ben95] BENDIX, Lars: *PhD dissertation, Configuration Management and Version Control Revisited*. http://www.itu.dk/~oladjones/semester2/advancedOOP/materials/Dissertation.pdf. Version: 1995, Abruf: 2014-05-28

[BSI05] BSI: *ITIL und Informationssicherheit*. http://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/ITIL/itil_pdf.pdf?__blob=publicationFile. Version: 2005, Abruf: 2014-05-28

[Cha09] CHACON, Scott: *Pro Git*. https://github.s3.amazonaws.com/media/progit.en.pdf. Version: 2009, Abruf: 2014-06-13

[Che09] CHELF, B.: *Software Build Analysis- Eliminate Production Problems to Accelerate Development*. Coverity. http://www.coverity.com/library/pdf/Coverity-Software-Build-Analysis.pdf. Version: 2009, Abruf: 2014-06-09

[Deu14] DEUTSCHLAND, Bundesrepublik: *Das V-Modell XT:Anforderungen (Lastenheft)*. http://ftp.uni-kl.de/pub/v-modell-xt/Release-1.2/Dokumentation/html/index.html?refer=http://ftp.uni-kl.de/pub/v-modell-xt/Release-1.2/Dokumentation/html/14794f684e963e8.html. Version: 2014, Abruf: 2014-06-02

[DK11] DR. KNIESEL, G.: *Kapitel 2 Software Configuration Management*. https://sewiki.iai.uni-bonn.de/_media/teaching/lectures/se/2011/folien/02-scm.pdf. Version: 2011, Abruf: 2014-06-14

[Edu04] EDUCATION, Pearson: *What Is Software Configuration Management ?* http://www.pearsonhighered.com/assets/hip/us/hip_us_pearsonhighered/samplechapter/0321200195.pdf. Version: 2004, Abruf: 2014-06-14

[Fe07a] FORSCHUNG E.V., Fraunhofer G. a.: *re-wissen.de: Anforderungsspezifikation*. http://www.re-wissen.de/Wissen/Anforderungsspezifikation/. Version: 2007, Abruf: 2014-06-10

[Fe07b] FORSCHUNG E.V., Fraunhofer G. a.: *re-wissen.de: Funktionale Anforderungen erheben*. http://www.re-wissen.de/Wissen/Anforderungserhebung/Praktiken/Funktionale_Anforderungen_erheben.html. Version: 2007, Abruf: 2014-06-10

[Gbt] GBT.CH: *Konfigurationsmangement*. http://www.gbt.ch/Lexikon/K/Konfigurationsmanagement.html, Abruf: 2014-06-14

[GDHHIS07] GEISSER, Michael ; DR. HERRMANN, Andrea ; HILDENBRAND, Tobias ; ILLES-SEIFERT, Timea: *Verteilte Softwareentwicklung und Requirements-Engineering: Egebnisse einer Online-Umfrage*. http://www.sigs.de/publications/os/2007/06/herrmann_geisser_OS_06_07.pdf. Version: 2007, Abruf: 2014-06-14

[Ham12] HAMPE, Klaus-Dieter: *ISO 10007 Leitfanden zum Konfigurationsmanagement*. Deutsche Gesellschaft für Qualität. http://www.qm-hamburg.de/archiv.qm-hamburg/Archiv/20120521_Konfigurationsmanagement.pdf. Version: 2012, Abruf: 2014-06-14

[HB09] HÜTTERMANN, Michael ; BOROSCH, Reinhard: *Software-Konfigurationsmanagement: Mit agilen Strategien Wahrscheinlichkeiten erhöhen*. http://huettermann.net/artikel/SCM-Huettermann.pdf. Version: 2009, Abruf: 2014-06-14

[How12] HOWARD, D: *IT Release Management*. 1.Auflage. CRC Press, 2012. – ISBN 978–1–4665–0914–6

[ISM13] IT-SERVICE MANAGEMENT, Munich I.: *Change Management*. http://www.mitsm.de/itil-wiki/prozess-beschreibungen-deutsch/change-management. Version: 2013, Abruf: 2014-06-04

[Kas07] KASPARICK, M.: *Software-Konfigurationsmanagement*. https://swt.cs.tu-berlin.de/lehre/sepr/ws0708/referate/SCM_Ausarbeitung.pdf. Version: 2007, Abruf: 2014-06-09

[MLS13] MEISTER, F. ; LORENZ, K. ; SCHAARSCHMIDT, G.: *Zehn Stolpersteine im Change Management*. http://www.computerwoche.de/a/zehn-stolpersteine-im-change-management,2532992. Version: 2013, Abruf: 2014-06-05

[OUO14] OSSIETZKY UNIVERSITÄT OLDENBURG, Carl von: *Change Management*. http://www.uni-oldenburg.de/itdienste/ueber-uns/it-service-management/change-management/. Version: 2014, Abruf: 2014-06-04

[PDEBD11] PROF. DR. EICKER, S ; B.SC. DIERMANN, A.: *Paradigmen und Konzepte der Softwareentiwcklung II (PKS II)*. Universität Duisburg-Essen, Lehrstuhl für Wirtschaftsinformatik und Softwaretechnik. http://www.softec.wiwi.uni-due.de/studium-lehre/lehrveranstaltungen/sommersemester-11/pks2-3274/download/PKSII_Vorl7_Buildmanagement_v1.1.pdf/. Version: 2011, Abruf: 2014-06-09

[Pil10] PILORGET, L.: *MIIP - Modell zur Implementierung der IT-Prozesse*. 1. Auflage. Viehweg + Teubner Verlag| Springer Fachmedien Wiesbaden GmbH, 2010. – ISBN 978–3–8348–1308–4

[PP14] PROF. PARTSCH, H.: *Vorlesungsskript Requirements Engineering*. Universität Ulm, Institut für Programmiermethodik und Compilerbau, 2014

[Som00] SOMMERVILLE, I.: *Software Engineering*. 6.th edition. Addison Wesley, 2000. – ISBN 9780201398151

[Spi12] SPINELLIS, Diomidis: *Tools of the Trade - Git*. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6188603. Version: 2012, Abruf: 2014-05-28

[SS08] SCHIEFER, H. ; SCHITTERER, E.: *Prozesse optimieren mittels ITIL: Abläufe mittels Prozesslandkarte gestalten - Compliance erreichen und Best Practices nutzen mit ISO 20000, BS 15000 & ISO 9000*. 2. überarbeitete Auflage. Viehweg + Teubner Verlag | GWV Fachverlage GmbH Wiesbaden, 2008. – ISBN 978–3–8348–0503–4

[Wi10] WIKI-ITIL.DE: *Prozess Change Management*. http://www.wiki-itil.de/Prozess_Change_Management. Version: 2010, Abruf: 2014-06-04

[Wik13a] WIKI.DE: *Change Management ITIL V2*. http://wiki.de.it-processmaps.com/index.php/Change_Management_-_ITIL_V2. Version: 2013, Abruf: 2014-06-05

[Wik13b] WIKI.DE: *Release Management*. Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Germany License. http://wiki.de.it-

processmaps.com/index.php/Release_Management. Version: 2013, Abruf: 2014-06-10

[Wik13c] WIKI.DE: *Service Support*. Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Germany License. http://wiki.de.it-processmaps.com/index.php/Service_Support. Version: 2013, Abruf: 2014-06-10