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

1.2 Gliederung

2 Software Configuration Management (SCM)

Steuert und verwaltet verteilte Software-Projekte. Ganz normaler Text.

3 Konfigurationsidentifikation

3.0.1 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 Personenkreisen zu synchronisieren, damit sich während der Arbeitsprozesse keine Eigenschaften aus einem Selbstzweck entwickeln, die den eigentlichen Vorgaben nicht dienlich sind.

3.0.2 Beteiligte Personen

Es liegt auf der Hand, dass verteilte Softwareprojekte zumeist ein hohes Maß an Arbeitsteilung aufweisen. Dabei agieren die beteiligten Personen häufig auch räumlich voneinander getrennt. Ferner muss man hinsichtlich der Anforderungen außerdem noch zwischen verschiedenen Interessensgruppen differenzieren, welche im Regelfall auch unterschiedliche Sichtweisen und Erwartungen bezüglich des Projektes haben. Im Allgemeinen muss man zwischen Stakeholdern und der Projektumgebung, bzw. den Projektverantwortlichen unterscheiden. Wie bereits erwähnt ist die korrekte Handhabe von Anforderungen wichtig, um den ursprünglichen Projektzweck nicht zu verfehlen. Man sollte die beteiligten Personenkreise aber auch hinsichtlich der Rückverfolgbarkeit

(traceability) von Anforderungen separat betrachten.

Stakeholder (Auftraggeber)

Als Stakeholder werden in diesem Kontext alle Interessensgruppen gesehen, welche die Kundensicht auf das Projekt repräsentieren. Sie stellen also vor allem Anforderungen aus der Benutzersicht auf, die weniger explizit und auf das Endergebnis gerichtet sind. Diese Anforderungen sind von höchster Relevanz und geben die Rahmenbedingungen für detailliertere Konfigurationsaspekte vor.

Projektumgebung (Auftragnehmer)

In erster Linie sind sie für die Realisierung der gestellten Anforderungen zuständig und befassen sich mit der Umsetzung des kompletten Software-Projekts. Aber es ist naheliegend, dass sich während dieses Prozesses auch hieraus neue Anforderungen ergeben, ohne die das Projekt nicht möglich ist. Diese Anforderungen sind wesentlich näher an die eigentliche Softwareentwicklung angelehnt und entstehen oft viel später als die Konfigurationsaspekte der Shareholder.

3.0.3 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]

3.0.4 Funktionale Anforderungen

Funktionale Anforderungen legen ausschließlich fest, was das System tun soll (funktional). Es wird jedoch nicht expliziert, wie es umzusetzen ist 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]

3.0.5 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]

3.0.6 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 ihren fachlichen Konventionen und deren Rolle innerhalb des Projekts adäquat abbilden (vgl. Fraunhofer_2 2007). 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]

3.0.7 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]

3.0.8 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 Anforderungen 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 (vgl. BRD 2004).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 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 4.1: 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 4.1 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?

4.0.9 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 Softwareprojekt 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 4.1 sieht man die Abhängigkeiten, die in einem Software-Projekt vorliegen, darunter auch den Build Prozess.

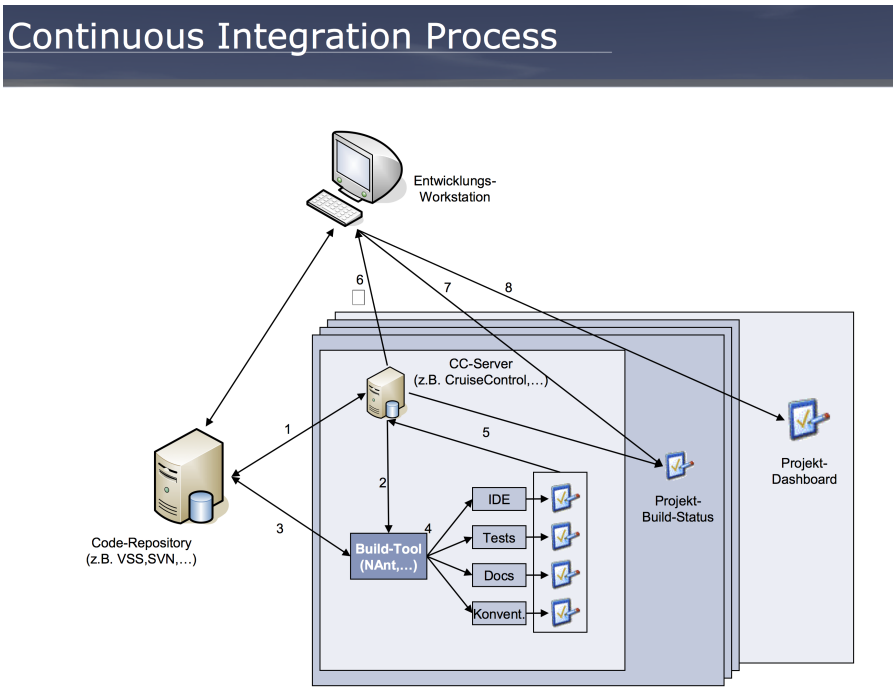

Abbildung 4.1: Continuous Build Prozess eines Softwareprojekts nach [PDEBD11]

4.0.10 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 vermutlich ein schwer zu findenden 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]

4.0.11 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

5 Versions-Management

6 Release Management

6.0.12 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. [Wik13a], [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. [Wik13b]

6.0.13 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]

6.0.14 Teilprozesse

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

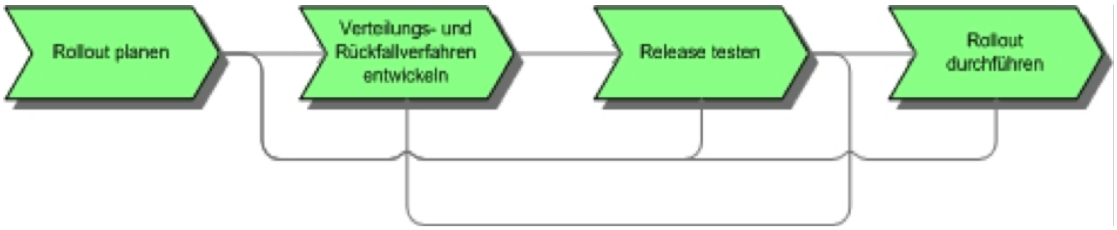

Abbildung 6.1: Teilprozesse des Release Management nach ITIL [Wik13a]

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. [Wik13a]

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. [Wik13a]

7 Change Management

8 Fazit

Lorem ipsum dolet.

Abkürzungsverzeichnis

etc.	et cetera
i.d.R.	in der Regel
SCM	Software Configuration Management
z. B.	zum Beispiel

Abbildungsverzeichnis

Literaturverzeichnis

[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

[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

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

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

[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

[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

[Wik13a] 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

[Wik13b] 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