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

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?

4.0.1 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 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 weniger häufig 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.

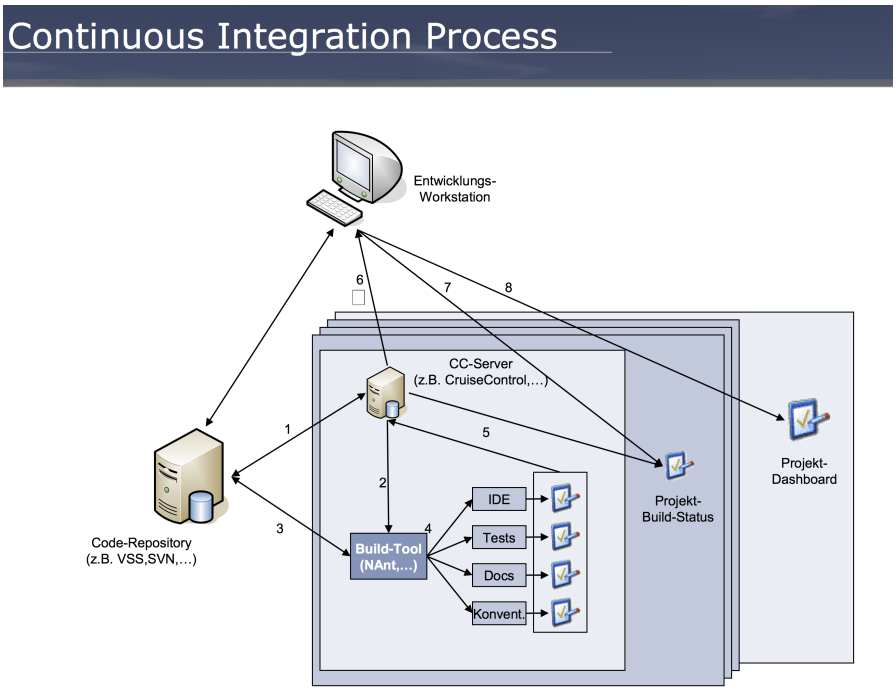

Abbildung 4.1: Continuous Build Prozess eines Softwareprojekts nach [PDEBD11]

4.0.2 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 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]

4.0.3 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

5 Versions-Management

6 Release Management

6.0.4 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.
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.

6.0.5 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.

6.0.6 Teilprozesse

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

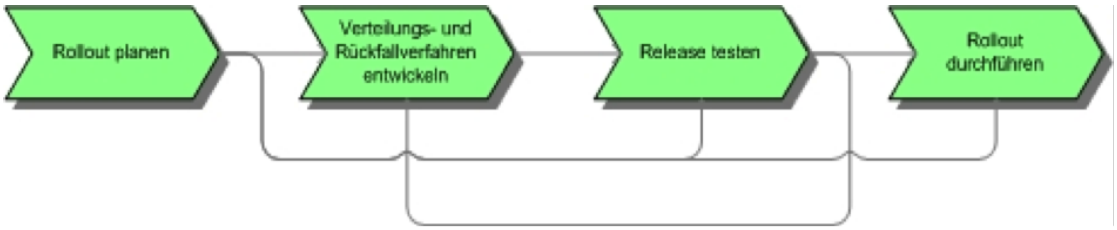

Abbildung 6.1: Teilprozesse des Release Management nach ITIL []

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

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. Im Rahmen des Release Prozesses sind drei Arten von Tests vorgesehen. Dies sind der Unit-, Integration- und der Abnahme-Test. 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 System-veränderungen 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. Andererseits findet eine Erfolgskontrolle statt.

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

[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

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