

Peter Lachenmaier
lachenma@in.tum.de

Hauptseminar: Management von Softwaresystemen

(TUM WS 2005/2006)

Konfigurations-, Build- und Releasemanagement



Technische Universität München
Fakultät für Informatik



Lehrstuhl IV: Software & Systems Engineering

Bearbeiter:

Peter Lachenmaier
lachenma@in.tum.de

Betreuer:

Tilman Seifert
seifert@in.tum.de

Inhalt:

<u>1. Einleitung.....</u>	<u>3</u>
<u>2. Konfigurationsmanagement.....</u>	<u>3</u>
<u>2.1 Definitionen und Begriffe.....</u>	<u>4</u>
<u>2.2 Die wichtigsten Konzepte.....</u>	<u>4</u>
<u>2.3 Herausforderungen und Probleme.....</u>	<u>6</u>
<u>2.4 Lösungsansätze.....</u>	<u>7</u>
<u>2.5 Organisation und Planung.....</u>	<u>8</u>
<u>2.5.1 Vorgehensmodell: V-Modell 97.....</u>	<u>8</u>
<u>2.5.2 Konfigurationsmanagement-Plan.....</u>	<u>11</u>
<u>2.6 Aktivitäten des Konfigurationsmanagement.....</u>	<u>13</u>
<u>2.7 Tool: CVS Concurrent Versions System.....</u>	<u>16</u>
<u>2.7.1 Checkin/Checkout.....</u>	<u>16</u>
<u>2.7.2 Änderungskonflikte.....</u>	<u>16</u>
<u>2.7.3 Tags.....</u>	<u>20</u>
<u>2.7.4 Branch/Merge.....</u>	<u>20</u>
<u>3. Buildmanagement.....</u>	<u>21</u>
<u>3.1 Anforderungen und Ziele.....</u>	<u>21</u>
<u>3.2 Prozessablauf.....</u>	<u>21</u>
<u>4. Releasemanagement.....</u>	<u>22</u>
<u>4.1 Release-Entscheidungen.....</u>	<u>23</u>
<u>4.2 Release-Erstellung.....</u>	<u>24</u>
<u>4.3 Release-Dokumentation.....</u>	<u>24</u>
<u>5. Diskussion/Zusammenhang.....</u>	<u>25</u>
<u>6. Zusammenfassung.....</u>	<u>26</u>
<u>7. Quellen/Literatur.....</u>	<u>27</u>

1. Einleitung

In dieser Ausarbeitung wird ein Überblick über Konfigurations-, Build- und Releasemanagement gegeben. Dabei wird auf die wichtigsten Konzepte des Konfigurationsmanagement eingegangen und gezeigt welche Herausforderungen und Probleme gelöst werden müssen. Die einzelnen durchzuführenden Aufgaben werden näher beschrieben und anhand von Beispielen verdeutlicht. Des weiteren wird unter dem Punkt Buildmanagement erläutert wie der Ablauf zum Erstellen einer Programmversion bzw. eines Release aussieht. Der Abschnitt Releasemanagement zeigt die Verwaltung und Planung einzelner freigegebener Versionen eines Softwaresystems.

2. Konfigurationsmanagement

Folgende Definition wurde vom ANSI (American National Standardization Institute) in Zusammenarbeit mit der EIA (Electronic Industries Alliance) veröffentlicht (vgl.: [\[ans2005\]](#)):

- ***„Configuration Management ... is a management process for establishing and maintaining consistency of a product's performance, its functional and physical attributes, with its requirements, design and operational information, throughout its life.“***

Die Übereinstimmung des Produkts mit den definierten Anforderungen des Kunden und den Entwicklungsanforderungen steht im Mittelpunkt. Diese Anforderungen werden schon während der Entwicklung, speziell aber beim fertigen Produkt und während dessen gesamter Laufzeit überprüft und verglichen.

Die „International Organization for Standardization“ (ISO) beschreibt die Ziele des Konfigurationsmanagement folgendermaßen:

- ***„Hauptziel von Konfigurationsmanagement ist, die gegenwärtige Konfiguration eines Produkts sowie den Stand der Erfüllung seiner physischen und funktionellen Forderungen zu dokumentieren und volle Transparenz herzustellen. Ein weiteres Ziel ist, dass jeder am Projekt Mitwirkende zu jeder Zeit des Produktlebenslaufs die richtige und zutreffende Dokumentation verwendet.“ (ISO 10007)***

Hieraus lassen sich einige Schwerpunkte des Konfigurationsmanagement ableiten. Die wichtigsten Konzepte werden unter Punkt [2.2. Die wichtigsten Konzepte](#) vorgestellt.

Entstanden ist das Konfigurationsmanagement in seiner ursprünglichen Form in den fünfziger Jahren bei Projekten der amerikanischen Raumfahrtindustrie. In der damaligen Zeit wurden während der Entwicklung von Raumfahrzeugen viele undokumentierte Änderungen durchgeführt. Dadurch, dass die Raumfahrzeuge bei ihrem Test im Normalfall zerstört wurden, waren die Hersteller nach einem erfolgreichen Test nicht in der Lage eine Serienproduktion anzufangen, oder ein entsprechendes Modell nachzubauen. Die Pläne waren nicht mehr auf dem aktuellen Stand und der Prototyp mit all seinen Änderungen war vernichtet. Um diesen Informationsverlust in Zukunft zu vermeiden, wurden Konfigurationsmanagementmechanismen entwickelt (vgl.: [\[Bal1998\]](#)).

2.1 Definitionen und Begriffe

Bevor näher auf die Details des Konfigurationsmanagements eingegangen wird, sollten zunächst verschiedene Begriffe geklärt werden.

- **Konfiguration**
- **Konfigurationseinheit**
- **Bezugskonfiguration**

Konfiguration: (engl.: „configuration“) Unter einer Konfiguration versteht man im Wesentlichen den Entwicklungsstand eines Softwareprodukts. Dazu gehören aber nicht nur der Quellcode und die Dokumentation, sondern in gleichen Maßen die Werkzeuge und Hilfsmittel, die zum Erstellen einer lauffähigen Version benötigt werden. Ebenso sind auch die Dokumentation und die Einstellungen der Werkzeuge und Hilfsmittel enthalten.

Konfigurationseinheit: (engl.: „configuration item“) Eine Konfigurationseinheit ist ein Teil einer Konfiguration, das vom Konfigurationsmanagement verwaltet wird. Das können unter anderem Quellcode, Dokumentationsdateien oder ein Buildfile sein.

Bezugskonfiguration: (engl.: „baseline“) Eine Bezugs- oder Referenzkonfiguration ist ein zu einem bestimmten Zeitpunkt ausgewähltes und freigegebenes Zwischenergebnis, das zu einer Konfiguration zusammengefasst wird, um sich später darauf beziehen zu können.

2.2 Die wichtigsten Konzepte

Das Konfigurationsmanagement beinhaltet verschiedene wichtige Konzepte:

- **Identifikation**
- **Verfügbarkeit vorangegangener Versionen**
- **Unterstützung von paralleler Arbeit**
- **Parallele Entwicklungszweige**
- **Verfolgung, Nachvollziehbarkeit und Dokumentation von Änderungen**

Identifikation: Bei der Erstellung von Softwaresystemen entstehen oft tausende von Dateien. Um den Überblick, über die einzelnen Konfigurationseinheiten und ihrer Abhängigkeiten zueinander, zu gewährleisten, muss die eindeutige Identifizierung aller einzelnen Elemente vom Konfigurationsmanagement bewerkstelligt werden. Unter Absatz [2.6 Aufgaben des Konfigurationsmanagement](#) wird genauer auf die Vorgehensweise eingegangen und ein Schema anhand eines Beispiels verdeutlicht.

Verfügbarkeit vorangegangener Versionen: In der Softwareentwicklung kann es notwendig sein, auf vorangegangene Versionen verschiedener Konfigurationselemente zurückgreifen zu müssen. Um beispielsweise das vor einem Jahr ausgelieferte System nachzubauen, braucht man exakt den Stand der damaligen „Source-Files“ und die verwendeten Werkzeuge. Ein Versionsmanagementsystem (vgl.: [2.7 Tool: CVS Concurrent Versions System](#)) unterstützt dieses Konzept.

Unterstützung von paralleler Arbeit: Um Teamarbeit in vollem Umfang zu ermöglichen ist es wichtig, dass mehrere Mitarbeiter gleichzeitig an den gleichen Dateien arbeiten können. Die Verwaltung kann ebenfalls das Versionsmanagementsystem (vgl.: [2.7 Tool: CVS Concurrent Versions System](#)) übernehmen.

Parallele Entwicklungswege: Das Konfigurationsmanagement muss die parallele Entwicklung an verschiedenen Varianten eines Softwareprodukts verwalten. In der Softwareentwicklung stellen sich Situationen ein, an denen eine geradlinige Weiterentwicklung nicht mehr möglich, oder aber eine spezielle Verzweigung erwünscht ist. Man arbeitet beispielsweise an einer Weiterentwicklung der Version 1.0. Der Kunde äußert währenddessen seine Änderungswünsche, welche am besten möglichst schnell umgesetzt werden sollen. Ausgehend von der Version 1.0 wird also parallel zum bereits bestehenden Entwicklungszweig ein weiterer Zweig eröffnet (Branch). Nach Fertigstellung des nächsten Release können die parallel erzeugten Änderungen bzw. Weiterentwicklungen wieder in den Hauptzweig eingearbeitet werden. Die beiden Entwicklungszweige vereinigen sich also wieder zu einem (Merge). [Abbildung 2.2-1](#) veranschaulicht diesen Prozess graphisch.

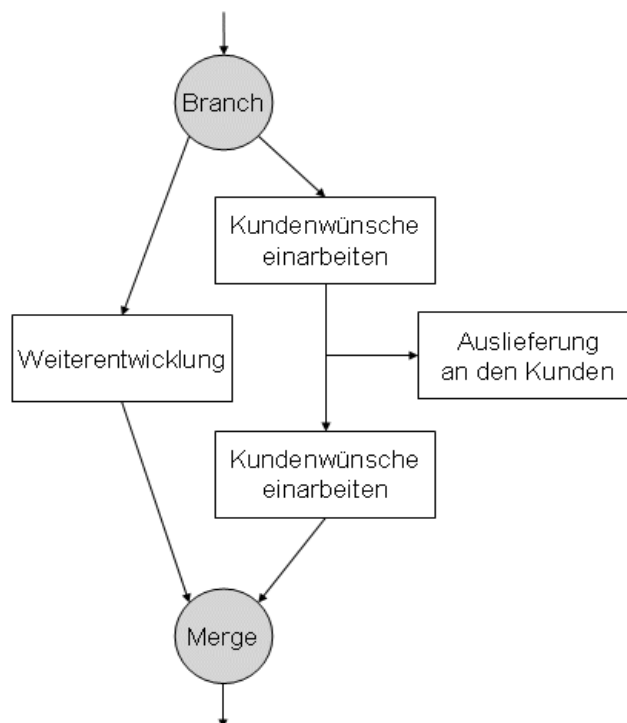


Abb.: 2.2-1 Quelle: [sok2005]

Verfolgung, Nachvollziehbarkeit und Dokumentation von Änderungen: Im Laufe eines Softwareentwicklungsprozesses wird eine Schar von Änderungen, beispielsweise durch entdeckte Fehler oder durch neue Anforderungen, durchgeführt. Das Konfigurationsmanagement muss sicherstellen, dass Änderungen mitsamt des Grundes, des Zeitpunktes und der Verantwortlichen dokumentiert werden. Unter *Konfigurationsüberwachung/Änderungsmanagement* im Punkt [2.6 Aufgaben des Konfigurationsmanagement](#) wird näher auf die dazu notwendigen Tätigkeiten eingegangen.

2.3 Herausforderungen und Probleme

Bei der Umsetzung dieser Konzepte wird man vor einige Herausforderungen und Probleme gestellt:

- *Behandlung von parallelen Änderungen*
- *Zusammenführen verschiedener Entwicklungszweige*
- *Speichern von allen Versionen aller Konfigurationselemente*

Behandlung von parallelen Änderungen: Ein wichtiges Konzept ist, dass mehrere Benutzer parallel an den gleichen Dateien arbeiten können. Dabei stellt sich die Frage, wie man die Änderungen der verschiedenen Benutzer verwaltet, denn es dürfen keine Änderungen verloren gehen. Wenn beispielsweise zwei Benutzer zeitgleich eine Datei bearbeiten, darf das Speichern der Änderungen des zweiten Benutzers nicht die des Ersten überschreiben. Nach Ende der Bearbeitung dürfen auch nicht zwei verschiedene Varianten einer Datei im System vorhanden sein.

Zusammenführen verschiedener Entwicklungszweige: Wie oben beschrieben, ist bei der Entwicklung umfangreicher Softwareprodukte die Unterstützung paralleler Entwicklungszweige von großer Bedeutung. Man verlangt, dass das Trennen und Zusammenführen der parallelen Entwicklungszweige unterstützt wird. Das Zusammenführen von ganzen Entwicklungszweigen stellt sich schwerer dar als die Behandlung von parallelen Änderungen an einzelnen Dateien. In den verschiedenen Varianten können mehrere Dateien hinzugefügt oder entfernt worden sein. Die Änderungen in den einzelnen Zweigen können sehr umfangreich sein und sich über viele Dateien erstrecken.

Speichern von allen Versionen aller Konfigurationselemente: Da man im Laufe des Entwicklungsprozesses oft auf alte Versionen zurückgreifen will, müssen alle Versionen der einzelnen Konfigurationselemente gespeichert werden. Man könnte dazu nach einer Änderung die jeweilige Datei, mit einer Kennzeichnung im Dateinamen, abspeichern. Dieses Vorgehen führt schon bei kleinen Projekten zu Unübersichtlichkeit und zu einem großen Speicherbedarf.

2.4 Lösungsansätze

Für die oben beschriebenen Herausforderungen und Probleme existieren verschiedene Lösungsansätze:

- *Sperrkonzept/Automatische Mischung*
- *Merge von Hand*
- *Speichern der Unterschiede (Deltaspeicherung)*

Sperrkonzept/Automatische Mischung: Um zu vermeiden, dass sich die Änderungen von verschiedenen Benutzern an den gleichen Dateien, wie oben gezeigt, gegenseitig beeinflussen, kann man im Wesentlichen zwei verschiedene Ansätze verfolgen. Man kann ein „pessimistisches“ Sperrkonzept oder ein „optimistisches“ Verfahren mit automatischer Mischung anwenden. Beide verlangen die Unterstützung durch ein spezielles Tool.

Bei der Strategie mit den Sperrungen wird durch das Tool verhindert, dass ein Entwickler eine Datei bearbeitet, die bereits von jemand Anderem benutzt wird. Erst wenn die Sperre durch den Entwickler wieder aufgehoben wird, kann der nächste seine Änderungen durchführen. Dadurch wird allerdings die parallele Arbeit insofern eingeschränkt, dass manche Benutzer selbst für kleine Änderungen, auch wenn diese nicht mit den Änderungen des aktuellen Bearbeiters konkurrieren, warten müssen, bis die Datei wieder freigegeben wird.

Das optimistischere Konzept mit automatischen Mischverfahren unterstützt die parallele Arbeit besser. Jeder Benutzer kann hierbei Änderungen an den verschiedenen Dateien vornehmen. Wenn die Änderungen an verschiedenen Stellen der Datei passieren, können die beiden Dateien durch ein spezielles Mischverfahren automatisch zu einer Version mit allen Änderungen zusammengeführt werden. Unter [2.7 Tool: CVS Concurrent Versions System](#) werden die Mechanismen, die dazu notwendig sind genauer beleuchtet und die verschiedenen Konflikte, die beim „Merging“ auftreten können, näher betrachtet.

Merge von Hand: Die Zusammenführung von ganzen Entwicklungszweigen erfordert aktiv das Eingreifen der beteiligten Entwickler. Dabei ist die gute Kommunikation zwischen den beteiligten Entwicklern wichtig. Es ist zu entscheiden welche Teile aus den einzelnen Zweigen in die endgültige Version einfließen sollen. Die Teile aus den verschiedenen Zweigen müssen von den Bearbeitern zu einem konsistenten Gesamtsystem zusammengefügt werden. Die Werkzeugunterstützung beschränkt sich hierbei auf die Behandlung von Änderungen an den gleichen Dateien der einzelnen Zweige. Die durch das verwendete Tool automatisierten Tätigkeiten sind von den Entwicklern zu überprüfen.

Speichern der Unterschiede (Deltaspeicherung): Um alle Versionen einer Datei übersichtlich und ohne hohen Speicherbedarf zu speichern, bietet sich die Verwendung eines Versionsverwaltungstools (vgl.: [2.7 Tool: CVS Concurrent Versions System](#)) an. Bei diesen Systemen werden ausgehend von einer Version der Datei, beispielsweise nur die Änderungen gespeichert, die zu der jeweiligen Version geführt haben. Man kann daraus alle bisherigen Versionen erzeugen. Solch ein Verfahren, bei dem nur die Unterschiede zwischen den verschiedenen Dateiversionen gespeichert werden, nennt man „Deltaspeicherung“. Dadurch kann man den Speicherbedarf enorm senken. Versionsverwaltungssysteme wie CVS bieten hinzu die Möglichkeit, Kommentare zu den jeweiligen Änderungen zu speichern, wodurch die Übersichtlichkeit und Rückverfolgung der Änderungen verbessert wird.

2.5 Organisation und Planung

Für das erfolgreiche Durchführen von Konfigurationsmanagement ist ein strategisches Vorgehen nötig. Es existieren verschiedene Vorgehensmodelle, die den Ablauf gezielt aufteilen und beschreiben. Hier wird kurz auf das V-Modell 97 (vgl.: [2.5.1 Vorgehensmodell: V-Modell 97](#)) eingegangen. Ein wichtiger Bestandteil der Planung ist die Erstellung des Konfigurationsmanagement-Plans. Unter [2.5.2 Konfigurationsmanagement-Plan](#) wird der Inhalt dieses Plans gezeigt.

2.5.1 Vorgehensmodell: V-Modell 97

Hier wird ein kurzer Einblick in das V-Modell 97 (vgl.: [V-Modell: \[v972005\]](#)) gegeben. Seit Februar 2005 ist auch eine neue Version, das V-Modell-XT (vgl.: [V-Modell-XT: \[vxt2005\]](#)) veröffentlicht. Beim V-Modell handelt es sich um ein „Vorgehensmodell zur Planung und Durchführung von IT-Vorhaben“. Es ist der „Entwicklungsstandard für IT-Systeme des Bundes“.

Im V-Modell wird zwischen vier Submodellen unterschieden:

- **Systemerstellung**
- **Qualitätssicherung**
- **Konfigurationsmanagement**
- **Projektmanagement**

Diese vier Bereiche stehen in enger Beziehung zueinander. Im Folgenden wird nur auf das hier relevante Submodell „Konfigurationsmanagement“ eingegangen. Dieses Submodell wird im V-Modell folgendermaßen definiert:

- *„Das Submodell Konfigurationsmanagement stellt sicher, dass Produkte eindeutig identifizierbar sind, Zusammenhänge und Unterschiede von verschiedenen Versionen einer Konfiguration erkennbar bleiben und Produktänderungen nur kontrolliert durchgeführt werden können“*

Es untergliedert sich in folgende vier Bereiche:

- *Konfigurationsmanagement-Planung (KM1)*
- *Produkt- und Konfigurationsverwaltung (KM2)*
- *Änderungsmanagement – auch Konfigurationssteuerung genannt (KM3) und*
- *Konfigurationsmanagement-Dienste (KM4)*

[Abbildung 2.5.1-1](#) gibt hierzu einen Funktionsüberblick.

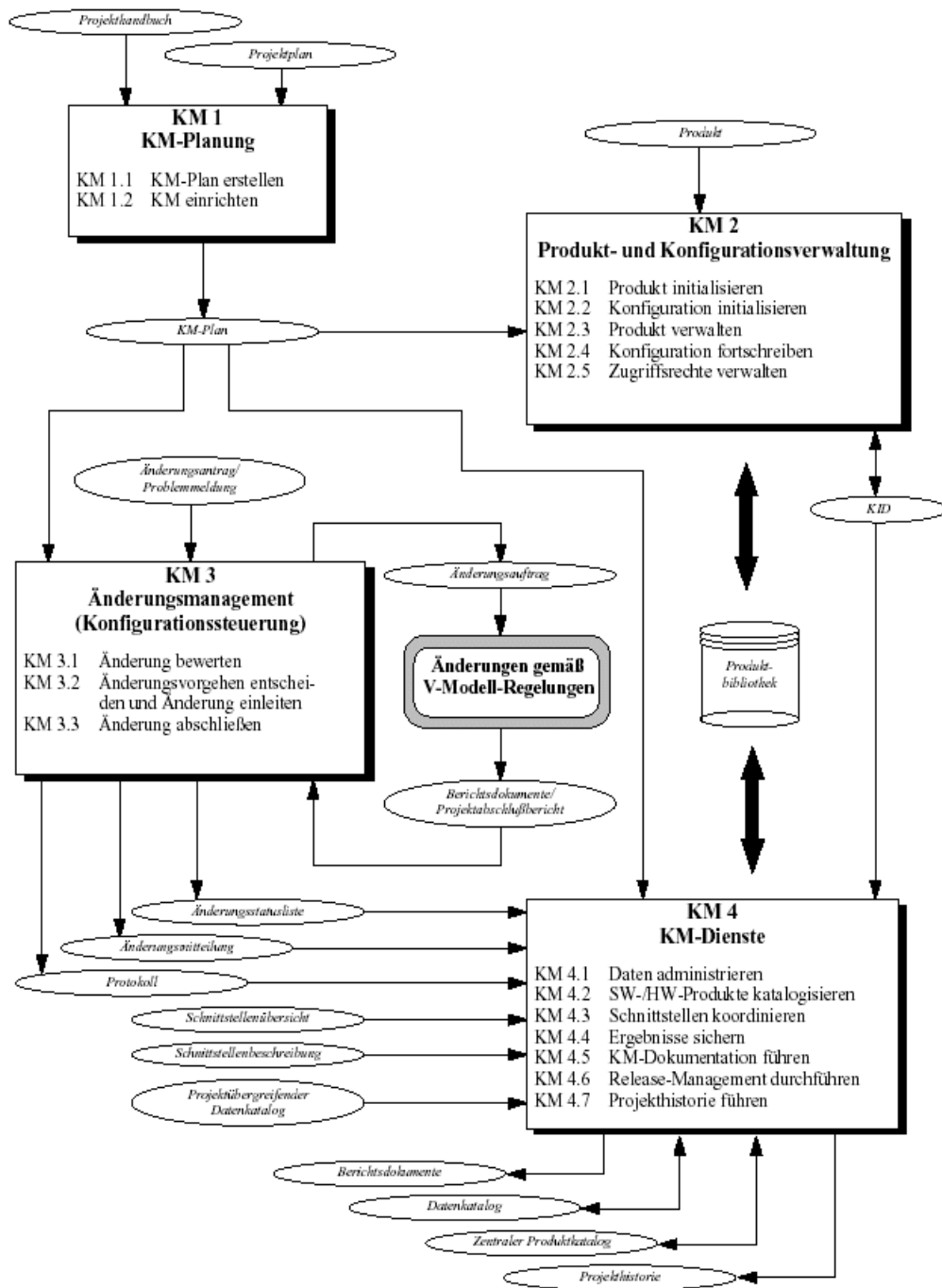


Abb.: 2.5.1-1 Quelle: [sok2005]

Die in diesen Bereichen zu bearbeitenden Produkte unterliegen im V-Modell einer definierten Abfolge von Zuständen und Zustandsübergängen. [Abbildung 2.5.1-2](#) stellt diese dar. Die Pfeile symbolisieren hierbei die möglichen Übergänge und die Kreise symbolisieren die Zustände.

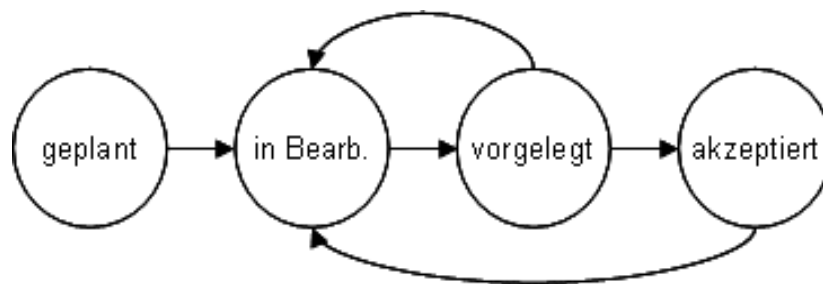


Abb.: 2.5.1-2 Quelle: [sok2005]

2.5.2 Konfigurationsmanagement-Plan

Im Konfigurationsmanagement-Plan werden alle organisatorischen und verfahrenstechnischen Details des Konfigurationsmanagements festgelegt. [Abbildung 2.5.2-1](#) zeigt den Aufbau und die Inhalte eines Konfigurationsmanagement-Plans aus [Bal1998]. Balzert verwendet dabei eine gekürzte und leicht modifizierte Version aus dem V-Modell 97.

2. Einführung des Konfigurationsmanagements (KM)

2.1 Produktbibliothek

Hier ist die Produktbibliothek mit ihrer Systematik der Produkt-, Nutzer-, Rechte- und Relationsverwaltung zu beschreiben, bzw. auf geeignete Dokumente oder Handbücher zu verweisen.

2.2 Projektspezifische Festlegungen

Dieser Gliederungspunkt legt fest,

- welche Produkte unter KM-Kontrolle gestellt werden
- anhand welcher Kriterien die Konfigurationseinheiten gebildet werden,
- welche Zustände Produkte durchlaufen,
- welche Produktattribute mitgeführt werden,
- wie Zugriffsrechte vergeben und kontrolliert werden,
- wie Produkte von Unterauftragnehmern unter KM-Kontrolle gestellt werden,
- welche sonstigen Objekte (Entwicklungsrechner, Werkzeuge, Prüfumgebung usw.) unter KM-Kontrolle gestellt werden,
- wie KM-Instanzen (z.B. Change Control Board) eingeführt und besetzt werden,
- welche Hilfsmittel (z.B. KM-Werkzeuge, Formblätter) bereitgestellt werden.

Zu Produktattribute:

Es wird festgeschrieben, welche Produktattribute geführt werden, welchen Initialwert sie besitzen und wie mögliche Wertfolgen der Attribute aussehen können.

Beispiele für Produktattribute sind:

- Zustand
- Eigentümer
- Bearbeiter
- Entstehungsdatum
- Änderungshistorie
- Vorgegebene Bearbeitungsdauer
- Zugriffsrecht
- Klassifizierung (Vertraulichkeitsvermerke)
- Schlüsselbegriffe (dienen zum Wiederauffinden)
- Zugehörige andere Produkte (Hier kann eine Verkettung aller zu einem Modul/ einer Komponente gehörenden Produkte erfolgen)
- Versionsattribute (Ist das gegenwärtige Produkt eine Version eines Vorgängers, so werden die Versionsattribute eingetragen. Jede Änderung an dem Produkt, die mit einer Zustandsänderung verknüpft ist, bedeutet automatisch eine Änderung der Versionsbezeichnung (normalerweise Versionserhöhung)).

2.3 Konventionen zur Identifikation

3. Änderungsmanagement

3.1 Änderungsprozedur

3.1.1 Verfahren im Änderungswesen

3.1.2 Änderungsaufträge

3.1.3 Schnittstellen zum Auftraggeber und zu anderen Stellen

3.1.4 Änderungskontrolle und Statusanzeige

3.2 Formulare des Änderungswesens und deren Handhabung

3.3 Versionskontrolle

3.4 Dokumente des Konfigurationsmanagements

4. Sicherung und Archivierung

Abb.: 2.5.2-1 Quelle: [Bal1998]

2.6 Aktivitäten des Konfigurationsmanagement

Die bisherigen Abschnitte haben sich mit den grundlegenden Konzepten des Konfigurationsmanagement befasst. Im Folgenden wird nun näher auf die Aktivitäten eingegangen, die notwendig sind um die Transparenz und Nachvollziehbarkeit des Projekts zu gewährleisten. Diese sind von Vorgehensmodell zu Vorgehensmodell verschieden, jedoch lassen sich in allen folgende vier grundlegende Punkte wiederfinden (vgl.: [\[sok2005\]](#)):

- **Konfigurationsidentifizierung**
- **Konfigurationsüberwachung/Änderungsmanagement**
- **Konfigurationsbuchführung**
- **Konfigurationsüberprüfung/Konfigurationsaudit**

Konfigurationsidentifizierung: Zur eindeutigen Identifizierung aller Konfigurationseinheiten (vgl.: [2.1 Definitionen und Begriffe](#)) wird eine entsprechende Konvention benötigt. Durch die eindeutige Identifizierung soll erreicht werden, dass jederzeit alle Ergebnisse und somit auch der aktuelle Stand des Projekts ersichtlich sind. Dadurch lassen sich auch die Abhängigkeiten verschiedener Konfigurationselemente zueinander darstellen. Dies ist für die Erstellung eines stabilen Produkts Voraussetzung, selbst wenn dieses parallel zur Auslieferung weiterentwickelt wird (vgl.: [2.7.4 Branch/Merge](#)).

Am Anfang des Projekts muss festgelegt werden, was als Konfigurationseinheit betrachtet wird. Außerdem ist ein Benennungsschema zu bestimmen, das die Konfigurationseinheiten bezeichnet. Das folgende Beispiel zeigt hierzu einen Ansatz aus [\[Som2001\]](#). Das Benennungsschema kann die Beziehungen zwischen den einzelnen Konfigurationseinheiten protokollieren. Dies kann zum Beispiel in einem hierarchischen Namensschema über den gleichen Namensstamm bei verwandten Elementen erfolgen:

- **PCL-TOOLS/EDIT/FORMS/DISPLAY/AST-INTERFACE/CODE**
- **PCL-TOOLS/EDIT/HELP/QUERY/HELPFRAMES/FR-1**

Dabei handelt es sich beim ersten Bestandteil des Namens um den Projektnamen, „PCL-TOOLS“. Dieses Projekt umfasst vier separate Werkzeuge. Der Werkzeugname wird als zweiter Teil des Namens verwendet. Jedes Werkzeug besteht aus verschiedenen mit einem Namen versehenen Modulen. Dieser Unterteilungsprozess wird so lange fortgeführt, bis die einzelnen formalen Dokumente benannt werden (vgl.: [Abb.: 2.6-1](#)). Die Blätter der Hierarchie bilden die formalen Projektdokumente. [Abbildung 2.6-1](#) zeigt, dass drei formale Dokumente für jedes der verwalteten Objekte benötigt werden. Hierbei handelt es sich um die Objektbeschreibung (OBJECTS), den Programmcode (CODE) und eine Reihe von Tests für diesen Code (TESTS).

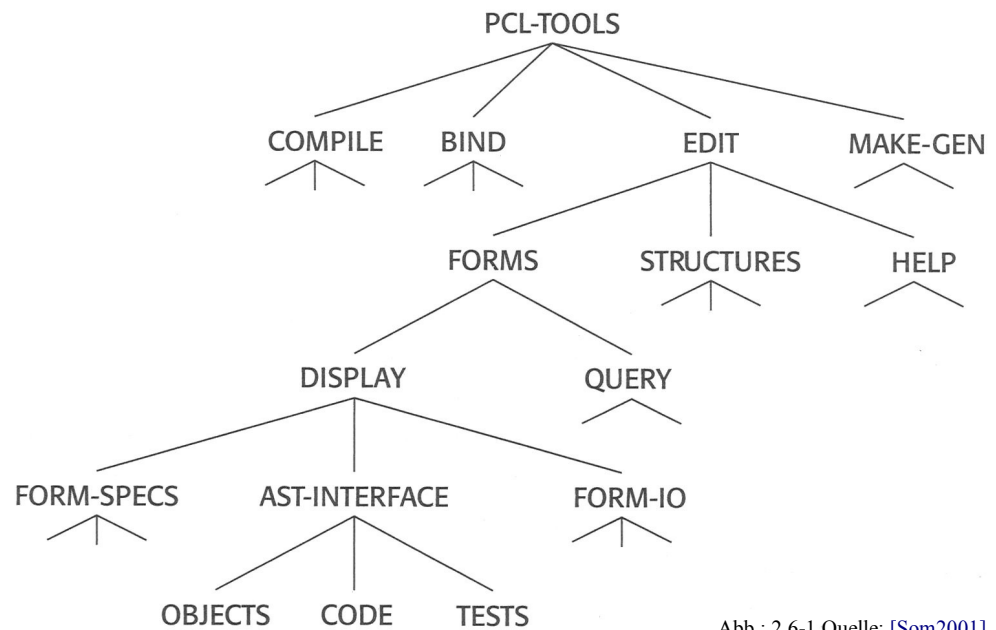


Abb.: 2.6-1 Quelle: [Som2001]

In großen Softwarefirmen werden in vielen Projekten eine Reihe von gleichen Komponenten verwendet. Dies können Teile sein, die bereits im Rahmen eines anderen Projekts entwickelt wurden, oder auch Komponenten von Drittanbietern. Hierbei kann es sich unter anderem um Bibliotheken oder verschiedene Tools handeln. Um Redundanz zu vermeiden und sicherzustellen, dass in allen Projekten die gleichen (neuesten) Versionen verwendet werden, sollten die wiederverwendbaren Komponenten aus dem projektspezifischen Namensschema herausgenommen werden. Man kann sie beispielsweise nach ihrem Anwendungsgebiet in ein eigenes Schema einordnen. Auch diese Komponenten müssen der Versionskontrolle unterliegen.

Konfigurationsüberwachung/Änderungsmanagement: Die Hauptaufgabe der Konfigurationsüberwachung ist das Änderungsmanagement. Außerdem müssen in diesem Punkt die Verantwortlichkeiten in der Vergabe und Kontrolle von Sonderfreigaben geregelt werden. Sonderfreigaben sind Freigaben von Teilen der Konfiguration, die (noch) nicht den Anforderungen entsprechen. Eine solche Sonderfreigabe kann erfolgen, um den Fortschritt des Entwicklungsprozesses nicht zu gefährden. Sonderfreigaben sollten auf einen bestimmten Zeitraum begrenzt erfolgen.

Änderungen können sich zum einen durch geänderte Anforderungen des Kunden oder durch im Test aufgedeckte Fehler ergeben. Sämtliche Änderungen müssen dokumentiert und begründet werden. Bevor allerdings Änderungen durchgeführt werden, ist die Änderungsanforderung bezüglich Nutzen und Risiken zu bewerten. Dies führt zur Genehmigung oder Ablehnung der Änderung. Nach Durchführung der Änderung muss diese verifiziert werden (vgl.: [Abb.: 2.6-2](#)).

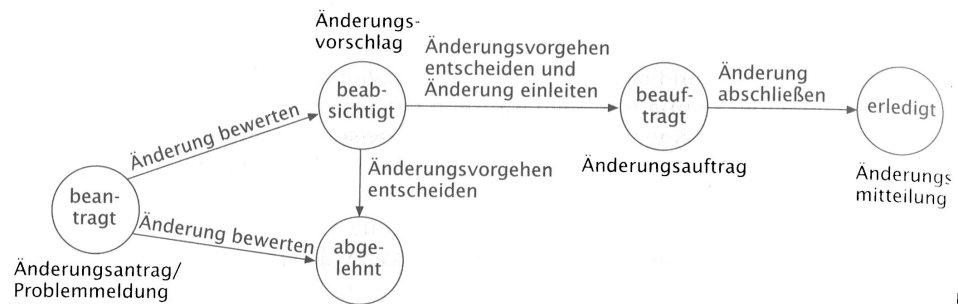


Abb.: 2.6-2 Quelle: [Bal1998]

Konfigurationsbuchführung: Ziel ist es, die Rückverfolgbarkeit der Änderungen sicherzustellen. Die alleinige Verwendung eines Versionsverwaltungstools (vgl.: [2.7. Tool: CVS Concurrent Versions System](#)), das alle Produkte beinhaltet, reicht für diese Aufgabe nicht aus. Es sind zu allen

- **Konfigurationseinheiten**
- **Änderungen (an Konfigurationseinheiten, Bezugskonfigurationen, ...)**
- **Sonderfreigaben (an Konfigurationseinheiten, Bezugskonfigurationen, ...) und**
- **Bezugskonfigurationen**

Informationen abzulegen, die Transparenz ermöglichen. Dies können z. B. Identifizierungsschlüssel oder eine Historie für Änderungen mit Freigabe- und Einarbeitungsstatus sowie Begründungen für Änderungen sein. Diese Informationen wurden zum Teil bereits bei der *Konfigurationsüberwachung* generiert und müssen eventuell übersichtlicher aufgearbeitet werden.

Die Konfigurationsbuchführung sollte generell als „Nebenprodukt“ der täglichen Arbeit geschehen.

Konfigurationsüberprüfung/Konfigurationsaudit: Um sicherzustellen, dass das Produkt den vereinbarten Anforderungen entspricht und dass die Konfigurationsdokumente das Produkt richtig darstellen, ist die Konfiguration des Systems mittels Audits zu überprüfen.

Konfigurationsaudits kommen hierbei hauptsächlich bei der Definition von Bezugskonfigurationen zum Einsatz. Dabei wird überprüft, ob die Konfiguration der Dokumentation und den Anforderungen entspricht. Andernfalls muss ein Änderungsprozess an den betroffenen Konfigurationseinheiten eingeleitet werden. So muss die Konfiguration von der korrekten Umsetzung in einer Programmiersprache, bis hin zur Stimmigkeit mit den vom Kunden gestellten Anforderungen verifiziert werden.

2.7 Tool: CVS Concurrent Versions System

Einen wichtigen Beitrag zum erfolgreichen Konfigurationsmanagement leistet die Versionsverwaltung. Sie muss dabei u. a. die Möglichkeit unterstützen, jederzeit auf sämtliche Versionen aller Konfigurationseinheiten zurückgreifen zu können. Die Änderungen zwischen den einzelnen Versionen sollten dokumentiert werden können.

CVS ist ein solches Versionsverwaltungssystem. Es ist bereits in den 80er Jahren entstanden und erfreut sich heute großer Verbreitung. Speziell bei Open Source Projekten wird es bevorzugt eingesetzt. CVS ist selbst Open Source und im Internet frei erhältlich. Es ist ein reines Kommandozeilen Tool, mittlerweile existieren jedoch zahlreiche graphische Oberflächen. In verschiedenen Entwicklungsumgebungen ist ein CVS-Client bereits eingebunden. Seine große Verbreitung verdankt es auch der Tatsache, dass es für viele verschiedene Plattformen und Betriebssysteme verfügbar ist. CVS kann als lokale Installation oder als Client/Server-Anwendung genutzt werden. Die Teamarbeit wird durch die Client/Server-Version wesentlich besser unterstützt.

Mit dem Concurrent Versions System ist es dem Nutzer möglich, auf alle Versionen einer Datei im Repository zuzugreifen. Gespeichert wird dabei immer nur die aktuellste Version und die Änderungen, die zur jeweiligen Version geführt haben. Das eben genannte Repository ist das Verzeichnis, in dem der gesamte Datenbestand gehalten wird. Zu diesem Repository können von verschiedenen Usern Dateien hinzugefügt werden.

2.7.1 Checkin/Checkout

Zum Bearbeiten einer bestimmten Datei kann eine Kopie in ein lokales Verzeichnis geholt werden (Auschecken/Checkout). Nach dem Bearbeiten können die Änderungen in das Repository zurückgeschrieben werden (Einchecken/Checkin). Bei jedem Einchecken einer geänderten Datei wird die Versionsnummer dieser Datei von CVS erhöht (vgl.: [Abb.: 2.7.1-1](#)).

```
+-----+   +-----+   +-----+   +-----+   +-----+
! 1.1 !----! 1.2 !----! 1.3 !----! 1.4 !----! 1.5 !
+-----+   +-----+   +-----+   +-----+   +-----+
```

Abb.: 2.7.1-1 Quelle: [Ced2005]

2.7.2 Änderungskonflikte

CVS erlaubt es, dass eine Datei gleichzeitig von mehreren Benutzern bearbeitet wird. Erst beim Zurückschreiben in das Repository müssen evtl. entstandene Konflikte gelöst werden.

Will der Benutzer eine Datei in das Repository zurückschreiben und die Datei wurde bereits von einem anderen Benutzer mit Änderungen zurückgeschrieben, so wird er vom System benachrichtigt. Der Benutzer wird aufgefordert, seine lokale Kopie zu aktualisieren. Beim Aktualisieren versucht CVS die Änderungen an der lokalen Datei und der Datei im Repository zusammenzuführen. CVS benutzt hierzu einen so genannten „Drei

Wege Merge“. Beim „Drei Wege Merge“ werden sowohl die beiden Dateien mit den Änderungen, wie auch die ursprüngliche Datei miteinbezogen. Dabei können die Änderungen entweder automatisch von CVS zusammengeführt werden, oder es entstehen Konflikte, die vom Benutzer von Hand aufgelöst werden müssen.

Wurde die Datei an verschiedenen Bereichen verändert, so ist CVS in der Lage, die Änderungen automatisch in eine Datei zu übernehmen. Dabei können allerdings logische, semantische oder syntaktische Fehler auftreten. Das folgende Beispiel zeigt, wie ein logischer Fehler vom CVS System erzeugt werden kann.

Am Anfang befindet sich ein logischer Fehler im Code. Die Funktion `umfang()` verlangt den Durchmesser, um den Umfang eines Kreises zu berechnen. Die `main()` Funktion ruft die Umfangsfunktion allerdings mit dem Radius auf und liefert somit das falsche Resultat. Nutzer 1 und Nutzer 2 bemerken beide den Fehler und versuchen ihn zu beheben. Nutzer 1 denkt, dass die Umfangsfunktion korrekt ist und ändert deswegen den Aufruf in `main()`. Nutzer 2 denkt, dass die Funktion `umfang()` falsch implementiert wurde und ändert deshalb diese ab (vgl.: [Abb.: 2.7.2-1](#)). Beide Programme liefern nun das gewünschte Ergebnis. Nutzer 1 checkt nun seine Version ins Repository ein. Nutzer 2 will danach seine Version in das Repository einchecken und wird von CVS benachrichtigt, dass die Datei bereits verändert wurde. Er führt nun ein Update aus um seine lokale Kopie auf den neuesten Stand zu bringen. CVS kann dabei alle Änderungen automatisch übernehmen und meldet keine Konflikte, da sich die Änderungen von Nutzer 1 und Nutzer 2 an verschiedenen Stelle der Datei befinden. Nutzer 2 kann nun die Datei mit allen Änderungen in das Repository einchecken. Wie in [Abbildung 2.7.2-1](#) zu sehen ist, tritt allerdings wieder ein logischer Fehler auf. CVS stellt somit nicht sicher, dass wenn der Merge ohne Konflikte abläuft, die entstandene Datei auch fehlerfrei ist.

Ausgangsversion mit Fehler:

```
int main()
{
    float um;
    float radius = 1.0;
    um = umfang(radius);
    printf("\n\nUmfang: %lf\n\n",um);
}

// Funktion umfang()

float umfang(float durchmesser)
{
    return (durchmesser * 3.14);
}
```

Änderung Nutzer 1:

```
int main()
{
    float um;
    float durchmesser = 2.0;
    um = umfang(durchmesser);
    printf("\n\nUmfang: %lf\n\n",um);
}

// Funktion umfang()

float umfang(float durchmesser)
{
    return (durchmesser * 3.14);
}
```

Änderung Nutzer 2:

```
int main()
{
    float um;
    float radius = 1.0;
    um = umfang(radius);
    printf("\n\nUmfang: %lf\n\n",um);
}

// Funktion umfang()

float umfang(float radius)
{
    return (2 * radius * 3.14);
}
```

Gemergte Version:

```
int main()
{
    float um;
    float durchmesser = 2.0;
    um = umfang(durchmesser);
    printf("\n\nUmfang: %lf\n\n",um);
}

// Funktion umfang()

float umfang(float radius)
{
    return (2 * radius * 3.14);
}
```

Abb.: 2.7.2-1

Überschneiden sich Änderungen verschiedener Benutzer, so können diese nicht automatisch verschmolzen werden. Der Benutzer, der später versucht seine Arbeit einzuchecken, wird vom System aufgefordert die Konflikte per Hand zu beheben, bevor der Checkin-Vorgang durchgeführt werden kann. CVS kann dazu beide Änderungen, speziell gekennzeichnet, in eine gemeinsame Datei einfügen (vgl.: [Abb.: 2.7.2-2](#)). Nachdem der Benutzer die gekennzeichneten Stellen bearbeitet und alle Markierungen entfernt hat, kann das Einchecken vollzogen werden.

Ausgangsversion mit Fehler:

```
int main()
{
    float um;
    float radius = 1.0;
    um = umfang(radius);
    printf("\n\nUmfang: %f\n\n",um);
}

// Funktion umfang()

float umfang(float durchmesser)
{
    return (durchmesser * 3.14);
}
```

Änderung Nutzer 1:

```
int main()
{
    float um;
    float radius = 1.0;
    float durchmesser;
    durchmesser = radius * 2.0;
    um = umfang(durchmesser);
    printf("\n\nUmfang: %f\n\n",um);
}

// Funktion umfang()

float umfang(float durchmesser)
{
    return (durchmesser * 3.14);
}
```

Änderung Nutzer 2:

```
int main()
{
    float um;
    float durchmesser = 2.0;

    um = umfang(durchmesser);
    printf("\n\nUmfang: %f\n\n",um);
}

// Funktion umfang()

float umfang(float durchmesser)
{
    return (durchmesser * 3.14);
}
```

Gemergte Version:

```
int main()
{
    float um;
    <<<<<<< umfang.c
    float durchmesser = 2.0;

    um = umfang(durchmesser);
    =====
    float radius = 1.0;
    float durchmesser;
    durchmesser = radius * 2.0;
    um = umfang(durchmesser);
    >>>>>>> 1.3
    printf("\n\nUmfang: %f\n\n",um);
}
...
```

Abb.: 2.7.2-2

2.7.3 Tags

In CVS is es möglich „Tags“ zu vergeben. Man kann verschiedenen Versionen verschiedener Dateien eine spezielle Markierung (Tag) zuordnen, um zum Beispiel die einzelnen Komponenten eines bestimmten Release zu kennzeichnen (vgl.: [Abb.: 2.7.3-1](#) und [Abb.: 2.7.3-2](#)).

```

file1  file2  file3  file4  file5

1.1     1.1     1.1     1.1  /--1.1*      <--*  TAG
1.2*-   1.2     1.2     -1.2*-
1.3  \- 1.3*-   1.3     / 1.3
1.4           \ 1.4  / 1.4
           \-1.5*-  1.5
                1.6
  
```

Abb.: 2.7.3-1 Quelle: [Ced2005]

```

file1  file2  file3  file4  file5

                1.1
                1.2
                1.3
            1.1  1.3
1.1     1.2     1.4     1.1
1.2*----1.3*----1.5*----1.2*----1.1  (--- <--- Look here
1.3           1.6     1.3
1.4           1.4
                1.5
  
```

Abb.: 2.7.3-2 Quelle: [Ced2005]

2.7.4 Branch/Merge

Das Branch/Merge Konzept wird von CVS für einzelne Dateien unterstützt. Dabei wird an die Versionsnummer aus der die Verzweigung entstanden ist eine Branchnummer angehängt, die den Zweig mit seinem Ursprung eindeutig identifiziert (vgl.: [Abb.: 2.7.4-1](#)). Auf den einzelnen Zweigen können wie gewohnt Checkin/Checkout Operationen durchgeführt werden.

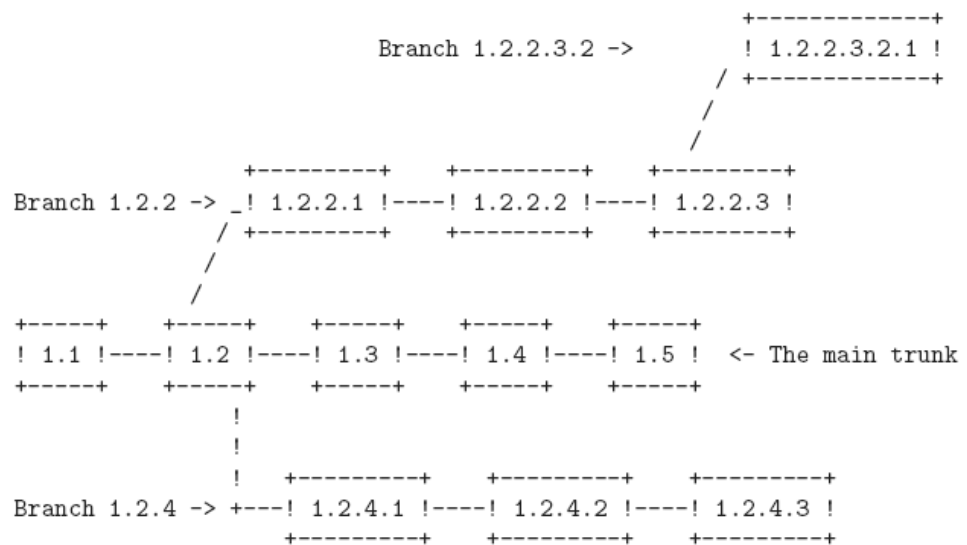


Abb.: 2.7.4-1 Quelle: [Ced2005]

3. Buildmanagement

3.1 Anforderungen und Ziele

Durch das Buildmanagement soll die Erstellung der Software transparent und wiederholbar gestaltet werden. Wiederholbar bedeutet hierbei, dass es zu jeder Zeit möglich sein muss auch eine ältere Version des Systems zu erstellen. Hierzu müssen die erforderlichen Quellen von der Versionsverwaltung bereitgestellt werden (vgl.: [2.7 Tool: CVS Concurrent Versions System](#)). Der Buildprozess sollte bei jeder Version des Systems gleich ablaufen. Dadurch muss der Ablauf nur einmalig beschrieben werden und Abfolge und Zeitdauer können besser geplant werden.

3.2 Prozessablauf

Elementare Schritte sind:

- *Festlegung des Buildprozessablaufs*
- *Bereitstellung/Dokumentation der benötigten Werkzeuge inklusive Einstellungen*
- *Gewährleistung einer stabilen Umgebung, in der der Build erstellt wird*

Den Ablauf des Systemerstellungprozess bis zum fertigen ausführbaren System zeigt [Abbildung 3.2-1](#).

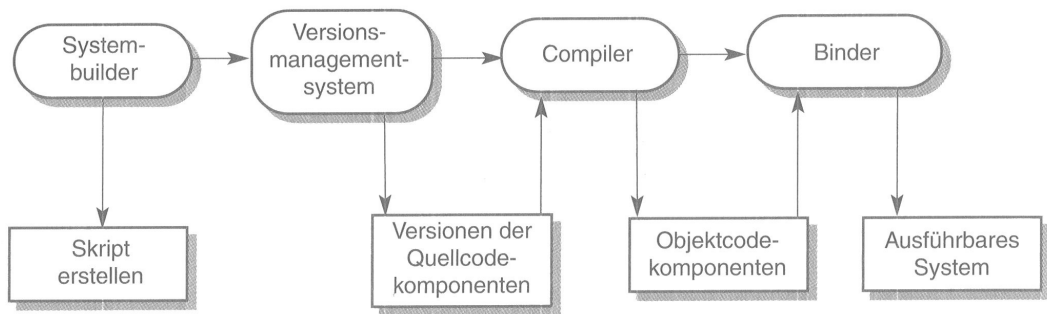


Abb.: 3.2-1 Quelle: [Som2001]

Um einen konstanten Ablauf des Buildprozesses garantieren zu können, wäre es wünschenswert, dass über den gesamten Entwicklungs- und Wartungsprozess alle Werkzeuge und Einstellungen zum Erstellen der Releases gleich bleiben würden. In der Praxis ist dies doch meist nicht der Fall. Aus diesem Grund müssen alle Informationen, die zur Wiederherstellung des Buildsystems notwendig sind, gesichert werden. Dies sind z. B. Versionsnummern von Compilern und eingebundener Standardbibliotheken. Die Verwaltung dieser Informationen unterliegt dem Konfigurationsmanagement (*vgl.: [2. Konfigurationsmanagement](#)*).

Um Seiteneffekte und ihre Auswirkungen besser kontrollieren zu können, sollte zur Erstellung der Releases im Idealfall ein eigener „Buildrechner“ vorhanden sein, der nur hierfür genutzt wird. Um die Wiederherstellung des Buildsystems zu gewährleisten, kann man beispielsweise nach dem Erstellen eines Release ein genaues Abbild (Image) des Buildrechners erstellen. Dadurch kann jederzeit wieder die exakte Umgebung rekonstruiert werden. Mit dieser Variante kann zwar das Buildsystem exakt wiederhergestellt werden, jedoch ist sie auch mit einem hohen Speicherbedarf verbunden. Die Erstellung von Images und die Wiederherstellung eines Systems aus einem Image ist speziell bei großen Systemen zeitintensiv. Eine Ressourcen-schonendere Möglichkeit wäre die Sicherung der kompletten Konfiguration des Buildrechners. Alle Einstellungen und Versionen der verwendeten Werkzeuge müssen dazu gespeichert werden. Die Entscheidung, wie das Buildsystem verwaltet wird, wird während der Planung für das Konfigurationsmanagement festgelegt.

4. Releasemanagement

Ein Release ist eine Systemversion, die extern für Kunden, oder auch nur intern für andere Entwickler freigegeben wird. Das Releasemanagement ist für den Zeitpunkt an dem das System freigegeben werden kann, den Prozess der Erstellung des Release und der Vertriebsdatenträger verantwortlich. Außerdem muss für die Dokumentation des Release Sorge getragen werden.

Ein Release kann neben dem ausführbaren Systemcode noch weiteres enthalten:

- **Konfigurationsdateien, zur Konfiguration für bestimmte Installationen**
- **Datenfiles, die für den erfolgreichen Betrieb erforderlich sind**
- **Ein Installationsprogramm**
- **Elektronische und gedruckte Dokumentation**
- **Verpackung und die damit verbundene Werbung**

4.1 Release-Entscheidungen

Die Entscheidung wann ein neues Release freigegeben wird hängt von vielen Faktoren ab und sollte wohlüberlegt erfolgen. Wenn die Auslieferung neuer Releases zu häufig erfolgt so könnte es sein, dass viele Kunden nicht auf die neue Version aufrüsten. Geschieht die Freigabe allerdings zu selten, so könnte ein gewisser Kundenstamm an die Konkurrenz verloren gehen. [Abbildung 4.1-1](#) aus [\[Som2001\]](#) zeigt Faktoren für die Freigabe neuer Systemversionen.

Faktor	Beschreibung
Technische Qualität des Systems	Wenn ernste Systemfehler festgestellt werden, die die Art und Weise beeinträchtigen, auf die viele Kunden das System benutzen, könnte es erforderlich sein, ein Release zur Fehlerbehebung herauszubringen. Geringfügige Systemfehler können jedoch durch die Veröffentlichung von Patches (häufig über das Internet vertrieben) behoben werden, die auf das aktuelle System-Release angewendet werden können.
Lehmans fünftes Gesetz	Dieses stellt fest, dass die durchschnittliche Erhöhung der Funktionalität in einem Release in etwa konstant ist. Gab es ein System-Release mit wesentlichen neuen Funktionen, könnte daher daraufhin ein Release zur Fehlerbehebung folgen.
Konkurrenz	Ein neues System-Release könnte erforderlich werden, weil ein Konkurrenzprodukt mit größerer Funktionalität zur Verfügung steht.
Marketinganforderungen	Die Marketingabteilung einer Organisation könnte sich verpflichtet haben, ein Release zu einem bestimmten Zeitpunkt zur Verfügung zu stellen.
Änderungsvorschläge von Kunden	Bei kundenspezifischen Systemen können die Kunden bestimmte Systemänderungen vorgeschlagen und bezahlt haben und erwarten nun ein System-Release, sobald diese implementiert sind.

Abb.: 4.1-1 Quelle: [\[Som2001\]](#)

Sommerville unterscheidet in diesem Beispiel bei der technischen Qualität des Systems zwischen ernststen und geringfügigen Systemfehlern. Diese Unterscheidung kann jedoch sehr gefährlich sein, auf alle Fälle können die Begriffe „ernst“ und „geringfügig“ unterschiedlich aufgefasst werden. Man muss sich sehr genau überlegen, wie man einen Fehler klassifiziert. Es könnte beispielsweise ein kleiner Fehler im Code vorhanden sein, der in keiner Weise die Arbeit des Kunden mit dem System beeinträchtigt. Dieser Fehler könnte aber einem Angreifer die Möglichkeit geben, mit einer bösartigen Attacke, auf sensible Daten zuzugreifen. Selbst wenn ein so genannter „geringfügiger“ Fehler nur ein einziges Mal, bei einem einzelnen Kunden aufgetreten ist und dabei einen Systemabsturz mit Datenverlust verursacht hat, kann man den Fehler nicht unbedingt als geringfügig einstufen. Die Art und Weise, in der viele der Kunden das System benutzen ist zwar nicht beeinträchtigt, für den einzelnen betroffenen Kunden kann jedoch ein erheblicher Schaden entstanden sein. Deshalb müssen nicht nur die Fehler, sondern vielmehr die Auswirkungen dieser Fehler sehr genau untersucht und kritisch bewertet werden, bevor ein Fehler klassifiziert wird.

4.2 Release-Erstellung

Die Release Erstellung ist der Prozess in dem die Komponenten, also Dateien und Dokumentationen des Releases erstellt werden. Eventuell müssen verschiedene Installationsskripts und Konfigurationsbeschreibungen für unterschiedliche Plattformen und Betriebssysteme erstellt werden. Alle Dateien und Beschreibungen werden zusammengestellt und zur Auslieferung bereitgestellt. Die Auslieferung an die Kunden kann klassisch über Datenträger (CDs, DVDs) oder über das Internet erfolgen.

4.3 Release-Dokumentation

Jedes Release muss dokumentiert werden um es auch in Zukunft exakt nachproduzieren zu können. Speziell bei kundenspezifischen Systemen mit langer Lebensdauer ist dies wichtig. Hier können auch nach Jahren noch Änderungswünsche an einer speziellen Version auftreten. Aus der Dokumentation muss genau ersichtlich sein, welche Versionen der einzelnen Quellcode Dateien für das Erstellen der ausführbaren Dateien verwendet wurden. Versionsverwaltungstools wie CVS stellen dazu Funktionen bereit, mit denen beispielsweise alle verwendeten Dateien mit einer bestimmten Marke gekennzeichnet werden (*vgl.: [2.7.3 Tags](#)*).

5. Diskussion/Zusammenhang

In den bisherigen Ausführungen wurden Build- und Releasemanagement noch als eigene Punkte neben dem Konfigurationsmanagement betrachtet. Es besteht aber ein enger Zusammenhang zwischen diesen Punkten. Build- und Releasemanagement bauen stark auf dem Konfigurationsmanagement auf. Es stellt sich die Frage:

- ***Warum ist Konfigurationsmanagement zentral für Build- und Releasemanagement?***

Beim Buildmanagement treten bei der Erstellung einer Systemversion u. a. folgende Fragen auf:

- ***Sind die erforderlichen Versionen des Compilers und der benötigten Werkzeuge verfügbar?***
- ***Sind die richtigen Versionen der verwendeten Bibliotheken verfügbar?***
- ***Aus welchen Komponenten besteht das System?***
- ***Welche Versionen der Komponenten müssen verwendet werden?***

All diese Fragen können nur vom Konfigurationsmanagement beantwortet werden. Die erforderlichen Versionen der einzelnen Komponenten und Werkzeuge werden von der Versionsverwaltung bereitgestellt. Die Entscheidung, welche Komponenten und welche Versionen dieser Komponenten zu der entsprechenden Konfiguration gehören, muss vor dem Buildprozess vom Konfigurationsmanagement gefällt werden.

Beim Releasemanagement müssen ähnliche Fragen bedacht werden:

- ***Welche Änderungen/Erweiterungen wurden im neuen Release umgesetzt?***
- ***Sind alle benötigten Datenfiles vorhanden?***
- ***Welche Versionen welcher Datenfiles müssen verwendet werden?***
- ***Wie müssen die Verweise auf die Datenfiles auf dem Zielsystem aussehen?***
- ***Sind die Dokumentationsdateien vorhanden?***
- ***Welche Versionen der Dokumentationsdateien müssen mitgeliefert werden?***

Die Informationen, die nötig sind um diese Fragen zu beantworten, werden vom Konfigurationsmanagement verwaltet. Die entsprechenden Versionen der benötigten Dateien werden im Versionsverwaltungssystem gehalten.

Aus den obigen Gründen ist es für das Build- und Releasemanagement wichtig, dass sie auf dem Konfigurationsmanagement basieren. Eine enge Verbindung und geregelte Kommunikationswege sind wichtig, damit die benötigten Informationen immer zur Verfügung stehen.

6. Zusammenfassung

Moderne Softwaresysteme werden immer komplexer, für einzelne Personen ist es nicht mehr möglich ein gesamtes System zu überschauen und zu begreifen. Konfigurationsmanagement wird an dieser Stelle immer wichtiger. Der gesamte Entwicklungsprozess wird überwacht, überprüft und dokumentiert. Alle Änderungen können nachvollzogen und bei Bedarf rückgängig gemacht werden. Man trägt somit dazu bei, die Entstehung von unbrauchbaren Softwareprodukten zu vermeiden. Die Wartung und Weiterentwicklung wird erleichtert. Selbst bei kleinen Softwareprojekten empfiehlt sich die Anwendung verschiedener Konfigurationsmanagementmechanismen, wie den Einsatz eines Versionsverwaltungssystems.

Die Erstellung von ausführbaren Systemversionen und das Management von Freigaben unterliegen durch Build- und Releasemanagement einem kontrolliertem Ablauf und bedürfen genauer Planung. Alle dazu benötigten Daten werden vom Konfigurationsmanagement verwaltet.

7. Quellen/Literatur

- [Bal1998] Helmut Balzert: Lehrbuch der Software-Technik, Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung
Spektrum, Akad. Verl., 1998
- [Som2001] Ian Sommerville: Software Engineering, 6th edition
Adisson-Wesley, 2001
- [For2004] Peter Forbrig: Lehr- und Übungsbuch Softwareentwicklung
Fachbuchverlag Leipzig, 2004
- [Bru2004] Bernd Brügge: Objektorientierte Softwaretechnik
Prentice Hall, 2004
- [Ced2005] Per Cederqvist: Version Management with CVS
Ximbiot <http://ximbiot.com>, 2005
- [sok2005] www.software-kompetenz.de: (Seiten über Konfigurationsmanagement)
Stand: Oktober 2005
- [vxt2005] www.v-modell-xt.de: V-Modell XT - Der Entwicklungsstandard für IT-Systeme des Bundes
Stand: Oktober 2005
- [v972005] www.v-modell.iabg.de: V-Modell - Der Entwicklungsstandard für IT-Systeme des Bundes
Stand: Oktober 2005
- [ant2005] ant.apache.org: Apache Ant
Stand: Oktober 2005
- [svn2005] svnbook.red-bean.com: Version Control with Subversion
Stand: Oktober 2005
- [cmm2005] www.sei.cmu.edu/cmmi/: Capability Maturity Model Integration (CMMI)
Stand: Oktober 2005
- [ans2005] www.ansi.org/: American National Standards Institute
Stand: Oktober 2005