Einführung in das Konfigurationsmanagement

Udo Kelter

09.03.2003

Zusammenfassung dieses Lehrmoduls

Im Laufe der Entwicklung eines Softwaresystems entstehen viele Versionen von Quelltexten, begleitender Dokumentation und anderen Software-Elementen. Gruppen einzelner Elemente bilden Konfigurationen. Von größeren Systemen existieren oft Varianten. Dieses Lehrmodul erklärt die grundlegenden Begriffe, Aufgaben und Lösungsansätze des Konfigurationsmanagements, soweit sie für kleinere Projekte relevant sind.

Vorausgesetzte Lehrmodule: keine

Stoffumfang in Vorlesungsdoppelstunden: 0.7

Inhaltsverzeichnis

1 Grundbegriffe

Bei der Entwicklung von Software sind viele Dokumente zu verwalten. Den Begriff Dokument verstehen wir sehr allgemein, Beispiele sind der Quelltext einer Klasse, ein Testdatenbestand, ein Kapitel eines Handbuchs, ein Diagramm usw. In der Literatur wird hier meist die Bezeichnung **Software-Element** verwendet.

Über die Zeit hinweg existieren viele Versionen eines einzelnen Dokuments bzw. Software-Elements. Es gibt zwei Arten von Beziehungen zwischen Versionen:

- Wenn Version B **Revision** einer Version A ist, dann ist B der Nachfolger von A und zeitlich jünger. B ersetzt A in bestimmten Kontexten.
- Wenn die Versionen A und B **Varianten** voneinander sind, dann existieren beide zeitlich parallel. Beispiele für Varianten sind Programmodule, die Meldungen in verschiedenen Sprachen ausgeben oder die auf verschiedenen Versionen einer Bibliothek basieren.

Ein komplettes (lauffähiges oder auslieferbares) System besteht fast immer aus mehreren Software-Elementen, die unabhängig voneinander versioniert sein können. Eine Gruppe zueinander passender Versionen von Software-Elementen nennt man eine **Konfiguration**.

Ein Software-Element, das aus Sicht des Konfigurationsmanagements keine Teile enthält, die unabhängig voneinander versioniert werden können, heißt dagegen **Atom**.

Die Unterscheidung zwischen Konfiguration und Atom hängt manchmal vom Standpunkt ab: Eine Modulbibliothek kann für einen Verwender als Atom erscheinen, während sie für den Hersteller eine Konfiguration ist.

Ein **Release** (oder eine Freigabe) ist eine Konfiguration, die an einen externen Kunden weitergegeben wurde; hier bestehen erhöhte Anforderungen an die Dokumentation und die Überprüfung, ob alle geplanten Änderungen tatsächlich realisiert sind. Releases haben eine eigene Numerierung, die unabhängig von der Numerierung der zugrundeliegenden Konfigurationen ist.

Ein Software-Element kann aus einem oder mehreren anderen Elementen automatisch (d.h. ohne Benutzereingaben) **abgeleitet** sein. Beispielsweise ist ein Bindemodul aus dem Quelltext des Moduls (und ggf. Include-Dateien) ableitbar, aus einem LaTeX-Quelltext sind verschiedene Anzeigeformate (PostScript, PDF usw.) ableitbar. Ein nicht abgeleitetes Element bezeichnen wir als **Quellelement**.

2 Aufgaben des Konfigurationsmanagements

2.1 Übersicht

Aufgabe des **Konfigurationsmanagements** (KM) ist es, die Software-Elemente, die im Laufe der Entwicklung eines Systems oder einer Systemfamilie entstehen, zu verwalten. Hierbei tritt eine Vielzahl von Detailproblemen auf.

Wir zitieren i.f. die Liste der KM-Aufgaben gemäß dem ISO-Standard ISO9000, Teil 3 (ISO9001), darin Kapitel 6 "Qualitätssicherungssystem – Unterstützende Tätigkeiten (phasenunabhängig)", Abschnitt 6.1 "Konfigurationsmanagement (KM)", Unterabschnitt 6.1.1. "Allgemeines":

> Das KM stellt einen Mechanismus zur Identifizierung, Lenkung und Rückverfolgung der Versionen jedes Software-Elements dar. In vielen Fällen sind auch frühere, nach wie vor in Verwendung befindliche Versionen zu warten und zu lenken. Das KM sollte
>
> 1. die Versionen jedes Software-Elements (SE) eindeutig identifizieren;
> 2. die Versionen von Software-Elementen, die gemeinsam eine bestimmte Version eines Produkts bilden, identifizieren;
> 3. den Entwicklungsstatus von Softwareprodukten während der Entwicklung bzw. nach der Lieferung und Installation identifizieren;
> 4. das gleichzeitige Überarbeiten eines bestimmten Software-Elements durch mehr als eine Person lenken;

5. für die Koordinierung der Überarbeitung von mehreren Produkten an einer oder mehreren Stellen sorgen, sofern notwendig;
6. alle Aktionen und Änderungen, die aus einem Änderungsvorhaben resultieren, identifizieren und von der Auslösung bis zur Freigabe verfolgen.

Manche dieser Probleme treten erst bei sehr umfangreichen Systemen mit mehrjähriger Entwicklungszeit auf, andere schon bei kleinen Systemen, die z.B. im Rahmen von Programmierpraktika realisiert werden. Wir konzentrieren uns hier auf die zweite Art von Problemen.

Konfigurationsmanagementsysteme (**KMS**) bieten Funktionen an, die KM-Aufgaben lösen. Bekannte Beispiele für KMS sind SCCS (Source Code Control System), RCS (Revision Control System) und CVS (Concurrent Versions System); [Ea01] bietet eine umfangreiche Produktübersicht.

Die gängigen KMS gehen davon aus, daß die Dokumente in Dateien gespeichert werden - übliche Werkzeuge arbeiten mit Dateien -, andere Datenhaltungssysteme werden nicht unterstützt. Ferner wird davon ausgegangen, daß ein Dokument in genau einer Datei gespeichert wird, so daß die Begriffe Dokument und Datei zusammenfallen.

2.2 Archive

Für die tägliche Arbeit sind normalerweise nur die jeweils neuesten Revisionen der Dokumente relevant; alle früheren Versionen werden nur in Ausnahmefällen gebraucht. Würde man nun alle früheren Versionen als Dateien mit entsprechenden Versionsnummern in den Arbeitsverzeichnissen speichern, z.B.

```
classABC_v01.java  ....  classABC_v37.java  ....
classABC_v01.class  ....  classABC_v37.class
```

usw., so wäre die Vielzahl der Dateien sehr störend. Außerdem müßte ein Entwickler ständig die aktuelle Versionsnummer kennen und benutzen, was stört und fehleranfällig ist.

Um diese Probleme zu lösen, realisieren KM-Systeme sogenannte Archive. Ein **Archiv** (auch **Repository** genannt) ist aus Benutzer-

sicht eine "Datenbasis", in der versionierte Software-Elemente verwaltet werden können, d.h. es können neue Elemente bzw. Versionen eingetragen und vorhandene ausgelesen werden. Insb. können also früher vorhandene Konfigurationen wiederhergestellt werden.

Das KMS verwaltet diese Datenbasis und bietet entsprechende Kommandos für den Zugriff auf die Datenbasis an. Speicherungstechnisch realisiert werden Archive durch eigene Verzeichnisse und darin enthaltene Dateien, auf die man aber im Sinne der Datenkapselung nicht direkt zugreifen sollte. Innerhalb der Archive werden die Versionen nicht komplett gespeichert, sondern nur die Differenzen. Hierbei wird ausgenutzt, daß sich Versionen von Programmen meist nur punktuell unterscheiden.

2.3 Arbeitsbereiche

Der eigentliche Dokumentbestand befindet sich also im Archiv und *nicht* mehr in Form normaler Dateien im Dateisystem! Wenn Entwickler nun Dokumente mit üblichen dateibasierten Werkzeugen bearbeiten wollen, müssen sie erst Kopien im Dateisystem erzeugen. Diese Kopien werden in sogenannten Arbeitsbereichen verwaltet (s. Bild 1).

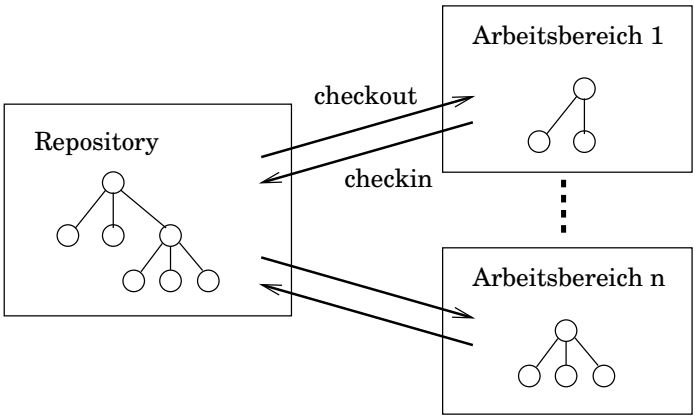

Abbildung 1: Repository und Arbeitsbereiche

Ein **Arbeitsbereich** ist im einfachsten Fall ein normales Verzeichnis, in dem allerdings das KMS einige spezielle eigene Dateien oder Unterverzeichnisse angelegt hat; in diesen ist vermerkt, welche Versionen welcher Dokumente hierhin kopiert worden sind.

Zu einem Archiv kann es mehrere Arbeitsbereiche, die unterschiedlichen Benutzern gehören, geben.

Um Kopien von Dokumenten in einem Arbeitsbereich anzulegen, stellt das KMS ein Kommando namens `checkout` (oder `get` oder `fetch`) zur Verfügung. Man spricht hier auch von "Ausleihe" oder "Ausbuchen".

Die Kopie kann dann bearbeitet werden. Nach Beendigung der Bearbeitung muß die neue Version in das Archiv übernommen werden; das entsprechende Kommando des KMS heißt meist `checkin` (oder `commit`). Einige Details solcher Kommandos besprechen wir später.

Es ist nicht möglich, Elemente im Repository zu überschreiben; stattdessen werden immer neue Revisionen eingefügt. Alte Versionen können gelöscht werden, allerdings sollte, sofern technisch möglich, dies für normale Benutzer verboten sein und dem Administrator des Repositorys vorbehalten bleiben.

2.4 Dokumentation

Es muß dokumentiert werden, wann und warum neue Versionen angelegt wurden und was die Versionen voneinander unterscheidet. Während bestimmte Daten (Datum der Entstehung einer Version, Autor, nächsthöhere Versionsnummer) beim `checkin` automatisch gewonnen werden können, kann die eigentlich interessierende inhaltliche Beschreibung der Änderung nur vom Entwickler angegeben werden.

2.5 Identifizierung von Versionen und Konfigurationen

Einzelne Dokumente (bzw. Dateien) wird man relativ häufig in das jeweilige Archiv eintragen; dies bietet den Vorteil, auf frühere Zustände zurücksetzen zu können. Die Versionsnummern werden daher relativ groß und korrespondieren nicht mehr mit Nummern, mit denen das Gesamtsystem nach außen hin versioniert wird. Daher sollte es neben

der i.d.R. automatischen Durchnumerierung neuer Versionen möglich sein, einzelnen Elementen zusätzlich explizit symbolische Bezeichner zu geben (z.B. “1.0”, “1.3-beta”).

Elemente mit gleichem symbolischen Bezeichner bilden eine zusammengehörige Gruppe; es sollte möglich sein, eine derartige Gruppe von Elementen mit einem einzigen Kommando in einem Arbeitsbereich zu restaurieren.

2.6 Überwachung paralleler Änderungen

Softwaresysteme werden arbeitsteilig in Gruppen entwickelt. Wenn nun mehrere Entwickler das gleiche Software-Element bearbeiten, können Änderungen überschrieben werden und verlorengehen. Selbst dann, wenn in den Editoren überprüft wird, ob eine zu schreibende Datei zwischenzeitlich verändert worden ist, hat man ein Problem:

- Die Kopien der Datei sind unabhängig voneinander verändert worden (man hat praktisch zwei Varianten erzeugt). Man muß nun zunächst alle Änderungen in eine einzige neue Version zusammenmischen; bei Texten ist dies oft automatisch möglich, bei anderen Dokumenttypen kommt man ggf. nicht an einer manuellen Integration der Änderungen vorbei.
- Die parallel vorgenommenen Änderungen können unverträglich sein, müssen also ganz oder teilweise aufgehoben und erneut bearbeitet werden. Hierdurch kann viel Arbeitsaufwand verlorengehen.

Um derartige Probleme zu vermeiden, kann man zwei unterschiedliche Strategien anwenden: Sperrungen oder Varianten und automatische Mischung.

Sperrungen. Bei dieser Strategie hindert das KMS einen Entwickler daran, ein Dokument zu bearbeiten, das gerade von einem anderen Entwickler bearbeitet wird. Hierzu wird ein Dokument bei einem `checkout` gegen ein erneutes `checkout` durch einen anderen Benutzer gesperrt. Erst wenn der Halter der Sperre diese aufhebt und ggf. eine neue Version einspielt, kann jemand anders das Dokument bearbeiten.

Bild 2 enthält ein Beispiel hierzu. Das Repository enthält mehrere Revisionen eines Dokuments, die neueste (mit Nr. 2.17) wurde in Arbeitsbereich 1 übertragen. Nun wird versucht, Version 2.17 in Arbeitsbereich 2 zu übertragen. Dies würde hier abgelehnt werden.

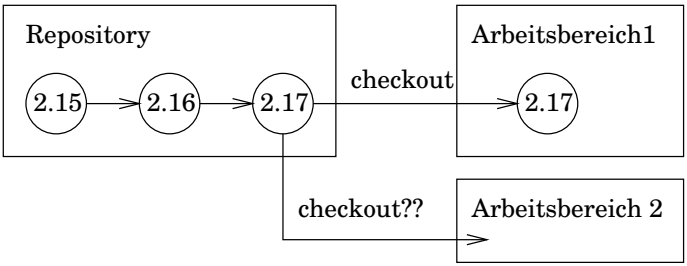

Abbildung 2: Paralleles `checkout`

Varianten und automatische Mischung. Das KMS verhindert hier parallele Änderungen nicht – die Entwickler arbeiten also ungeschützt –, aber es warnt die Entwickler und erzeugt im Konfliktfall Varianten bzw. unterstützt Mischungen.

Im Beispiel in Bild 2 wäre der Versuch, Version 2.17 in Arbeitsbereich 2 zu übertragen, erfolgreich. Das KMS sollte indes eine Warnung ausgeben, daß diese Version in einen Arbeitsbereich übertragen und von dort noch nicht zurückübertragen worden ist.

Aus jedem (!) Arbeitsbereich kann mit `checkin` eine neue Revision des Dokuments im Repository erzeugt werden. Bild 3 zeigt die Lage nach einem `checkin` aus Arbeitsbereich 2.

In unserem Beispiel kann anschließend aus Arbeitsbereich 1 heraus *nicht mehr* ohne weiteres eine neue Version des Dokuments (die Nummer 2.19 bekommen müßte) im Repository angelegt werden. Hier liegt der Fall vor, daß ein Dokument mit `checkin` eingetragen werden soll und dieses Dokument zwischenzeitlich schon von einem anderen Arbeitsbereich her eingetragen worden ist. Es gibt nun zwei Lösungen des Problems:

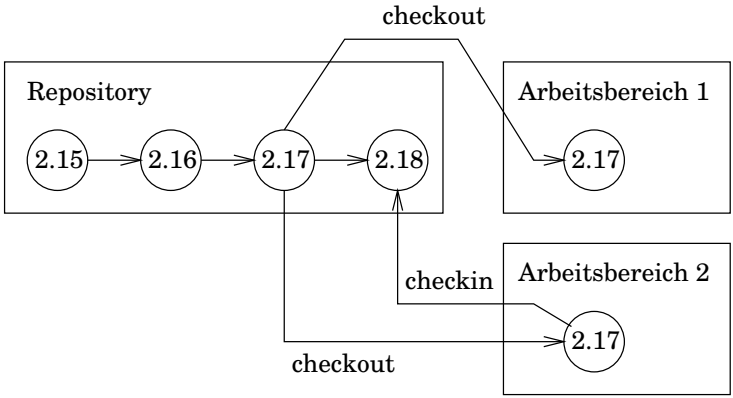

Abbildung 3: Zustand nach einem `checkin`

– Die neue Version wird eine Variante der Ursprungsversion. In unserem Beispiel könnte sie z.B. die Nummer 2.17.1.1 bekommen. Parallel zu den Versionen 2.17, 2.18 usw. könnte ein separater Strang 2.17.1.1, 2.17.1.2 usw. geführt werden. Dieses Vorgehen ist aber wenig sinnvoll, sofern die Varianten auf Dauer wieder zusammengeführt werden müssen; das Problem, die Varianten zu mischen, wird nur aufgeschoben und komplizierter.
– Beide Versionen werden sofort gemischt. Diese Mischung wird in unserem Beispiel in Arbeitsbereich 1 durchgeführt. Angenommen, das Dokument hat den Namen `xyz`, dann existiert in Arbeitsbereich 1 bereits eine Datei dieses Namens, die die modifizierte Version 2.17 enthält. Für die Mischung benötigen wir dort zusätzlich die Version 2.18. Im einfachsten Fall führt das KMS die Mischung vollautomatisch durch, d.h. im Inhalt der Datei `xyz` werden die Änderungen, die in Version 2.17 gegenüber Version 2.18 erfolgten, nachvollzogen. Nunmehr kann die Datei `xyz` als modifizierte Kopie von Version 2.18 (!) betrachtet werden, d.h. sie kann sofort oder nach ggf. weiteren Modifikationen als Nachfolgeversion zu Version 2.18 im Repository eingetragen werden.

Wichtig ist hier auf alle Fälle, daß das Zusammenmischen von ent-

standenen Varianten sehr gut unterstützt wird. Neben automatischen Verfahren sind für die Fälle, wo der Benutzer selbst entscheiden muß, gute Werkzeuge zum Darstellen der Differenzen zwischen Dokumenten und zum Erzeugen einer "vereinigten" Version wichtig.

Zusätzlich sollte das KMS jemanden, der ein Dokument bearbeitet, über ein Nachrichtensystem informieren, wenn jemand anders das gleiche Dokument aus dem Repository kopiert hat und möglicherweise bearbeiten wird.

Manche KMS unterstützen beide Strategien, manche nur eine.

Vergleich der Strategien. Eine parallele Bearbeitung von Dokumenten läßt sich manchmal schlecht vermeiden, weil andernfalls erhebliche Wartezeiten auftreten würden. Eine gute Unterstützung für das Zusammenmischen von Varianten ist daher auf alle Fälle sehr wünschenswert.

Das Sperren beim `checkout` erscheint zunächst besser, weil sicherer. In der Praxis treten allerdings leicht Probleme auf, wenn z.B. vergessen wird, Sperren freizugeben, oder wenn Inhaber von Sperren ungeplant ausfallen (z.B. wegen Krankheit), wodurch dann ärgerliche Verzögerungen in der Projektarbeit auftreten. Es besteht auch die Gefahr, daß auf Verdacht unnötig viele Dokumente ausgebucht und damit vermeidbare Blockaden erzeugt werden.

Letztlich ist abzuwägen, welches Risiko größer ist: Arbeitsverzögerungen durch Sperren oder die Beseitigung eventuell entstandener Varianten. Bei der Entscheidung spielt auch der Arbeitsstil in einer Gruppe eine Rolle. In vielen Entwicklungsprojekten entstehen Varianten nur selten und sind leicht zu beseitigen. Sofern gravierende Differenzen in Varianten auftreten, deutet dies darauf hin, daß das System schlecht modularisiert ist oder daß in der Arbeitsgruppe kein Konsens über die Struktur des Systems besteht; in solchen Fällen helfen aber auch Sperren nicht viel, und es sollten zunächst einmal diese originären Probleme gelöst werden.

Bei vielen "binären" Dokumenten - wozu Texte in den Dateiformaten vieler gängiger Textprozessoren gehören, ferner Graphiken - sind

keine brauchbaren Programme bzw. Verfahren zum Mischen von Dokumenten verfügbar. Hier sollte man besser sperren oder zu mischbaren Formaten (z.B. LaTeX für formatierte Texte) wechseln.

Generell muß ein KMS den Entwicklern Auskunftsfunktionen anbieten, über die sich die Entwickler über den Ausleihe- bzw. Bearbeitungszustand von Dokumenten informieren können.

Automatisches Mischen. In KMS, die nicht mit Sperrungen arbeiten, kommt es leicht zu Varianten, die wieder zusammengeführt werden müssen. Von Hand ist dies sehr umständlich, insb. wenn ein System aus einer Vielzahl einzelner Dateien besteht.

Einen Ausweg bilden automatische Mischverfahren. Hierbei werden zunächst "Einzeländerungen" in den beiden Varianten gegenüber der gemeinsamen Ausgangsversion bestimmt. Eine Einzeländerung ist eine Einfügung, Löschung oder Änderung einer oder mehrerer aufeinanderfolgender Textzeilen. Das Mischen besteht darin, *alle* Einzeländerungen auf die Ausgangsversion anzuwenden mit Ausnahme solcher, die zu dicht beieinander liegen.

Es wird hier unterstellt, daß Einzeländerungen in den beiden Varianten, die textuell weit voneinander entfernt liegen, *inhaltlich unabhängig voneinander* sind. Diese optimistische Annahme trifft in der Praxis meist zu, und die Fälle, wo zwei Änderungen unverträglich miteinander waren, werden meist rasch identifiziert (weil sie z.B. zu Syntaxfehlern in Programmen führen) und können dann von Hand bereinigt werden.

Bei Einzeländerungen, die zu dicht beieinander liegen, spricht man von einem **Mischkonflikt**. Sofern die Mischung nicht komplett abgebrochen wird, muß der Konflikt in geeigneter Weise dokumentiert werden. Im System CVS wird z.B. in dem Fall, daß in einer Datei `xyz` die gleiche Zeile der Ausgangsversion in den beiden Varianten unterschiedlich geändert worden, eine Warnung ausgegeben und in der Mischdatei ein Text nach Muster in Bild 4 eingetragen. Darin ist `:versionsnummer:` die Nummer der Variante im Repository. Die entstandene Mischdatei muß jetzt von Hand nachbearbeitet werden.


```
<<<<<<< xyz
  lokale Variante der Zeile im Arbeitsbereich
=======
  Variante der Zeile im Repository
>>>>>>> :versionsnummer:
```

Abbildung 4: Darstellung eines Mischkonflikts

3 Der KM-Plan

Die vorstehenden Abschnitte zeigten einige Probleme auf, die im Zusammenhang mit Versionen und Konfigurationen auftreten, und skizzierten zugehörige Funktionen in KMS. KMS alleine lösen die Probleme indes nicht, sie sind kein Ersatz für Projektmanagement, und selbst beim Einsatz von KMS muß in einem Projekt eine konsistente Strategie vorhanden sein, wie das Konfigurationsmanagement betrieben wird und wie KMS benutzt werden. ISO 9000 verlangt hierzu einen Konfigurationsmanagementplan (KM-Plan) und beschreibt diesen in Abschnitt 6.1.2 “KM-Plan” wie folgt:

> Der Lieferant sollte einen KM-Plan entwickeln, der folgende Punkte beinhaltet:
> 1. die Organisationen, die am KM beteiligt sind, sowie die ihnen zugewiesenen Tätigkeiten
> 2. die auszuführenden KM-Tätigkeiten
> 3. die zu verwendenden KM-Werkzeuge, -Technologien und Methoden
> 4. das Stadium, in dem Elemente der Konfigurationslenkung unterworfen werden sollen

Als Beispiel betrachten wir einen KM-Plan für das Programmierpraktikum:

1. organisatorische Einheiten:
   – Verantwortlich ist jede Gruppe als Ganze, ggf. vertreten durch einen Sprecher: zuständig für Erstellung des Dokumentrepositorys

- Betreuer (Wissenschaftlicher Mitarbeiter oder vertretungsweise studentische Hilfskraft): zuständig für Installation eines zentralen Repositorys und ggf. Unterstützung bei der Nutzung

2. auszuführende KM-Tätigkeiten:
   - Dem KM unterworfen werden sollen
     - die ggf. entwickelten UML-Diagramme
     - Java-Quelltexte (aber keine abgeleiteten Dateien)
     - das Benutzerhandbuch
     - die ggf. vorhandene Installationsanleitung

     Eine eigene Konfiguration ist anzulegen am Ende jedes der beiden Teilprojekte sowie, falls die Ergebnisse eines Teilprojekts nicht direkt abgenommen wurden, jede neue zur Abnahme vorgelegte Systemversion.
3. Werkzeuge und Methoden: Es ist ein zentrales Repository zu verwenden, das von den Betreuern eingerichtet wird und das auf CVS basiert.
4. Beginn KM: KM ist spätestens mit Abgabe der Ergebnisse des ersten Teilprojekts einzusetzen. Empfohlen wird der Einsatz ab Beginn des ersten Teilprojekts.

Literatur

[Ea01] Eaton, Dave (ed.): Configuration Management Tools Summary; `http://www.daveeaton.com/scm/CMTools.html`; 2001/05/01

Glossar

Archiv: Datenbasis, in der Versionen von Software-Elemente verwaltet werden

Arbeitsbereich (*work space*): Bereich im Dateisystem, in dem Kopien von Versionen aus einem Archiv stehen und bearbeitet werden.

Atom: Software-Element, das keine Teile enthält, die unabhängig voneinander versioniert werden können

`checkout`**:** Kopieren einer Version von einem Versionsarchiv in einen Arbeitsbereich

`checkin`**:** Erzeugen einer Nachfolgeversion zu einer zuvor ausgebuchten Version in einem Versionsarchiv

(Software-) Element: Dokument, das der Versions- und Konfigurationsverwaltung unterliegt

Konfiguration: Gruppe von zueinander passenden Versionen von Software-Elementen

Konfigurationsmanagement: Mechanismen zur Identifizierung, Lenkung und Rückverfolgung der Versionen von Software-Elementen

KM-Plan: Festlegung, zu welchen Zeitpunkten, für welche Dokumente, von wem und unter Einsatz welcher Werkzeuge Tätigkeiten im Rahmen des Konfigurationsmanagements durchzuführen sind

Mischung (*merge*): Produktion einer gemeinsamen Nachfolgeversion zu zwei vorhandenen Varianten; automatisch oder manuell

Mischkonflikt (*conflict*): Sachverhalt, daß zwei Varianten an der gleichen oder nahezu der gleichen Stelle gegenüber ihrer gemeinsamen Ursprungsversion verändert worden sind; verhindert automatisches Mischen an dieser Stelle

Release: Konfiguration, die an eine externe Instanz weitergegeben wurde; typischerweise mir zusätzlicher Benennung

Revision: Wenn Version B Revision einer Version A ist, dann ist B der Nachfolger von A und zeitlich jünger; B ersetzt A in bestimmten Kontexten.

Variante: parallel existierende und weiterentwickelte Versionen

Version: (a) Synonym zu Dokument (b) Dokument A ist Version von Dokument B, wenn ein Dokument aus dem anderen durch Bearbeitung hervorgegangen ist oder beide Dokumente durch Bearbeitung eines gemeinsamen Ursprungsdokuments

Index