Teil V

Verwaltung und Erhaltung von Software

21 Konfigurationsverwaltung

Software-Entwicklung ist Teamarbeit. Damit alle daran beteiligten Personen geordnet zusammenarbeiten können und die dabei erstellten Dokumente nicht heillos durcheinander abgelegt und geändert werden, müssen organisatorische und technische Vorkehrungen getroffen werden. Diese Regelungen und Maßnahmen fasst man mit dem Begriff der Konfigurationsverwaltung zusammen.

Da sie nicht unmittelbar zur Entstehung des Produkts beiträgt, wird sie oft unterschätzt und vernachlässigt. Dabei stellt sie für die Software-Projekte eine ebenso wichtige Infrastruktur dar wie die Stromversorgung für einen produzierenden Betrieb: Wenn sie vorhanden ist und funktioniert, bemerkt man sie nicht, aber wenn sie unterbrochen ist, geht nichts mehr.

21.1 Grundlagen der Konfigurationsverwaltung

Während der Software-Entwicklung entstehen viele Dokumente und Komponenten. Einige davon werden nur zur Entwicklung selbst benötigt (z. B. ein Testtreiber), andere werden, bevor sie an den Kunden ausgeliefert werden, noch mehrfach korrigiert und geändert (z. B. die Bedienkomponente).

Es ist nahezu unmöglich, dabei den Überblick zu behalten, sofern man dazu nicht spezielle Vorkehrungen schafft. So kommt es immer wieder zu vermeidbaren Fehlern:

- Falsche Komponenten werden integriert.
- Bereits entwickelte Programmteile werden übersehen und erneut entwickelt.
- Testdaten werden einmal verwendet und gehen verloren, obwohl sie mit erheblichen Aufwand entworfen wurden und in der Wartung höchst nützlich wären.
- Größere Änderungen werden begonnen, ohne dass der Stand *vor* der Änderung archiviert ist; darum ist später weder ein Rückzug noch ein Vergleich möglich.
- Modifikationen an Dokumenten und Komponenten gehen verloren, weil sie von anderen Entwicklern überschrieben werden.

Noch schwieriger wird die Situation, wenn die Entwicklung beendet ist und das System beim Kunden läuft. Jetzt gibt es nur noch wenige Leute, die überhaupt von der Software wissen, und da sie nur noch gelegentlich daran arbeiten, haben sie im Allgemeinen keinen Überblick. Diese Situation ist sehr riskant: Die Integration modifizierter Komponenten führt zu unerwarteten Problemen, weil die Integration des laufenden Systems nicht sorgfältig dokumentiert war. Längst beseitigte Fehler tauchen wieder auf, weil das fehlerhafte Modul nicht gelöscht wurde. Einem Kunden, der aus einem fernen Land einen Fehler gemeldet hat, kann nicht geholfen werden, weil sich seine Software-Konfiguration nicht ermitteln lässt; man weiß zwar, welches System er hat, aber man kennt nicht die genauen Versionen oder hat sie nicht archiviert.

Die Konfigurationsverwaltung hat den Zweck, diese Schwierigkeiten zu vermindern oder zu vermeiden.

21.1.1 Software-Einheiten

Bevor wir auf die Konfigurationsverwaltung und ihre Ziele eingehen, müssen wir die Begriffe Version, Variante und Konfiguration klären. Um von den unterschiedlichen Dokumentarten zu abstrahieren, führen wir den Oberbegriff *Software-Einheit* ein. Wir definieren:

Software-Einheit — Ein Stück Software, das im Software-Lebenslauf entsteht und für die Entwicklung, den Betrieb oder die Wartung des Software-Systems relevant ist. Eine Software-Einheit kann unabhängig von anderen Software-Einheiten bearbeitet, gespeichert und ersetzt werden. Sie besteht nicht aus kleineren Software-Einheiten, sie ist also im Sinne der Verwaltung atomar.

Die Unabhängigkeit von anderen Software-Einheiten betrifft nur die Verwaltung, nicht die Schnittstellen.

21.1.2 Versionen

Aus einer Software-Einheit E entsteht eine neue *Version*, wenn in E eine Änderung durchgeführt wird, die diese Einheit in irgendeinem Sinne besser macht, z. B. eine Korrektur, eine Erweiterung oder eine Anpassung an veränderte Bedingungen. Die neue Version (der *Nachfolger*) ersetzt die alte (den *Vorgänger*) und übernimmt deren Bezeichner (genauer: deren festen Teil). Damit die beiden nicht verwechselt werden, gibt es im Bezeichner (typischerweise am Ende) eine ein- oder mehrstufige Versionsnummer, die bei der Bildung einer neuen Version erhöht wird. Beispielsweise ersetzt das neue OS10.4.6 das ältere OS10.4.5. Die Versionen stehen also in einer zeitlichen Ordnung hintereinander.

Der IEEE-Standard unterscheidet zwischen Version und Revision; die Revision entsteht, wenn eine Software-Einheit nicht komplett ersetzt, sondern repariert wird (z. B. durch Austausch einzelner Seiten in einem Dokument).

version — (1) An initial release or re-release of a computer software configuration item, associated with a complete compilation or recompilation of the computer software configuration item.

(2) An initial release or complete re-release of a document, as opposed to a revision resulting from issuing change pages to a previous release.

IEEE Std 610.12 (1990)

Wir verzichten darauf, die übrigen einschlägigen Definitionen aus dem IEEE-Glossar zu zitieren, sie erscheinen uns allzu eng.

21.1.3 Varianten

Anders als die Versionen stehen *Varianten* zeitlich nicht *hintereinander*, sondern *nebeneinander*. Wenn Kunde A ein Software-System bekommen hat, das später auch an den Kunden B verkauft werden kann, dann sind meist für B spezifische Anpassungen nötig. Damit entsteht eine Variante. Ebenso entstehen Varianten, wenn eine Software für verschiedene Betriebs- oder Datenbanksysteme variiert wird.

Abbildung 21–1 zeigt schematisch den Zusammenhang zwischen Versionen und Varianten. X2 und X3 sind Versionen von X1. Xa2 und Xb2 sind Varianten von V2. X2 und seine Varianten Xa2 und Xb2 bilden zusammen eine *Variantenfamilie*.

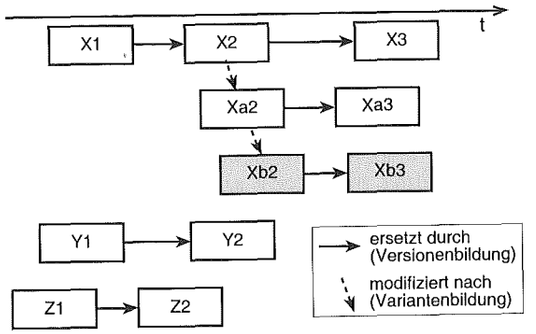

Abb. 21–1 Die Entstehung von Versionen und Varianten

21.1.4 Konfigurationen

Eine *Konfiguration* ist eine Menge von Software-Einheiten, die für einen definierten Zweck zusammengestellt sind und zueinander passen, sowie das Geflecht der Beziehungen zwischen diesen Software-Einheiten.

Bei einer Variantenfamilie bestimmt das dazugehörende Auswahlverfahren, wieviele ihrer Software-Einheiten in einer Konfiguration enthalten sein müssen

und sein dürfen. Wenn genau eine Einheit aus der Variantenfamilie gewählt werden muss, spricht man von einer 1-aus-N-Auswahl (z. B. die MySQL-spezifische Komponente zur Speicherung der Daten). Können mehrere Software-Einheiten derselben Variantenfamilie in einer Konfiguration enthalten sein, bezeichnet man dies als N-aus-M-Auswahl. So kann beispielsweise eine Konfiguration einer Web-Anwendung mehrere Varianten von Browser-spezifischen Komponenten enthalten, wenn diese Anwendung die Eigenschaften unterschiedlicher Browser unterstützen muss.

Natürlich kann dieselbe Software-Einheit in verschiedenen Konfigurationen enthalten sein. Wir sprechen auch bei den Konfigurationen von Varianten; zwei Konfigurationen stellen (Konfigurations-)Varianten dar, wenn sie unterschiedliche Varianten einer Variantenfamilie enthalten. In unserem Beispiel (Abb. 21–1) könnte die Konfiguration A aus den Einheiten Y1, Z1 und Xa2 bestehen, die Konfiguration B aus den Einheiten Y1, Z2 und Xb2.

21.1.5 Probleme mit Versionen und Varianten

In Zusammenhang mit Versionen und Varianten gibt es eine Reihe von Problemen:

Varianten erschweren die Wartung. Wenn die notwendige Änderung am System seine variantenspezifischen Teile betrifft, muss sie für jede Systemvariante einzeln durchgeführt werden. Darum bemüht man sich, die variantenspezifischen Teile eines Systems möglichst klein zu halten. Eventuell ist deshalb bei der Variantenbildung eine Aufspaltung von Software-Einheiten sinnvoll. Als Faustformel kann man sagen: Der Wartungsaufwand ist proportional der Summe von P_0 und aller P_i, wobei P_0 der Umfang der universellen Einheiten ist, die P_i sind die Umfänge der einzelnen Varianten.

Eine Variantenbildung mit »copy and paste« hat darum katastrophale Wirkung auf den Wartungsaufwand: P_0 ist null, die P_i sind jeweils so groß, wie vorher das Gesamtsystem war.

In vielen Fällen können oder wollen nicht alle Kunden auf eine neue Version umsteigen, beispielsweise, weil diese nicht mehr mit älteren Geräten kompatibel ist, die der Kunde weiterhin benutzt. In diesem Falle verwenden also zur selben Zeit verschiedene Kunden verschiedene Versionen. Der Hersteller muss dann auch die älteren Versionen vorhalten, um Probleme, die bei Kunden auftreten, nachvollziehen zu können.

Auch bei den Kunden, die nicht umgestellt haben, können Korrekturen oder Anpassungen nötig werden. Das widerspricht aber dem Versionenkonzept, wie es oben definiert wurde. Tatsächlich bilden sich in diesem Falle Varianten, auch wenn sie als Versionen bezeichnet werden. In der Abbildung 21–2 ist die Entstehung einer neuen Version V7.0 aus V6.6 dargestellt. Kunden, die weiterhin V6.6 verwenden, bekommen später noch eine korrigierte Fassung

V6.7. Damit ist V6.7 faktisch eine Variante von V7. In der Praxis verwendet man mehrstufige Nummern, um bei dieser Variantenbildung wenigstens nicht in Konflikt mit der Namenskonvention zu geraten. Außerdem weist die Änderung der ersten Nummer auf eine erhebliche Änderung hin, die weit über die Fehlerkorrektur (typisch für eine Änderung der letzten Zahl) hinausgeht.

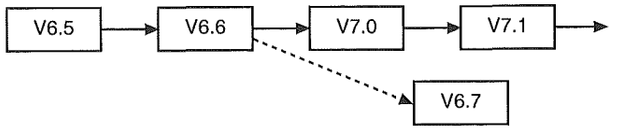

Abb. 21–2 Weiterführung einer älteren Version als Variante

Wenn die Gründe entfallen, die zu einer Variantenbildung geführt haben, können die Varianten wieder zusammengeführt werden (z. B. die Varianten a und b in Abb. 21–1). Diese Vereinigung ist schwierig, weil in den verschiedenen Varianten inkompatible Komponenten entstanden sein können. Die Kunden erhalten eine neue Version, die – aus ihrer Sicht – gegenüber dem Vorgänger nicht verbessert ist, sondern eventuell neue Mängel zeigt.

21.1.6 Baselines und Releases

Eine *Baseline* ist eine Konfiguration, die so zusammengestellt und geprüft ist, dass sie als stabil gilt und als Bezugspunkt für die weitere Entwicklung geeignet ist. Das *Release* ist eine Baseline, die mit dem Zweck definiert wurde, diese Konfiguration an die Kunden auszuliefern. Eine Baseline *kann*, ein Release *muss* ein lauffähiges System bilden.

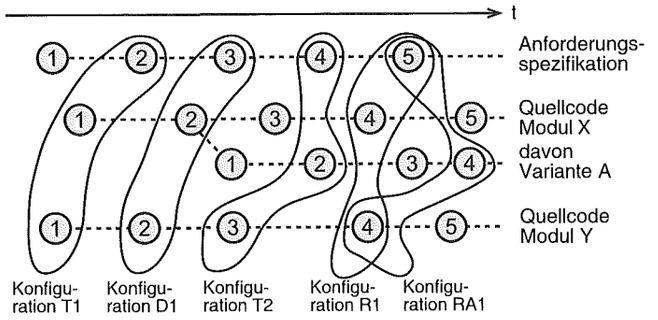

Abb. 21–3 Versionen, Varianten und Konfigurationen

Abbildung 21–3 zeigt schematisch die Entwicklungshistorie einiger Software-Einheiten sowie Konfigurationen über diesen Einheiten. Die Konfigurationen T1

und T2 wurden für den Test gebildet, die Konfiguration D1 ist ein Demonstrationsprototyp, die Konfiguration R1 wurde bei verschiedenen Kunden installiert, die Konfiguration RA1 ist kundenspezifisch (Auswahl einer entsprechenden Variante) und wurde nur beim Kunden A installiert. Die Konfigurationen R1 und RA1 sind Releases.

21.1.7 Konfigurationsverwaltung

Den systematischen Umgang mit den Konfigurationen eines Systems leistet die Konfigurationsverwaltung. Wir definieren:

Die **Konfigurationsverwaltung** ist diejenige Rolle oder Organisationseinheit, die die Software-Einheiten und Konfigurationen identifiziert, verwaltet, bei Bedarf bereitstellt und ihre Änderungen überwacht und dokumentiert. Dazu gehört auch die Rekonstruktion älterer Software-Einheiten und Konfigurationen.

Die Konfigurationsverwaltung schließt also die Versions- und Variantenverwaltung ein und geht damit über das hinaus, was die Bezeichnung ausdrückt. Die Grundidee der Konfigurationsverwaltung ist simpel:

> Die Konfigurationsverwaltung hat ein Monopol auf die Bereitstellung von Software. Damit wird verhindert, dass unbemerkt Varianten entstehen, unkontrollierte Änderungen stattfinden usw. Alle Entwickler übergeben ihre Software-Einheiten nur an die Konfigurationsverwaltung. Von dort und nur von dort erhalten alle anderen Stellen, vor allem die Prüf- und Integrationsabteilung, die Software. An der Konfigurationsverwaltung führt kein Weg vorbei.
>
> Alle Software-Einheiten, die von den Entwicklern erstellt und abgeliefert werden, werden eindeutig identifiziert, erfasst, archiviert und vor Änderungen geschützt.
>
> Überholte Versionen werden nicht einfach überschrieben, sondern aufbewahrt, damit sie später rekonstruiert werden können, sei es, weil man zu einem alten Zustand zurückkehren will, weil ein Kunde mit dieser Version Probleme hat oder weil man feststellen will, wann ein bestimmter Effekt erstmals aufgetreten ist.
>
> Wenn sich die Entwicklung verzweigt hat, werden alle Varianten bereitgehalten.
>
> Konfigurationen werden für die Auslieferung zusammengestellt und präzise dokumentiert.
>
> An allen Software-Einheiten, die der Konfigurationsverwaltung unterstehen, können Metriken erhoben werden (siehe Kap. 14).

Der Aufwand für die Konfigurationsverwaltung ist nicht unerheblich, ihr Nutzen übersteigt den Aufwand aber zuverlässig. Wo die Konfigurationsverwaltung bislang unzureichend ist – und das ist in vielen, vor allem kleineren Unternehmen der Fall –, lassen sich allein durch eine Verbesserung in diesem einen Punkt erhebliche Einsparungen erzielen.

21.2 Die Aufgaben der Konfigurationsverwaltung

Eine systematisch aufgesetzte und durchgeführte Konfigurationsverwaltung übernimmt die folgenden Aufgaben:

- *Versionskontrolle*

  Alle Software-Einheiten sind eindeutig identifiziert.
  Die erstellten Versionen und Varianten aller Software-Einheiten werden über lange Zeiträume verwaltet, ein Zugriff darauf ist jederzeit möglich. Nur Versionen, die nicht in Konfigurationen eingesetzt und vor längerer Zeit ersetzt wurden (Version 3 des Moduls X in Abb. 21–3), könnten gelöscht werden, wenn sie nicht aus speziellen Gründen aufzubewahren sind. Bei geeigneter Speichertechnik (nur die Änderungen werden gespeichert) ist das aber kaum sinnvoll.

- *Konfigurationskontrolle*

  Es können beliebige Konfigurationen des Software-Systems mit genau definierten Eigenschaften erstellt werden, wenn die benötigten Software-Einheiten vorliegen.
  Es ist sichergestellt, dass die Bestandteile einer Konfiguration konsistent, also miteinander verträglich sind.

- *Konstruktion ausführbarer Programme*

  Der Prozess, um aus dem Quellcode ausführbare Programme zu erzeugen (der sogenannte Build-Prozess), wird mit den Konfigurationsinformationen gesteuert und ist automatisiert.

- *Änderungskontrolle*

  Alle Änderungen an Software-Einheiten werden überwacht, alle Prüfresultate verwaltet (siehe Abschnitt 22.4).

- *Koordination der Teamarbeit*

  Die Zusammenarbeit der Entwickler oder Entwicklergruppen wird durch die gemeinsame Referenzumgebung und durch die Konflikterkennung oder -vermeidung unterstützt (siehe Abschnitt 21.4).

21.3 Identifikation und Benennung von Software-Einheiten

21.3.1 Die Benennung von Software-Einheiten

Wie bei technischen Zeichnungen erweist sich auch bei der Software ein *Bezeichnungsschema* zur Kennzeichnung der Software-Einheiten als sinnvoll. Einige Dokumente entstehen in jedem Falle, unabhängig von der konkreten Aufgabenstellung:

- Begriffslexikon
- Anforderungsspezifikation
- Systementwurf
- Installationsvorschrift
- Benutzungshandbuch
- Testvorschriften

Mit dem Entwurf werden die einzelnen Bestandteile des Software-Systems sichtbar und können benannt werden. Aber schon zu Beginn des Projekts kann man Software-Einheiten generisch definieren:

- Entwurfsbeschreibung für jedes Teilsystem, jede Komponente usw.
- Testvorschriften für den Modultest, den Integrationstest und den Systemtest
- Testberichte
- Quellcode
- Review-Berichte

Mit Ausnahme von Test- und Review-Berichten können alle diese Software-Einheiten im Laufe des Projekts geändert werden (siehe Tab. 12–1 auf S. 263).

Ein Bezeichnungsschema für Software-Einheiten basiert auf beschreibenden Kürzeln und Nummern. Die primitivste Bezeichnung besteht aus einem Kürzel und einer Nummer. Das andere Extrem ist eine Kette von Kürzeln ohne Nummer. Tabelle 21–1 zeigt Beispiele.

Bezeichnungsschema	Beispiel	Bestandteile
linear	DSG.00013	Produkt (DSG) und laufende Nummer
hierarchisch	DSG.EDA.EB	Produkt (DSG), Teilsystem (EDA), Art der Software-Einheit (Entwurfsbeschreibung)

Tab. 21–1 *Bezeichnungsschemata für Software-Einheiten (ohne Versionsnummer)*

Beide Verfahren haben Nachteile: Kürzel verlieren oft ihren Sinn, dann sind es nur noch unverständliche Chiffren. Zudem können bei einer Änderung des Entwurfs Einheiten in andere Umgebungen geraten. Sie müssen dann umbenannt werden, was einigen Aufwand verursacht. Nummern lassen keine Rückschlüsse

auf den Inhalt zu und werden leicht verwechselt und verdreht. Um die Versionen einer Einheit zu unterscheiden, hängt man in jedem Fall an die Bezeichnung noch eine Versionsnummer an.

Bei der Programmierung verwendet man »sprechende« Bezeichner. Dieses Prinzip sollte man aber nicht schematisch auf die Kennzeichnung der Software-Einheiten übertragen, denn diese werden oft in umfangreichen Listen, Konfigurationstabellen usw. geführt, die durch lange Bezeichner unübersichtlich werden. Zudem ist eine Software-Einheit in fast jedem Falle zu komplex, um wirklich durch einen Bezeichner, und sei er auch sehr lang, ausreichend charakterisiert zu werden.

Beim Code ist es zweckmäßig, den Programmnamen (d. h. den Namen des Moduls, Klasse o. Ä.) als Dateinamen zu verwenden. Auch bei allen anderen Dokumenten sollte man dafür sorgen, dass das Dokument nicht nur unter seinem Namen abgelegt ist, sondern diesen Namen auch explizit enthält, sodass man ohne Mühe feststellen kann, welches Dokument man gerade vor sich hat. Die Konsistenz dieser Angabe mit der Bezeichnung sollte automatisch überwacht werden.

Abbildung 21–4 zeigt ein Beispiel für die vollständige Kennzeichnung einer Software-Einheit: Sie gehört zum Produkt DSG, heißt DSG.EDA.EB und hat die Versionsnummer 02. Das Kürzel EB zeigt an, dass es sich um die Entwurfsbeschreibung der Komponente EDA handelt. Wenn Varianten entstehen, kommt ein weiteres Kürzel zur Kennzeichnung der Variante hinzu.

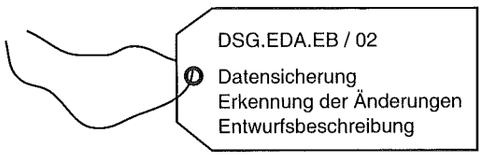

Abb. 21–4 *Kennzeichnung einer Software-Einheit*

21.3.2 Identität und Verwaltung von Software-Einheiten

Werkzeuge für die Konfigurationsverwaltung sorgen automatisch dafür, dass Versionsnummern fortlaufend vergeben werden und nicht versehentlich zwei verschiedene Software-Einheiten denselben Bezeichner bekommen.

Trotzdem kommt es immer wieder vor, dass die Mechanismen zur eindeutigen Kennzeichnung der Software-Einheiten unterlaufen werden. Ursachen können Schlampereien der Entwickler, Fehler bei der Konfigurationsverwaltung oder auch Notreparaturen sein, die fern vom Standort des Software-Herstellers vorgenommen wurden, weil eine korrekte Bearbeitung des Fehlers viel zu lange gedauert hätte. Wenn ein Ingenieur irgendwo im Urwald eine Anlage in Betrieb nehmen soll und das nur schafft, indem er ein Programm verändert (und sei es im Maschi-

nencode), dann wird er das tun. Wenn endlich alles funktioniert, wird er die Notreparatur höchstwahrscheinlich vergessen, und niemand weiß, dass eine Variante entstanden ist. Einige Monate später wird eine neue Version der Software zur Anlage geschickt und von den Betreibern installiert. Natürlich tritt nun wieder der Fehler auf, an den sich niemand mehr erinnert hat.

Mit einer Prüfsumme kann man Veränderungen praktisch sicher erkennen. Einfaches Aufaddieren reicht aber nicht aus, weil damit Umstellungen unerkannt bleiben können; man verwendet einen effizienten Hashing-Algorithmus, der den vollständigen Inhalt der Datei verarbeitet. Die Prüfsumme sollte bei allen Ausgaben und Anfangsmeldungen angezeigt werden und leicht zu überprüfen und abzufragen sein.

Die Prüfsumme kann man auch zur Identifikation binärer Dateien verwenden. Verteilt man die Software als Quellcode, kann man mit Hilfe der Prüfsumme feststellen, ob die ausführbare Form des Programms am Zielort richtig generiert wurde.

Die Informationen, die die Konfigurationsverwaltung speichern und verknüpfen muss, sind umfangreich und extrem kritisch hinsichtlich Verlust oder Verfälschung. Darum ist es sinnvoll, eine *Konfigurationsdatenbank* zu erstellen, die direkt aus den Arbeitsumgebungen der Entwickler abgefragt werden kann. In der Konfigurationsdatenbank werden alle Informationen zu den Software-Einheiten und zu den definierten Konfigurationen abgelegt. So werden beispielsweise der Status der einzelnen Software-Einheiten und die Beziehungen zwischen ihnen gespeichert. Diese Daten dienen nicht nur zur Generierung von Konfigurationen, sondern für die Projektleitung und die Qualitätssicherung auch als Informationsquelle über den aktuellen Stand.

21.4 Arbeitsbereiche für die Software-Verwaltung

Als die Zahl der Einwanderer in den Vereinigten Staaten gegen Ende des 19. Jahrhunderts immer weiter stieg und mit ihr auch die Zahl derer, die krank oder kriminell waren und dadurch Kosten und Probleme verursachten, wurden sie (soweit sie über New York reisten) nicht sofort ins Land gelassen, sondern nach Ellis Island gebracht, wo sie befragt und untersucht wurden. Wer den Anforderungen nicht genügte, wurde zurückgeschickt. Dieses harte, aber wirksame Verfahren ist auch bei der Verwaltung der Software-Einheiten sinnvoll.

21.4.1 Die Bereiche und ihr Zusammenspiel

Zwischen die Entwicklung und die Auslieferung der Software ist die von der Konfigurationsverwaltung beherrschte *Referenzumgebung* gesetzt. Sie sorgt dafür, dass nur geprüfte Software-Einheiten als Systembausteine verwendet werden.

Abbildung 21–5 zeigt das Prinzip. Ein Entwickler erstellt, modifiziert und überprüft Software-Einheiten in seiner persönlichen *Konstruktionsumgebung*. Fertige Einheiten übergibt er der *Referenzumgebung*. Dort werden neue oder veränderte Software-Einheiten mit dem Status »ungeprüft« versehen und gespeichert. Sie sind damit der Konfigurationsverwaltung unterstellt, d. h., niemand kann sie dort verändern, auch nicht ihr Entwickler.

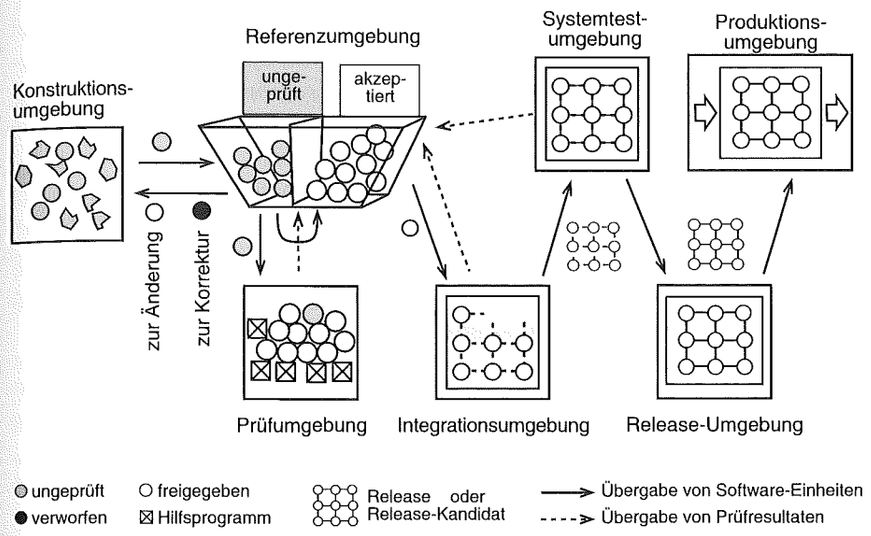

Abb. 21–5 *Die Bereiche der Software-Entwicklung und -Verwaltung*

In der *Prüfumgebung* werden die Software-Einheiten geprüft. Dabei werden andere, bereits geprüfte Einheiten sowie Hilfsprogramme (Testtreiber, Platzhalter) verwendet. Eine Einheit, in der Fehler entdeckt werden, erhält den Status »verworfen«; sie wird nicht weiter verwendet, sondern zur Korrektur an ihren Entwickler zurückgegeben. Werden keine Fehler entdeckt, so wird der Status auf »gut« verändert; diese Einheit kann nun an die *Integrationsumgebung* weitergegeben werden.

Dort wird das System aus guten Einheiten schrittweise zusammengebaut. Jeder Integrationsschritt wird durch einen Test geprüft. Lässt sich eine Einheit nicht fehlerfrei integrieren oder zeigen sich im Test Fehler, ist sie oder eine der bereits integrierten Einheiten nicht in Ordnung. In diesem Fall muss der Fehler lokalisiert werden; die fehlerhafte Einheit bekommt den Status »verworfen«. Erst wenn sie korrigiert und erneut geprüft ist, kann die Integration fortgesetzt werden.

Mit dem letzten Integrationsschritt ist ein vollständiges System entstanden, das auch als Release-Kandidat bezeichnet wird. Der Systemtest erfolgt anschließend in der *Systemtestumgebung*. Fehler, die im Systemtest auffallen, haben die

gleiche Wirkung wie die aus dem Integrationstest. Ist der Release-Kandidat ohne Befunde getestet, so wird daraus ein Release, das an die *Release-Umgebung* geht.

Meist werden dort auch freigegebene Updates (siehe Abschnitt 22.2.3) für ältere Releases bereitgestellt. Jede Software-Lieferung erfolgt aus der Release-Umgebung. Das Release wandert dann in die *Produktionsumgebung*, wo es produktiv eingesetzt wird. Im Fall eines Auftragsprojekts liegt diese Umgebung beim Kunden.

Durch Werkzeuge und Organisation ist sichergestellt, dass Einheiten nur auf die oben beschriebene Weise, also vom Entwickler und nach Prüfung ohne Beanstandung, in den »sauberen« Bereich der Referenzumgebung, die Sammlung guter Einheiten, gelangen kann. Aus anderen Umgebungen fließen nie Einheiten in die Referenzumgebung zurück.

Neue oder geänderte Anforderungen machen es immer wieder notwendig, bereits freigegebene Software-Einheiten zu verändern. Es entstehen neue Versionen, die ebenfalls den beschriebenen Weg durchlaufen.

21.4.2 Realisierung der Arbeitsbereiche

In den beschriebenen Umgebungen werden Werkzeuge eingesetzt. Die Konstruktionsumgebung besteht aus einer Vielzahl von Werkzeugen, die zur Analyse, zum Entwurf, zur Codierung und zur Fehlersuche verwendet werden.

In der Referenzumgebung wird immer ein Werkzeug zum Verwalten der Versionen, Varianten und Konfigurationen verwendet. Es realisiert die Konfigurationsdatenbank und kontrolliert den Zugriff auf darin abgelegte Software-Einheiten. Ein Werkzeug zur Verwaltung der Änderungsanträge dient dazu, die Änderungsanträge systematisch zu bearbeiten und Querverweise zwischen diesen und geänderten Software-Einheiten zu verwalten.

In der Integrationsumgebung werden häufig Werkzeuge genutzt, um teilintegrierte Systeme automatisch zusammenzubauen, sowie Werkzeuge, die den Integrationstest automatisiert durchführen. In der Prüfumgebung und in der Systemtestumgebung werden die Werkzeuge eingesetzt, die für die unterschiedlichen Prüfungen und die Verwaltung der Prüfresultate benötigt werden. Die Release-Umgebung kann im einfachsten Fall durch eine Ordnerstruktur realisiert werden, in der die freigegebenen Releases abgelegt werden. Häufig kann auf Releases nur über eine spezielle (Web-)Anwendung zugegriffen werden, die den Zugang kontrolliert.

Es sollte aber klar sein, dass viele Probleme des Software Engineerings nicht durch Werkzeuge gelöst werden, sondern durch Methoden und organisatorische Maßnahmen. Die Konfigurationsverwaltung bedarf der Organisation; Werkzeuge sind nützlich, wenn die Organisation funktioniert.

22 Software-Wartung

Die große Bedeutung der Wartung ergibt sich schon daraus, dass sie, verglichen mit den Schritten der Entwicklung, das mit Abstand größte Budget erfordert. Zudem ist ein (erfolgreiches) Software-System im größten Teil seiner Existenzdauer in Wartung.

Der relativ geringe Umfang dieses Kapitels sagt also mehr über unser Wissen als über die Bedeutung der Wartung aus; bis heute ist die Wartung ein weißer Fleck auf der Landkarte. Der Grund ist sehr einfach: Können wir uns bei der Entwicklung die Verhältnisse (mit Einschränkungen) so gestalten, wie wir sie uns wünschen, müssen wir bei der Wartung alles akzeptieren, wie es ist. In diesem Sinne ist die Wartung der Geriatrie vergleichbar, die sich mit den unterschiedlichsten Krankheitsbildern alter Menschen auseinandersetzen muss, auch mit Krankheiten, die die Patienten selbst herbeigeführt haben, mit sehr seltenen Kombinationen von Krankheiten und vielen unheilbaren Fällen.

22.1 Begriff und Taxonomie der Software-Wartung

22.1.1 Verschleiß und Wartung

Die Wartung eines Flugzeugs ist planbar; durch Berechnungen oder aus Erfahrung ist bekannt, dass bestimmte Teile nach einer gewissen Zeit ihre Eigenschaften verändert haben oder nach einer gewissen Flugstrecke abgenutzt sind. In der periodischen Wartung werden sie darum geprüft oder gleich ersetzt.

Dagegen ist die Reparatur eines Flugzeugs nach einem Zusammenstoß mit einem Vogelschwarm nicht planbar. Man kann zwar, da es Statistiken über solche Unfälle gibt, ein Budget für diese Reparaturen vorsehen, doch der einzelne, konkrete Unfall kommt immer überraschend. Die Umrüstung eines Flugzeugs für einen anderen Einsatz, z. B. für die Beförderung von Fracht statt Menschen, ist zwar das Resultat einer bewussten Entscheidung, aber diese Entscheidung wird typischerweise kurzfristig, z. B. nach einem Wechsel des Eigentümers, getroffen; darum kann auch diese Arbeit nicht langfristig, bereits in der Entwicklung, ge-