FRED BROOKS, der «Vater» des IBM-Betriebssystems OS/360, hat vor Jahren vor der Suche nach der «Silver Bullet» gewarnt — einer Methode, mit der die Werwölfe der Software-Entwicklung schlagartig aus dem Weg geräumt werden können. Wiederverwendung ist keine solche Wunderwaffe. Sie ist aber Bestandteil einer umfassenden Strategie für das Software-Engineering, mit der Qualität und Produktivität schrittweise verbessert werden können.

Konfigurationsmanagement für die Software-Wiederverwendung

Wiederverwendung als Wunderwaffe?

Von Erhard Plödereder

Die Wiederverwendung von Software wird derzeit als geeignete Methode angesehen, dem Anstieg der Komplexität und der Entwicklungskosten entgegenzuwirken. Eine erfolgreiche Wiederverwendung verspricht klare Vorteile:

- Kostenminderung, da weniger Software neu entworfen, erstellt und getestet werden muß
- Verkürzte Entwicklungsdauer und damit schnellere Marktpräsenz, die erheblichen Einfluß auf das Verkaufsvolumen haben kann
- Höhere Software-Qualität, da die Software unter verschiedenen Einsatzbedingungen schneller ausreift und Verbesserungen oder Korrekturen mehreren Produkten zugute kommen.

Trotz der genannten Anreize beginnt die Wiederverwendung nur langsam Fuß zu fassen. Dafür sind diverse Faktoren verantwortlich, zum Beispiel:

- Rechtliche Probleme: Urheberrechte, Lizenzen, Gewährleistung und Haftung schaffen eine Reihe von Fragen bei Wiederverwendung firmenfremder Software, die bei der Eigenentwicklung nicht auftreten. Darum wird die Wiederverwendung von Software, die kommerziell oder frei verfügbar ist, oft als zu hohes Risiko betrachtet.
- Vertragliche Hindernisse: Bei einer nach Aufwandsentschädigung erfolgenden Software-Entwicklung besteht häufig nicht die Möglichkeit, die durch Wiederverwendbarmachung und Wiederverwendung entstehenden Kosten in Rechnung zu stellen. Damit wirkt sich die Software-Wiederverwendung sogar nachteilig auf die Firmenbilanz aus.
- NIH-(«not invented here»-)Syndrom: Unsere Software-Entwickler stehen fremder Software oft – und nicht immer zu Unrecht – sehr skeptisch gegenüber. Anstatt sich in solche Software einzuarbeiten, ziehen sie es vor, sie nach ihren eigenen Vorstellungen neu zu entwickeln. Die Aussicht, eventuell auftretende Fehler in ihnen unbekannter Software finden und korrigieren zu müssen, ist abschreckend. Demzufolge wird Software zur Wiederverwendung nur akzeptiert, wenn sie hinlänglich bekannt ist oder Kosten und Zeitbedarf einer Neuentwicklung nicht vertretbar sind.
- Technische Probleme: Welche Charakteristika machen Software wiederverwendbar? Wie ist die Infrastruktur zu gestalten, damit gesuchte Komponenten leicht gefunden werden können, falls sie vorhanden sind? Welche Änderungen im Entwicklungsprozeß sind nötig, um die Wiederverwendung zu erleichtern? Und – das Thema dieses Beitrags – welche Wechselwirkungen entstehen mit der Konfigurationsverwaltung für die Produkte, die wiederverwendete Software enthalten?

Trotz der genannten Schwierigkeiten gibt es Branchen der Software-Industrie, für die eine ausgeprägte Wiederverwendung interner Software-Komponenten der Regelfall, zum Teil sogar zur Voraussetzung der Konkurrenzfähigkeit geworden ist. Typischerweise trifft dies auf produktorientierte Firmen zu, deren Produkte gemeinsame Systemstrukturen aufweisen, so daß sie als Ausprägungen aus wiederverwendbarer Software erzeugt werden können.

Ein Beispiel ist die Produktion von Compilersystemen für verschiedene Programmiersprachen, unterschiedliche Wirts- und Zielrechner und diverse Anwendungsbereiche. Solche Produktgruppen werden heute von nahezu allen Firmen im wesentlichen durch Rekombination wiederverwendbarer Komponenten erstellt. Ein Compilersystem, einschließlich der peripheren Werkzeuge, für eine komplexe Programmiersprache (zum Beispiel Ada, C++) kann die Größenordnung einer halben Million Quellzeilen erreichen. Um die Entwicklungskosten zu amortisieren, sind entweder hohe Verkaufszahlen eines einzelnen Compilers oder die weitgehende Wiederverwendung der Software in mehreren Produkten notwendig.

Die folgende Diskussion zum Konfigurationsmanagement für wiederverwendbare Software ist größtenteils aus der zehnjährigen Erfahrung einer Firma abgeleitet, die solche Übersetzersysteme erstellt und als Produkte vermarktet.

Charakterisierung der wiederverwendeten Software

Wir sprechen hier hauptsächlich von der Wiederverwendung relativ großer, für eine bestimmte Klasse von Systemen anwendbarer Programmeinheiten, zum Beispiel Compiler-Front-Ends für diverse Programmiersprachen, Codegeneratoren für Klassen von Zielarchitekturen, zielrechnerabhängige und -unabhängige Code-Optimierungen usw. Nur in den wenigsten Fällen kann diese Software ohne weitere Anpassungen wiederverwendet werden. Die spezielle Ausprägung eines Produkts wird über geeignete Parametrierung erreicht, in Form charakteristischer Werte oder Entscheidungsalgorithmen, auf die von der wiederverwendbaren Software Bezug genommen wird.

Eine schwierige Entwurfsentscheidung ist dabei die pragmatische Trennung der Parametrierung von der wiederverwendbaren Software. Je mehr spezifische Anpassung durch Parametrierung erreicht werden kann, desto breiter ist das Anwendungsspektrum der Software; allerdings wächst auch der Aufwand, der bei der Anpassung dieser Software für jedes Produkt entsteht. Selbstverständlich muß die Implemen-

ERHARD PLÖDEREDER, Universität Stuttgart, Institut für Informatik, Breitwiesenstraße 20–22, D-W-7000 Stuttgart 80.

tierung der wiederverwendbaren Software strikt von ihrer jeweiligen produktspezifischen Parametrierung getrennt werden.
In der Praxis wird die Wiederverwendung der Software in einem geänderten Umfeld häufig zu zusätzlicher Parametrierung und weiterer Evolution der wiederverwendeten Software Anlaß geben. Nach unserer Erfahrung ist es nicht sinnvoll oder möglich, alle Möglichkeiten der Parametrierung bereits im ersten Entwurf der Software einzuplanen. Überspitzt ausgedrückt: Die Software wird erst bei mehrfacher Wiederverwendung durch Evolution tatsächlich wiederverwendbar.
Die Wiederverwendung kleiner Software-Pakete, zum Beispiel die Implementierung häufig benötigter abstrakter Datentypen oder einer Sammlung numerischer Algorithmen, stellt aus der Sicht des Konfigurationsmanagements keine besondere Herausforderung dar, da diese Pakete in den wenigsten Fällen einer nennenswerten Evolution unterliegen. Ihre erfolgreiche Wiederverwendung hängt von anderen Faktoren (Verfüg- und Auffindbarkeit, Akzeptanz, Flexibilität) ab, auf die wir hier nicht näher eingehen wollen.

Konfigurationsmanagement in der Entwicklung

Die Mehrzahl kommerziell verfügbarer Systeme zur Unterstützung des Konfigurationsmanagements ist darauf ausgerichtet, die Neu- und Fortentwicklung von Software zu unterstützen, die laufenden Veränderungen zu verfolgen, Konfliktsituationen bei der Änderung durch mehrere Entwickler zu vermeiden und mechanische Konstruktionsschritte (zum Beispiel Übersetzung) zu automatisieren. Dagegen wird der Aufbau von Konfigurationen aus existierenden Subsystemen oft nicht speziell unterstützt; gerade dies ist jedoch bei der Integration großer Systeme auch aus vorhandener, also wiederzuverwendender Software ein wichtiger Gesichtspunkt.
Tichy illustriert die Notwendigkeit des Konfigurationsmanagements am Beispiel eines kleinen, aus 100 Dateien bestehenden Systems, wobei jeweils zwei Versionen jeder Datei existieren. Daraus ergeben sich rechnerisch 2^{100} mögliche Konfigurationen des Systems, mehr als es Sekunden gab, seit das Universum entstanden ist. Gruppiert man diese Dateien aber in Subsysteme, zum Beispiel fünf Subsysteme aus je 20 Dateien, und verwaltet man jeweils zwei Versionen der Subsysteme, dann sind nur noch $2^5 = 32$ Systemkonfigurationen möglich. Dieses Rechenexempel illustriert, daß eine Aggregierung der Software in Subsysteme zur Bewältigung der Komplexität bei der Komposition von Systemen aus existierender Software unabdingbar ist. Kernforderungen an das Konfigurationsmanagement müssen daher sein,

- daß Subsysteme klar identifizierbar sind
- daß die detaillierte Verfolgung von Veränderungen innerhalb der Subsysteme erfolgt und zum geeigneten Zeitpunkt zu einer Versionierung dieser Subsysteme führt
- daß die Systemkomposition auf der Ebene der Subsystemversionen erfolgt
- daß Subsysteme Bestandteil mehrerer (Sub-)Systeme sein können, ohne daß sie dazu dupliziert werden müssen

Die wesentliche Frage bei der Integration von Subsystemen ist, neben der relativ einfach zu entscheidenden Vollständigkeit, die Konsistenz dieser Subsysteme.
Während relativ leicht feststellbar ist, ob bei der Integration von Subsystemen Vollständigkeit erreicht wurde, läßt sich die Konsistenz der Teile nur schwer überprüfen. Unter Konsistenz verstehen wir hier ein korrektes funktionales Zusammenwirken der Subsysteme. Bei der Neuentwicklung eines Systems ist die Konsistenz der Subsysteme durch Entwurf und Entwicklung impliziert, so daß die Systemintegration diesbezüglich weitgehend problemlos ist. Bei der Komposition aus wiederverwendbaren Subsystemen ist die Situation sehr viel schwieriger, da hier mit den Gegebenheiten existierender Software-Komponenten gearbeitet werden muß, die im Integrationsschritt erstmals zusammentreffen. Häufig tritt dabei der Fehler auf, daß das Gesamtsystem aus inkonsistenten Versionen gebaut wird. Eine mögliche Ursache dafür ist die unvollständige Adaption der Beschreibung einer ähnlichen, bereits existierenden Konfiguration. Ein Teil dieser Fehler kann bei der Integration zwar frühzeitig diagnostiziert werden (zum Beispiel durch Schnittstellenprüfungen in Programmiersprachen wie Ada oder Werkzeugen wie «lint»), aber Fehler, die sich aus nicht zusammenpassender Funktionalität ergeben, werden – wenn überhaupt – erst im ausgiebigen Test erkannt. Es ist hier ein Defizit an Spezifikationsmethoden und Werkzeugen anzumerken, die sich mit der Beschreibung der funktionalen Abhängigkeit von Software-Komponenten befassen und bei der Auswahl konsistenter Komponenten Hilfestellung geben.
Speziell ist die folgende Situation von größtem Interesse: Die Komponente X soll durch eine andere Version ersetzt werden. Welche weiteren Komponenten müssen aufgrund dieser Änderung ebenfalls durch neuere Versionen ersetzt werden, um zu einem konsistenten Gesamtsystem zu kommen? Welche mittelbaren Folgen haben diese Modifikationen?
Werkzeuge für Konfigurationsmanagement, wie sie heute zur Verfügung stehen, bieten für diese zentrale Frage keine Antwort. Aber gerade hier liegt die häufigste Fehlerquelle bei der Systemkonfigurierung, da das Wissen über bestehende Abhängigkeiten und Konsistenz oft nicht zentral verfügbar ist, sondern nur durch Befragung der Entwicklungsgruppen aus Teilinformationen zusammengestellt werden kann.
Eine weitere Fehlerquelle ist der Versionskonflikt («version skew»). Er tritt auf, wenn zwei Komponenten, A und B, ihrerseits eine dritte Komponente C benötigen, wobei A mit Version 1 von C und B mit Version 2 von C entwickelt (und ausgiebig getestet) wurde, die Versionen von C aber nicht untereinander austauschbar (kompatibel) sind. Sollen nun diese bisher unabhängig verwendeten Komponenten A und B Bestandteil desselben Systems werden, das aber nur eine Version von C enthalten kann, ist ein Fehlverhalten des Systems im Wortsinne vorprogrammiert, wenn der Versionskonflikt nicht vor der Systemgenerierung erkannt wird. Meist ist es sehr schwierig, einen solchen Konfigurationsfehler als Ursache des Systemverhaltens zu erkennen, zumal die Entwickler der Komponenten A, B und C aufgrund ihrer jeweiligen Erfahrungen von der Korrektheit ihrer Software überzeugt sind.
Die Kosten eines Konfigurationsfehlers können erheblich sein. Nach der aufwendigen Fehlersuche kann seine Beseitigung einen mehrtägigen Produktionsprozeß nach sich ziehen. Mehr noch: Wird der Fehler erst im Rahmen eines mehrere Wochen dauernden Systemtests gefunden, so kann sich die Auslieferung des Produkts deutlich verzögern, da die Veränderung gravierend genug ist, um den Systemtest von neuem beginnen zu müssen. Es besteht weiter die Gefahr, daß die Beseitigung eines Fehlers neue Fehler hervorruft und daher der Produktionsprozeß lange Zeit durch eine Serie von Korrekturzyklen belastet ist.
Nach unserer Erfahrung sind – mit der seltenen Ausnahme zu spät erkannter grober Entwurfsfehler in der Software – Konfigurationsfehler der Hauptgrund für erheblich reduzierte Produktivität in der Software-Entwicklung. Es ist daher von überragender Bedeutung, Konfigurationsbeschreibungen frühzeitig zu ent-

wickeln, mit größter Sorgfalt zu prüfen und generell die Aufmerksamkeit der Entwicklungsteams auf die Einhaltung der Konfigurationspläne und -methoden zu richten.

Einfluß der Wartung auf das Konfigurationsmanagement

Gemäß genereller Erfahrungen der Industrie belaufen sich die nach der ersten Produktauslieferung anfallenden Wartungs- und Evolutionskosten auf 60 bis 80 % der Gesamtkosten des Software-Produkts. Dementsprechend muß sich eine Strategie der Wiederverwendung speziell auch dazu eignen, diese Kosten zu reduzieren. Wiederverwendete Software unterliegt ebenfalls einer Evolution durch Fehlerbeseitigung und Erfüllung veränderter, steigender Anforderungen. Wird die Wiederverwendung nur in der Entwicklungsphase gefördert, aber nicht in der Wartung, so daß mehrere Kopien unkoordiniert bearbeitet werden, dann wird das vorhandene Potential zur Kostensenkung und zur Qualitätserhöhung nur schwach ausgenutzt. Die wiederverwendete Software sollte einer eigenständigen, produktübergreifenden Wartung und Evolution unterliegen, so daß jede Verbesserung allen Produkten zugute kommt.

An diesem Punkt entstehen neue Forderungen an das administrative und technische Konfigurationsmanagement, das nun nicht mehr ausschließlich produktspezifisch betrieben werden kann. Vielmehr muß gewährleistet werden, daß die Fortentwicklung wiederverwendeter Software mit der Konfigurierung und Evolution aller betroffenen Produkte koordiniert wird.

Im technischen Bereich steigt die Motivation, die Evolution wiederverwendbarer Komponenten aufwärtskompatibel zu halten, so daß bei der Eingliederung in die diversen Produktrevisionen Integrationsarbeiten auf ein Minimum gesenkt werden können. Verbesserungen, die zwangsläufig zu Inkompatibilitäten führen, sollten zeitlich so zusammengefaßt werden, daß die Anpassung anderer Komponenten im Block erfolgen kann. In der Praxis zeigt sich, daß Änderungen, die mehrere Subsysteme betreffen, durch die nötige Koordination des Entwurfs, der Entwicklung und der Zeitpläne ungleich langwieriger sind als Änderungen, die innerhalb eines Subsystems vorgenommen werden können. Im administrativen Bereich sind Produkt- und Arbeitsplanung betroffen. Werden Komponenten revidiert, die in mehreren Produkten eingesetzt sind, so müssen die Produktionspläne der einzelnen Produkte aufeinander abgestimmt sein, so daß deren Integration nicht durch Abhängigkeiten verzögert wird. Zum Beispiel ist es nicht sinnvoll, eine neue, von Front-End-Daten abhängige Optimierung in einen Compiler zu integrieren, bevor diese Daten von einem revidierten Front-End bereitgestellt werden. Hier muß produktübergreifende Information des Konfigurationsmanagements in die Arbeitsplanung eingehen.

Vor allem sollte möglichst vermieden werden, daß aktiv vermarktete Produkte auf verschiedenen Versionen der wiederverwendeten Software basieren. Andernfalls werden die Benutzerbetreuung, die Analyse der im Betrieb aufgetretenen Fehler und die Einbringung von Korrekturen in die jeweilige interne Entwicklungsversion erheblich erschwert. Im Idealfall werden zu jedem Zeitpunkt nur zwei Versionen jeder Komponente global eingesetzt: die «eingefrorene» Referenzversion, die Bestandteil bereits ausgelieferter Produkte ist, und die Entwicklungsversion, in der Fehlerkorrekturen und funktionale Verbesserungen bis zur nächsten Produktauslieferung akkumuliert werden. Natürlich werden auch lokale Versionen der Subsysteme temporär existieren, in denen die eigentlichen Entwicklungen betrieben werden, bis diese den vorgesehenen Qualitätsgrad erreicht haben, um in die Entwicklungsversion übertragen zu werden.

Dieses Modell läßt sich nicht immer durchsetzen; bisweilen ist es unumgänglich, ein Produkt durch eine neuere Version mit minimalen (aber wichtigen) Unterschieden zu ersetzen. Aus diversen Gründen kann hier oft die bereits weiter fortgeschrittene Entwicklungsversion nicht eingesetzt werden. Das Konfigurationsmanagement für ausgelieferte Produkte muß daher in der Lage sein, die jeweilige Konfiguration mit minimalen Differenzen von der Referenzversion zu modifizieren und dafür zu sorgen, daß hier korrigierte Fehler auch in der Entwicklungsversion beseitigt werden. Diese «Delta»-Konfigurationen sind ein weiterer Grund dafür, aktive Produkte nicht auf verschiedenen Versionen wiederverwendeter Software aufzubauen.

Wechselwirkung mit der Firmenorganisation

Die Firmenorganisation beeinflußt die Strategie des Konfigurationsmanagements und damit auch die Erfolgsaussichten der Software-Wiederverwendung. Eine Strategie, die der Interessenlage des Mid-Managements zuwiderläuft, wird kaum erfolgreich sein.

Spiegelt die Firmenorganisation die einzelnen Produktlinien, so tendiert das Konfigurationsmanagement dazu, produktspezifisch zu sein. Die Kontrolle über die Evolution wiederverwendeter Komponenten ist damit zunächst der einzelnen Produktlinie überlassen. Die Wiederverwendung von Software beschleunigt den Entwicklungsprozeß. Es zeigt sich dann aber die Tendenz, die Evolution produktspezifisch voranzutreiben, ohne spezielle Rücksicht auf die Tatsache, daß die Software damit von einer Basisversion divergiert. Dies bringt zunächst Zeitvorteile, da Erwägungen, die Komponente wiederverwendbar zu erhalten oder Veränderungen mit den Zeitplänen anderer Produkte zu koordinieren, nicht im Vordergrund stehen. Zudem können Veränderungen ganz speziell den Bedürfnissen des Produkts angepaßt werden. Dementsprechend gestaltet sich die Produktintegration relativ einfach. Ähnliches gilt auch für die produktspezifische Wartung. Aus der Sicht des Produktverantwortlichen wird hier sehr effizient gearbeitet.

Aus globaler Firmensicht hat diese Entwicklung aber eklatante Nachteile: Die für mehrere Produkte geschaffenen Versionen der wiederverwendeten Software werden nun individuell gewartet. Jede Fehlerkorrektur sollte in all diesen Versionen repliziert und getestet werden, was sowohl kostspielig als auch fehleranfällig ist, insbesondere wenn mehr als zwei oder drei Produkte betroffen sind. Das Kontrollsystem für die Verfolgung aller Korrekturen in allen Versionen wird extrem gefordert. In der Praxis degeneriert das System leicht, so daß nur diejenigen Produkte korrigiert werden, für die die Benutzer Fehler gemeldet hatten. Als Konsequenz divergieren die Versionen der wiederverwendeten Software weiter. Der Vorteil einer produktübergreifenden Qualitätsverbesserung durch Software-Wiederverwertung geht weitgehend verloren. Zudem wird die Wiederverwendbarkeit als Qualität der Software langsam zerstört, so daß die Generierung des nächsten Produkts erst «Säuberungsaktionen» der wiederverwendeten Komponenten erfordert.

In einer Firma, deren Organisation nach Produktlinien ausgelegt ist, werden sich daher die Vorteile der Software-Wiederverwendung in der Wartung und Evolution nur schwer ausnutzen lassen.

Eine Alternative ist die Organisation entlang funktionalen Einheiten der Software, das heißt die Entwicklung und Evolution jedes (wiederverwendbaren) Subsystems wird autonom gemäß den Anforderungen aller Produkte betrieben. Für die Erstellung der Produkte

sind Integrationsteams verantwortlich. Dieses Modell entspricht in etwa der «Component Factory», wie sie in der Literatur vorgeschlagen wird.

In dieser Organisationsform steht die Erhaltung der Subsystem-Wiederverwendbarkeit im Vordergrund. Der Qualitätsverbesserung der einzelnen funktionalen Einheiten, die allen Produkten zugute kommt, wird hier Rechnung getragen. Dem steht als Nachteil gegenüber, daß der Einfluß der Produkterstellung vermindert und nur indirekt wirksam ist. Mißverständnisse über die Details der Anforderungen an die einzelnen Produkte führen bei der Integration der Subsysteme zu Konsistenz- und Funktionalitätsproblemen, die ihrerseits Nachbesserungen in der wiederverwendeten Software nötig machen. Zudem muß die Zeitplanung der funktionalen Einheiten auf die Bedürfnisse der diversen Produktintegrationen abgestimmt werden. In der Praxis ergeben sich hier oft Schwierigkeiten, weil nicht alle funktionalen Einheiten ihre Subsysteme in genügender Synchronisation bereitstellen können, speziell wenn nicht vorausplanbare Nachbesserungen nötig werden. Aus all diesen Gründen ist die Produktintegration langwierig und organisatorisch mühsam, sehr zum gemeinsamen Mißfallen der Produktmanager, Kunden und Firmenleitung. Eine Reorganisation wird nicht lange auf sich warten lassen.

Sowohl die produkt- als auch die funktionalorientierte Organisationsform erzeugen also Schwierigkeiten, die Produktion aus wiederverwendbaren Komponenten effizient zu gestalten. Eine Integration der Organisationsformen hat bessere Aussichten, das Ziel zu erreichen.

Nachdem wir unsere (schmerzlichen) Erfahrungen mit den geschilderten Organisationsformen gemacht hatten, hat sich eine matrixorientierte Organisationsform herausgebildet, in der die Software-Entwickler duale Verantwortlichkeit gegenüber der Produktentwicklung und den funktionalen Einheiten haben. Die Verantwortlichkeit für den Kompositionsaspekt des Konfigurationsmanagements liegt primär bei der Produkterstellung; die Fortschreibung der einzelnen Subsysteme erfolgt aber durch die funktionalen Einheiten. Durch entsprechende temporäre Personalzuordnung aus funktionalen Gruppen an die Produktintegration wird erreicht, daß Produktanforderungen direkt zum Empfänger kommen. Die permanente Zuordnung des Personals an funktionale Einheiten stellt sicher, daß die Versionsbildung wiederverwendbarer Komponenten nicht ausufert.

Qualitätskriterien für wiederverwendbare Software

Ein interessanter Aspekt des Konfigurationsmanagements für wiederverwendete Komponenten ist die Frage, welche Qualitätskriterien anzusetzen sind, um die Komponente für die Generierung einer Produktrevision freizugeben. Zunächst erscheint es als völlig ausreichend, wenn die Komponente die Erfordernisse des Produkts erfüllt und dies durch den Systemtest des Produkts nicht widerlegt wird. Darüber hinausgehende Forderungen an die Komponente zu stellen, erscheint widersinnig.
Leider fördert dieses Kriterium einen Effekt, den wir als «Versionsdrift» bezeichnen: Mit der Auslieferung des Produkts sollte die Komponente als Referenzversion eingefroren werden. Bei der Generierung der Revision des nächsten Produkts stellt sich dann aber meistens heraus, daß weitere Änderungen dieser Referenzversion nötig sind, um auch diesem Produkt gerecht zu werden. Im Laufe des Revisionszyklus einer Produktlinie entsteht nun eine Referenzversion zu jedem Produkt. Zwar enthalten die jeweils späteren Versionen alle vorangegangenen Veränderungen, aber das Konfigurationsmanagement für die ausgelieferte Software und die Benutzerbetreuung wie auch die eventuelle spätere Generierung von «Delta»-Versionen werden trotzdem durch die Vielzahl dieser Versionen erschwert.
Daher sollten die Qualitätskriterien für die Freigabe einer neuen Version der wiederverwendeten Software so angesetzt werden, daß die Komponente die Erfordernisse aller Produkte erfüllt, die mit dieser Software revidiert werden sollen. Als Nachweis dienen die einschlägigen Teile des Systemtests dieser Produkte. Damit wird eine Versionsdrift weitgehend verhindert. Voraussetzung ist allerdings, daß die Evolution all dieser Produkte parallel vorangetrieben wird, so daß die diesbezüglichen Systemtests auch tatsächlich möglich sind.

Mehr als nur die Bereitstellung wiederverwendbarer Komponenten

Erfolgreiche Software-Wiederverwendung erfordert mehr als nur die Schaffung und die Bereitstellung wiederverwendbarer Komponenten. Der gesamte Lebenszyklus der Produkte, die mit solchen Komponenten erstellt werden, wird beeinflußt. Speziell wirft die Wiederverwendung gravierende Fragen nach der optimalen Form des Konfigurationsmanagements auf. Die Prozesse der Konfigurations-, Produkt- und Arbeitsplanung werden komplexer und bedürfen weit größerer Aufmerksamkeit. Unter Umständen ist sogar die Organisationsstruktur des Entwicklungsbereichs betroffen.
Der Aufwand, der zur Lösung dieser Komplikationen notwendig ist, wird aber mehr als ausgeglichen durch die Vorteile, die von der Software-Wiederverwendung ausgehen. Wenn eine Firma Produkte mit einem Gesamtvolumen von zehn Millionen Quellzeilen im aktiven Betrieb hat, aber nur Software im Umfang von ein bis zwei Millionen Quellzeilen zu warten hat, dann kann man davon ausgehen, daß sich dieser Aufwand lohnt. [13]