Repositories und Software-Wiederverwendung

Udo Kelter

01.07.2001

Zusammenfassung dieses Lehrmoduls

Software-Wiederverwendung wird als ein entscheidendes Mittel angesehen, die Entwicklungszeit und -Kosten von Software zu senken und die Qualität zu erhöhen. Dieses Lehrmodul analysiert zunächst verschiedene Arten von Wiederverwendung, die möglichen Kosteneinsparungen und an der Wiederverwendung orientierte Vorgehensmodelle. Ein wesentlicher Aspekt der Wiederverwendung ist die Suche nach geeigneten Komponenten; diese sollte durch (Komponenten-) Repositories unterstützt werden.

Vorausgesetzte Lehrmodule:

obligatorisch:
- Integrationsrahmen für Software-Entwicklungsumgebungen
- Software-Entwicklungsumgebung

empfohlen:
- Transportdateien und die XML

Stoffumfang in Vorlesungsdoppelstunden: 1.0

Inhaltsverzeichnis

1 Wiederverwendung

1.1 Motivation

Wiederverwendung ist eines der Dauerthemen der Softwaretechnik. Die grundlegende Idee besteht darin, Produkte möglichst weitgehend aus vorgefertigten Standard-Komponenten zu konstruieren und so die individuelle Neuentwicklung von Komponenten zu vermeiden. Hiervon verspricht man sich folgende Vorteile:

- Kostensenkung, primär infolge einer signifikanten Erhöhung der Produktivität der Entwickler
- Erhöhung der Qualität des Produkts, da die Komponenten infolge des wiederholten Einsatzes ausgereifter sind als neue Software
- kürzere Entwicklungszeit

Die Verwendung von vorgefertigten Standard-Komponenten ist auch in anderen Industrien üblich. So wird bspw. ein Automobilhersteller nicht selbst Reifen, Autoradios, Lichtmaschinen, Benzinpumpen, Sitze, Scheinwerfer, Schrauben, Autolacke usw. herstellen, sondern zukaufen, eventuell sogar ganze Motorblöcke. Der Hersteller des Autoradios wird wiederum die elektrischen Bauteile (Transistoren, ICs, Kondensatoren usw.) oder Bleche für das Gehäuse nicht selbst herstellen, sondern zukaufen.

1.2 Wiederverwendung vs. technologische Arbeitsteilung

Eine genauere Betrachtung dieser Beispiele zeigt allerdings, daß es sich oft nicht wirklich um Wiederverwendung handelt. Es kann durchaus sein, daß z.B. für einen neuen PKW-Typ auch völlig neue Sitze, Scheinwerfer, Benzinpumpen und Lacke benötigt werden und daß diese Komponenten in keinem anderen PKW-Typ eingesetzt werden. Die Fremdherstellung dieser Komponenten ist trotz fehlender Wiederverwendung sinnvoll:

- Für die Entwicklung dieser Komponenten kann ein eigenes, umfangreiches Know-How erforderlich sein, das nicht zu den Kernkompetenzen des Anbieters gehört. Eine eigene Entwicklungsabteilung wäre aber wegen der geringen Zahl der Entwicklungen nicht ausgelastet.
- Die Herstellung der Produkte kann bei kleinen Mengen infolge von Skaleneffekten unwirtschaftlich sein, wenn z.B. teure Produktionsanlagen installiert werden müßten, die ansonsten nicht ausgelastet werden, oder nur geringe Materialmengen zu relativ hohen Preisen gekauft werden müßten.

Die Fremdherstellung von Komponenten entspringt hier dem generellen Prinzip der industriellen Arbeitsteilung, wonach einzelne Marktteilnehmer sich auf Produkte oder Dienstleistungen spezialisieren, für die spezielle, vertiefte Kompetenzen erforderlich sind; letztlich wird eine geringe Fertigungstiefe angestrebt. Das Prinzip der Arbeitsteilung kann auch auf einzelne Schritte eines Produktionsprozesses angewandt werden (Halbfabrikate).

Komponenten wie Lichtmaschinen, Autoradios, Schrauben oder Reifen werden zwar in dem Sinne wiederverwendet, daß die gleichen Komponenten (im Sinne von Typen) in verschiedenen Wagentypen eingesetzt werden, aber auch hier dürfte das allgemeine Prinzip der industriellen Arbeitsteilung das entscheidende Argument für die Fremdherstellung sein. Eine Wiederverwendung derartiger Komponenten ist vor allem dann wahrscheinlich, wenn durch einen überschaubaren Katalog an Standard-Modellen ("DIN-Schrauben"), für die idealerweise sogar eine Norm vorliegt, der überwiegende Teil der Nachfrage abgedeckt werden kann. Wiederverwendung im eigentlichen Sinne tritt hier eher beim Anbieter der Standard-Komponenten auf: dieser verwendet die gleichen Technologien und Kompetenzen zur Entwicklung aller Standard-Komponenten.

Vielfach wird beklagt, die Informatik hinke, was das Angebot an wiederverwendbaren Komponenten und das Ausmaß von deren Einsatz betrifft, weit hinter anderen Industrien her. Derartige Aussagen sind aber so pauschal aber nicht haltbar:

- Skaleneffekte durch große Produktionsmengen und Losgrößen treten bei der "Herstellung" von Softwareprodukten nicht auf, diese Motivation für den Zukauf von Komponenten entfällt völlig.
- Technologische Arbeitsteilung und dadurch induzierter "Zukauf" von Komponenten ist auch in der Informatik gang und gäbe. Kein normaler Entwickler käme auf die Idee,

  - Netzwerkprotokolle,
  - Treiber für Graphik- oder Soundkarten,
  - ein Fenstersystem oder elementare Graphikfunktionen,
  - ein Datenbankmanagementsystem,
  - komplexe mathematische Funktionen,
  - einen Transaktionsmonitor,
  - Verschlüsselungsverfahren

  usw. selbst zu entwickeln, obwohl diese letztlich einen Teil des entwickelten Systems ausmachen. Der Hauptgrund für den "Zukauf" dieser Komponenten liegt darin, daß der Entwickler fachlich nicht in der Lage ist, sie selbst zu entwickeln, es liegt hier also eine technologische Arbeitsteilung vor.

  Derartige Komponenten werden oft nicht als solche wahrgenommen, weil sie auf dem Zielrechner schon als Teil des Betriebssystems und anderer vorausgesetzter Basissoftware vorhanden sind und keine expliziten Kopien gemacht oder Lizenzgebühren bezahlt werden müssen. Wahrgenommen werden diese Komponenten nur dann, wenn sie in Form von Bibliotheken oder Frameworks als Teil der Entwicklungsumgebung sichtbar sind.

Auf Wiederverwendung, die in erster Linie durch technologische Arbeitsteilung motiviert ist, gehen wir i.f. nicht näher ein, weil sie ohnehin selbstverständlich ist.

Wir konzentrieren uns stattdessen auf solche Formen der Wiederverwendung, wo mehrere einander ähnliche Systeme in der gleichen Technologie entwickelt werden müssen. Dies beinhaltet den Fall, daß ein Katalog von Standard-Komponenten entwickelt bzw. benutzt wird.

Die Systeme, in denen Komponenten wiederverwendet werden, können durchaus in verschiedenen Zeiträumen entwickelt werden. Hierbei ist weniger an das Erstellen mehrerer Revisionen des "gleichen" Systems gedacht, sondern an unabhängig voneinander existierende Systeme (allerdings ist diese Unterscheidung unscharf).

Einen Sonderfall stellt die Entwicklung einer Systemfamilie dar, weil hier alle Systeme gemeinsam entwickelt werden und gemeinsame - also wiederverwendbare - Komponenten von vorneherein gezielt bestimmt werden können, während normalerweise bei der Realisierung einer Komponente nur darüber spekuliert werden kann, ob sie später noch einmal wiederverwendet werden wird.

1.3 Typen wiederverwendbarer Dokumente

Wiederverwendung ist im Prinzip bei beliebigen Software-Artefakten bzw. Dokumenttypen und in beliebigen Entwicklungsstufen denkbar:

- Anforderungen
- Quelltexte oder ausführbarer Code (Bindemodule) von Programmen
- Modul-/API-Spezifikationen
- Architekturen bzw. Architekturfragmente
- Datenbankschemata
- Gestaltungselemente von GUIs
- Testfälle
- Dokumentation: Bedien-, Installationshandbücher, Glossare etc.

Artefakte verschiedenen Typs können zusammenhängen (z.B. bei einer GUI-Komponente: Code, Gestaltung, Hilfesystem, Manual) und dann sinnvollerweise nur gemeinsam wiederverwendet werden.

In der Praxis liegt der Schwerpunkt heute nach wie vor bei Quelltexten, Architekturen, API-Spezifikationen und Datenbankschemata.

Eine besondere Art der Wiederverwendung von Architekturen bzw. Architekturfragmenten stellen Entwurfsmuster dar: hier wird eher die Erfahrung beim Architekturentwurf wiederverwendet, da die Muster i.d.R. nicht unverändert in das letztliche Produkt übernommen werden.

2 An Wiederverwendung orientierte Entwicklungsprozesse

2.1 Grundform

Wiederverwendung ist wie schon erwähnt bei beliebigen Dokumenttypen denkbar, am häufigsten aber bei Architekturen und Programmen[1]. Wir erläutern die Entwicklungsprozesse daher an diesem Beispiel. Der grundlegende Entwicklungsprozeß hat folgende Form:

1. Bilden der (Grob-) Architektur des Systems
2. Suche nach geeigneten Komponenten in einem Vorrat wiederverwendbarer Komponenten; ggf. anpassen der Architektur (d.h. zurückgehen zu Schritt 1)
3. Gefundene Komponenten in die Entwicklungsversion übertragen
4. Anpassen (abändern) bzw. ggf. Konfigurieren der übernommenen Komponenten

Dieser Ablauf ersetzt entsprechende Arbeitsschritte zur Gewinnung von Architektur und/oder Quelltexten im klassischen Phasenmodell.

Anmerkungen zu Schritt 1: Wiederverwendbare Komponenten setzen i.d.R. bestimmte architektonische Strukturen und Konventionen voraus, z.B. die konkreten Typen von Parametern oder Konventionen bei der Fehlerbehandlung. Im Extremfall wird durch ein Framework das Hauptprogramm komplett vorgegeben, d.h. hier werden die Grobarchitektur, Teile der Detailarchitektur und die Implementierung diverser Framework-Komponenten wiederverwendet.

Die Architektur des zu entwickelnden Systems muß an derartige architektonische Vorgaben angepaßt werden, im Falle von reinen Bibliotheken z.B. durch eine Adaptionsschicht.

Anmerkungen zu Schritt 2: Wie der Vorrat wiederverwendbarer Komponenten bestimmt und technisch verwaltet wird, werden wir im Abschnitt über Repositories ausführlicher behandeln.

[1] Bei datenbankgestützten Informationssystemen sind die Programme immer im Zusammenhang mit den Datenbankschemata zu sehen.

Anmerkungen zu Schritt 4: Sofern eine Komponente völlig unverändert wiederverwendet wird, entfällt dieser Schritt, und man spricht von einer **black-box-Wiederverwendung**, andernfalls von einer **white-box-Wiederverwendung**.

2.2 Kostenanalyse

Wir erinnern uns jetzt daran, daß wir durch Wiederverwendung eigentlich Kosten sparen wollen. Welche Kosten entstehen nun beim vorstehenden Entwicklungsprozeß im Vergleich zu einer klassischen Neuentwicklung?

Schritt 1: wahrscheinlich kein großer Unterschied

Schritt 2: Ohne besondere Vorbereitungen kann die Suche nach einer Komponente problematisch sein.

Als Beispiel nehmen wir an, ein Entwickler erinnere sich, daß in einem früheren Projekt einmal eine bestimmte Funktion realisiert worden ist. Sofern nun keine ausreichende Dokumentation vorhanden ist, kann es sehr schwierig sein, die entsprechenden Operationen oder ggf. sogar nur Abschnitte im Quellcode zu finden.

Zunächst einmal sind überhaupt die entsprechenden Dateien zu finden; sie können sich off-line auf einem Archivierungsmedium befinden, z.B. einer CD, oder in einem Versionsarchiv.

Sodann ist eine einfache Stichwortsuche, wie sie mit allgemein verfügbaren Textprozessoren möglich ist, problematisch, da vermutlich viele unzutreffende Stellen, die alle manuell aussortiert werden müssen, gefunden werden.

Selbst dann, wenn die meisten unzutreffenden Fundstellen rasch aussortiert werden können, kann es sein, daß ein oder mehrere von ihnen genauer dahingehend untersucht werden müssen, ob sie tatsächlich für den Einsatzzweck geeignet sind. Dies beinhaltet das Lesen von Spezifikationen oder ggf. sogar Analysieren von Quellcode. Neben der Analyse der Fundstelle ist noch einzuschätzen, welchen Aufwand eventuelle Änderungen und Anpassungen erforderlich machen.

Schritt 3: Die Übernahme der Komponente in die Entwicklungsversion kann aufwendig sein, wenn es sich nicht um ein abgeschlossenes Stück Code handelt, sondern um verstreute Codefragmente.

Schritt 4: Selbst nachdem die richtige Datei oder der richtige Abschnitt darin gefunden worden sind, kann es notwendig sein, den Code an vielen Stellen manuell an den neuen Kontext anzupassen (white-box-Wiederverwendung), wobei das Risiko besteht, Fehler einzuführen. Es ist also ein anschließender Test der Komponente erforderlich. Das Testen macht i.a. einen signifikanten Anteil an den gesamten Entwicklungsaufwänden aus, dementsprechend verringert sich die Kosteneinsparung durch die Wiederverwendung.

Wenn man eine Komponente, deren Neuentwicklung 15 Stunden dauern würde, nach einer halben Stunde findet und unverändert übernehmen kann, hat man den Aufwand um den Faktor 30 reduziert. Zu Beginn der Suche weiß man aber nicht, ob man fündig wird. Wenn nur jede dritte Suche erfolgreich ist, reduziert man den Aufwand immerhin noch um den Faktor 10.

Wesentlich ungünstiger stellt sich die Situation dar, wenn in unserem Beispiel die Suche länger dauert, weil Fundstellen analysiert werden müssen (z.B. 2 Stunden) und Änderungen an der Komponente erforderlich sind (6 Stunden incl. Test). Wenn im Durchschnitt 2 vergebliche Suchvorgänge auf einen erfolgreichen kommen, würde der Aufwand nur noch (aber immerhin) um 20 % reduziert. Zugleich ist die Risikostruktur deutlich ungünstiger geworden: einer eventuell verlorenen Zeitinvestition von 2 Stunden steht ein möglicher Gewinn von 8 Stunden bei der Realisierung gegenüber. Bei einem Mißerfolg verlängert sich die Bearbeitungszeit von 15 auf 17 Stunden.

Aus dieser groben Kostenanalyse können wir bereits zwei Schlußfolgerungen zur Pragmatik der Wiederverwendung ziehen:

- Die Wiederverwendung sehr kleiner Komponenten (z.B. kurze Operationen mit ca. 10 - 30 Zeilen Code) lohnt i.a. nicht, wenn diese mit wenig Aufwand (ca. eine Stunde) neu erstellt werden können.

- Je umfangreicher Komponenten sind, desto mehr Entwurfsentscheidungen und Funktionen beinhalten sie und umso geringer wird die Wahrscheinlichkeit, daß die Komponente unverändert zum aktuellen Bedarf paßt. Umfang und Wahrscheinlichkeit von Änderungen steigen stark an, sofern nicht die zu konstruierenden Systeme sehr ähnlich sind.

2.3 Geplante Wiederverwendung

Das vorstehende Kostenanalyse zeigt, daß Wiederverwendung vor allem dann erfolgreich ist, wenn

- die Suche nach Komponenten rasch vonstatten geht,
- die einzelnen Komponenten leicht dahingehend beurteilt werden können, ob sie für den angedachten Zweck geeignet sind,
- die gefundenen bzw. ausgewählten Komponenten unverändert wiederverwendet werden können.

Es liegt nahe, durch gezielte Vorbereitungen diese Voraussetzungen zu verbessern; in diesem Fall spricht man von **geplanter Wiederverwendung**. Denkbare vorbereitende Maßnahmen sind:

- Sammlung aller potentiell wiederverwendbaren Komponenten in einem (logisch) zentralen Repository
- die zusätzliche (also nicht zur normalen Entwicklungsdokumentation gehörige) Beschreibung potentieller Komponenten anhand eines Klassifikationsschemas und durch Schlagworte. Ein Klassifikationsschema muß i.d.R. speziell für den jeweiligen Anwendungskontext und Komponentenfundus entwickelt werden.

  Ferner sollte die Suche durch ein geeignetes Retrieval-System unterstützt werden.
- erhöhte Qualität der technischen Dokumentation der Komponenten, der Strukturierung und Lesbarkeit des Programmcodes usw.
- andere Gestaltung der Funktionalität und/oder Schnittstellen der Komponenten, z.B. Verallgemeinerung der Funktionalität, obwohl

diese im ursprünglichen Anwendungskontext nicht so allgemein benötigt wurde.
- besonders sorgfältiger Test der Komponenten, um das Vertrauen der Entwickler und die Akzeptanz zu erhöhen.

Diese Vorbereitungen kosten ihrerseits Aufwand; dieser Vorbereitungsaufwand hängt natürlich von diversen Umständen ab, liegt aber nach Aussagen diverser Quellen im Bereich von 30 - 60 % des normalen Entwicklungsaufwands[2].

Einer Faustregel zufolge amortisieren sich die Mehraufwände erst nach nach 3 Wiederverwendungen, d.h. erst bei noch häufigerer Wiederverwendung kommt es zu Kosteneinsparungen.

3 Repositories

3.1 Funktionsbereiche von Repositories

Unter einem (Komponenten- bzw. Wiederverwendungs-) **Repository** verstehen wir hier ein System, das wiederverwendbare Komponenten verwalten kann und das die Suche nach geeigneten Komponenten unterstützt. Ein Repository muß hierzu Funktionen in folgenden Bereichen anbieten:

Verwaltung deskriptiver Daten zu den Komponenten: Die deskriptiven Daten können vom Typ der Komponenten abhängen. Bei einem Programm könnten sie beinhalten: die Programmiersprache, Compilerversion, importierte Bibliotheken, Autor, Erstellungsdatum, Stichworte zum Inhalt, Klassifizierung usw.; bei einem Datenbankschema: DDL-Version, zug. Datenbank(en), Autor, Stichworte usw.

[2] Bei einer Aufwandsschätzung nach der Function-Point-Methode wird dagegen von einer Erhöhung des Gesamtaufwands eines Projekts von nur ca. 5 % infolge geplanter Wiederverwendung ausgegangen. Hierbei ist allerdings der spezielle Anwendungskontext zu berücksichtigen und von der Annahme auszugehen, daß nur derjenige Teil der entwickelten Software für eine Wiederverwendung präpariert wird, bei dem dies besonders aussichtsreich erscheint.

Statt von deskriptiven Daten redet man hier oft von **Metadaten** ("Daten über Daten"). Bei Datenbankschemata stimmt diese Begriffsbildung sogar mit der üblichen überein[3].

Pflege der Klassifikationsschemata: Funktionen für Aufbau und Weiterentwicklung; zugehörige Statistikfunktionen

Suche nach Komponenten: Basis hierfür sind i.d.R. alleine die deskriptiven Daten, nicht die Komponenten selbst. Sinnvoll sind oft dokumenttypspezifische Vergleichsoperatoren (z.B. "ist zum Interface XYZ aufwärtskompatibel" für Programmodule).

Es handelt sich hier um eine vage Suche, wie sie für Information-Retrieval-Systeme typisch ist, d.h. es wird nicht nach einem exakt bestimmten, sondern nach einem möglichst brauchbaren Element gesucht, und die Suchkriterien werden iterativ verfeinert und modifiziert.

Import von Komponenten aus Dateien in bestimmten Formaten bzw. Projektdatenbanken: Edierfunktionen zur direkten Eingabe von Komponenten sind i.a. nicht erforderlich, man kann davon ausgehen, daß die Komponenten bereits fertig entwickelt vorliegen, die Unterstützung der Entwicklung von Komponenten ist kein direktes Ziel eines Wiederverwendungsrepositorys.

Export von Komponenten. Manche Repository-Systeme verwalten die Komponenten nicht selbst, sondern nur Verweise auf Bezugsquellen. In diesem Fall entfallen die Import- und Export-Funktionen.

Nachweis von Wiederverwendungen: Erfassung von erfolglosen Analysen und erfolgreichen Wiederverwendungen. Für einen Entwickler, der im Repository gesucht hat, können für jeden angezeigten Treffer folgende Informationen wertvoll sein:

- Wie oft ist diese Komponente schon bei einer Suche gefunden worden?
- Wer hat sich diese Komponente schon einmal angesehen und

[3] Bei Programmen und anderen Arten von Komponenten paßt die Begriffsbildung nicht ganz exakt; hier hilft einem die Vorstellung, daß auch diese Artefakte irgendwie elektronisch gespeichert und dadurch zu Daten werden.

analysiert (wovon man profitieren könnte)?

In diesem Zusammenhang sollten Entwickler Komponenten bewerten können, d.h. jedesmal, wenn eine Komponente gefunden und von einem Entwickler analysiert wird, kann (und sollte) der Entwickler seine Meinung über die Komponente im System hinterlegen, so daß diese Arbeitsleistung später für andere Interessenten ohne weitere Rückfragen ausgenutzt werden kann.

- In welchen Systemen ist diese Komponente wiederverwendet worden? Welche anderen Komponenten wurden dort ebenfalls wiederverwendet?

Nutzer- und Rechteverwaltung, d.h. Kontrolle, wer das Repository wie nutzen darf. Bei offenen, im WWW frei zugänglichen Repositories fallen hierunter ggf. vorhandene Registrierungsprozeduren.

Abrechnungsfunktionen abhängig von den rechtlichen Rahmenbedingungen: hierunter fallen die Erfassung von kostenpflichtigen Nutzungen und ggf. zusätzlichen abrechnungsrelevanten Daten sowie ggf. Funktionen zur rechtsverbindlichen Bestellung und Zahlungsabwicklung.

Einige dieser Funktionen (z.B. Import, Wartung der Klassifikationsschemata, Rechteverwaltung) kann man einem Administrator vorbehalten; dies entspricht der Auffassung, daß die Sammlung wiederverwendbarer Komponenten ein wertvolles Gut ist, das dem Unternehmen gehört und dementsprechend betreut werden muß. Eine alternative Auffassung besteht darin, das Wiederverwendungsrepository als eine offene Austauschplattform anzusehen, in der jeder Entwicklung nach eigenem Ermessen Komponenten der Allgemeinheit anbieten kann. Wie restriktiv die Verwaltung eines Repositorys gehandhabt wird, muß abhängig vom konkreten Kontext entschieden werden.

3.2 Autarke vs. integrierte Repositories

Die im vorigen Abschnitt genannten Funktionsbereiche können recht verschieden ausgeprägt sein, ferner kommen bei vielen Repository-Systemen weitere wesentliche Funktionsbereiche hinzu. In diesem Zusammenhang ist es sinnvoll, zwei Arten von Repositories nach ihrem Verwendungszweck und Anwendungskontext zu unterscheiden:

autarkes Repository: Ein Beispiel hierfür wäre eine Sammlung von Quellprogrammen in einer bestimmten Sprache zu verschiedenen Problembereichen, die z.B. über eine WWW-Schnittstelle durchsucht und abgerufen werden können. Es besteht hier eine klare Trennung zwischen Anbieter und Käufer / Nutzer von Komponenten, ferner ist das Repository technisch getrennt von der Entwicklungsumgebung der Nutzer der Komponenten.

in eine SEU integriertes Repository bei einem Unternehmen mit wiederverwendungsorientierten Entwicklungsprozessen: Hier spielt das Repository eine zentrale Rolle bei der Anwendungsentwicklung und wird von den Entwicklern intensiv bei täglichen Arbeit genutzt. Neben der Wiederverwendung sollen hier mit einem Repository weitere strategische Ziele erreicht werden:

- Anwendungen zu dokumentieren
- Anwendungen inhaltlich zu integrieren
- Doppelentwicklungen zu vermeiden
- Auswirkungen von Änderungen einzuschätzen

3.3 Weitere adressierte Problembereiche

Wir haben Repositories bisher motiviert durch den Problembereich Wiederverwendung. Dieser Problembereich ist aber je nach Einsatzbedingungen nicht zu trennen von anderen Problembereichen; daher bieten manche Repository-Systeme, speziell bei in eine SEU integrierten Repositories, auch Funktionen für andere Bereiche an.

3.3.1 Versions- und Konfigurationsmanagement

Von wiederverwendbaren Komponenten können genau wie von beliebigen anderen Komponenten Versionen (im Sinne von Revisionen) existieren, ferner können wiederverwendbare Komponenten Konfigurationen bilden. Derartige Versions- und Konfigurationsbeziehungen zwischen Komponenten sollten durchaus im Repository repräsentiert werden. Selbst dann stellen Repositories keinen Ersatz für die Versionsarchive von VM/KM-Systemen dar:

er

- Versionsarchive enthalten i.d.R. sehr viele vorläufige, ggf. sogar inkonsistente Zwischenversionen. Diese Versionen sind nicht zur Wiederverwendung gedacht und geeignet. Der für die Eintragung in das Repository erforderliche zusätzliche Dokumentationsaufwand ist hier nicht gerechtfertigt. Schließlich würde die Vielzahl ähnlicher Treffer in den Suchergebnissen eher stören.
- Wiederverwendungsrepositories sind ihrer Natur nach eher eine zentrale Ressource für eine größere Entwicklergruppe. Versionsarchive können dagegen für jeden Entwickler oder einzelne Entwicklergruppen dezentral eingerichtet werden und bieten daher eine bessere Ausfallsicherheit und ggf. Performance.

3.3.2 Reverse Engineering

Unter **Reverse Engineering** versteht man die Gewinnung von Modellen bzw. Abstraktionen der frühen Entwicklungsphasen und ggf. technischer Dokumentation für ein Programm, für das diese Dokumente nicht vorhanden sind. Insb. soll also die architektonische Struktur und ggf. das vom Programm gelöste Problem erkannt werden. Es gibt eine Vielzahl von Reverse-Engineering-Werkzeugen, die automatisch oder teilweise interaktiv entsprechende Modelle bzw. Strukturdarstellungen erzeugen können. Manche Reverse-Engineering-Werkzeuge können ausgehend von Datenbank-Schemata oder Typdeklarationen in Programmen Datenmodelle rekonstruieren und Nachweise ertellen, in welchen Applikationen ein bestimmtes Datenfeld benutzt wird.

Im Vergleich zum lehrbuchmäßigen Vorgehen, das hier auch als *Forward Engineering* bezeichnet wird, wird beim Reverse Engineering die Entwicklungsrichtung umgekehrt. Typische Ursachen für das Fehlen der Dokumente aus den frühen Phasen sind: Sie wurden nie erstellt oder gingen verloren; sie wurden erstellt, sind aber nicht mehr gültig, weil die Programme später ohne korrespondierende Korrekturen in den Modellen abgeändert wurden.

Unter **Reengineering** versteht man die Renovierung bzw. Strukturverbesserung von Programmen, ggf. auf Basis eines vorhergehenden Reverse Engineering.

Anlaß für das Reverse Engineering eines Programms ist meist, daß es geändert werden muß oder ein Reengineering erfolgen soll und daß man hierzu seine Struktur, Funktionen und Zwecke verstehen muß. Das Verstehen und Beurteilen von Programmen ist aber, wie wir oben gesehen habe, ein ganz entscheidendes Problem bei der Wiederverwendung. Insofern sind die Leistungen von Reverse-Engineering-Werkzeugen auch für die Wiederverwendung nützlich. Unterschiede bestehen, wenn überhaupt, darin, daß im Kontext der Wiederverwendung vor allem solche Daten gesucht sind, die bei der Suche als Selektionskriterien verwendbar sind.

Durch Reverse-Engineering-Werkzeuge kann u.U. ein erheblicher Teil der zusätzlichen Dokumentation wiederverwendbarer Komponenten, die sonst manuell zu erstellen wäre, automatisch gewonnen werden. Auf diese Weise kann die globale Kosten-Nutzen-Relation deutlich verbessert werden.

3.3.3 Datenaustausch zwischen Werkzeugen

Sofern bei einem größeren Anwender mehrere Werkzeuge unterschiedlicher Herkunft eingesetzt werden, entsteht das Problem der Datenintegration dieser Werkzeuge bzw. des Dokumentaustauschs zwischen ihnen. Lösungsansätze sind u.a. Transportdateien und zug. Formate (vgl. [SEU, IRA], ferner [XML]).

Sofern weiter das Repository die Komponenten direkt verwaltet und über Import- und Export-Funktionen von und zu den jeweiligen

Werkzeugen verfügt, kann es für die Datenintegration der Werkzeuge eingesetzt werden. Das Repository spielt hier eine ähnliche Rolle wie Transportdateien (z.B. im CDIF-Format).

3.4 Abgrenzung zu anderen Systemen

Mit "Repository" (auf Deutsch: "Speicher") werden vielfach auch signifikant andere Systeme bezeichnet, die nur lose mit dem hier adressierten Problemkomplex zu tun haben:

Versionsarchive von VM/KM-Systemen: In VM/KM-Systemen (z.B. CVS) versteht man unter einem Repository bestimmte Dateien und Verzeichnisse, die i.w. Versionen von Quellprogrammen und anderen (primär textuellen) Dokumenten, die im Verlauf eines Projekts entstanden sind, sowie in geringem Umfang begleitende Dokumentation enthalten. Ein Repository ist hier insofern eine Projektdatenbank, als es die Komponenten enthält, die für ein bestimmtes Projekt benötigt werden bzw. darin entwickelt wurden; diese Komponenten sind normalerweise nicht wiederverwendbar (vgl. Abschnitt 3.3.1). Die Suche nach Komponenten und die dazugehörige Dokumentation werden nicht unterstützt.

dedizierte DBMS: Mit Repository werden diverse dedizierte DBMS (Beispiele: IRDS, PCTE, MS-Repository) bezeichnet, die aber analog zu den Repositories von VM/KM-Systemen eher als Projektdatenbanken bzw. deren Managementsysteme anzusehen sind. Sofern sie geeignete Suchfunktionen anbieten, eignen sie sich als reine Datenverwaltungskomponente innerhalb eines Wiederverwendungsrepositorys.

Data-Dictionary-Systeme: Diese dokumentieren primär Datenelementtypen in Datenbanken. Ein Schwerpunkt ist hier die Bereitstellung dieser Metadaten zur Laufzeit der Programme auf dem Produktionsrechner. Komponenten-Repositories stellen derartige Daten hingegen während der Entwicklung in einer Entwicklungsumgebung zur Verfügung. Fallweise kann es sinnvoll sein, beide Funktionsbereiche gemeinsam zu behandeln.

Literatur

[IRA] Kelter, U.: Lehrmodul “Integrationsrahmen für Software-Entwicklungsumgebungen”; 1999/11

[SEU] Kelter, U.: Lehrmodul “Software-Entwicklungsumgebungen”; 2001/02

[XML] Kelter, U.: Lehrmodul “Transportdateien und die SGML”; 2001

Index